Linux+™

Study Guide

Roderick W. Smith

San Francisco • Paris • Düsseldorf • Soest • London

SYBEX

Associate Publisher: Neil Edde
Acquisitions and Developmental Editor: Elizabeth Hurley
Editors: Rebecca Rider, Susan Berge, Jim Gabbert
Production Editor: Shannon Murphy
Technical Editor: Matthew Miller
Book Designer: Bill Gibson
Graphic Illustrator: Tony Jonick
Electronic Publishing Specialist: Nila Nichols
Proofreaders: Emily Hsuan, Nelson Kim, Laurie O'Connell, Yariv Rabinovitch, Suzanne Stein
Indexer: Ann Rogers
CD Coordinator: Christine Harris
CD Technician: Kevin Ly
Cover Designer: Archer Design
Cover Photograph: Natural Selection

Library of Congress Card Number: 2001089831

ISBN: 0-7821-2939-0

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

To Our Valued Readers:

Sybex is proud to have served as a member of CompTIA's Linux+ Advisory Committee. Just as CompTIA is committed to establishing measurable standards for certifying individuals who will support Linux systems in the future, Sybex is committed to providing those individuals with the skills needed to meet those standards. By working alongside CompTIA, and in conjunction with other esteemed members of the Linux+ committee, it is our desire to help bridge the knowledge and skills gap that currently confronts the IT industry.

Sybex expects the Linux+ program to be well received, both by companies seeking qualified technical staff and by the IT training community. Along with the existing line of vendor-neutral certifications from CompTIA, including A+, Network+, Server+, and i-Net+, the Linux+ certification should prove to be an invaluable asset in the years ahead.

Our authors and editors have worked hard to ensure that this *Linux+ Study Guide* is comprehensive, in-depth, and pedagogically sound. We're confident that this book will meet and exceed the demanding standards of the certification marketplace and help you, the Linux+ exam candidate, succeed in your endeavors.

Good luck in pursuit of your Linux+ certification!

Neil Edde
Associate Publisher—Certification
Sybex, Inc.

SYBEX Inc. 1151 Marina Village Parkway, Alameda, CA 94501
Tel: 510/523-8233 Fax: 510/523-2373 HTTP://www.sybex.com

Software License Agreement: Terms and Conditions

The media and/or any online materials accompanying this book that are available now or in the future contain programs and/or text files (the "Software") to be used in connection with the book. SYBEX hereby grants to you a license to use the Software, subject to the terms that follow. Your purchase, acceptance, or use of the Software will constitute your acceptance of such terms.

The Software compilation is the property of SYBEX unless otherwise indicated and is protected by copyright to SYBEX or other copyright owner(s) as indicated in the media files (the "Owner(s)"). You are hereby granted a single-user license to use the Software for your personal, noncommercial use only. You may not reproduce, sell, distribute, publish, circulate, or commercially exploit the Software, or any portion thereof, without the written consent of SYBEX and the specific copyright owner(s) of any component software included on this media.

In the event that the Software or components include specific license requirements or end-user agreements, statements of condition, disclaimers, limitations or warranties ("End-User License"), those End-User Licenses supersede the terms and conditions herein as to that particular Software component. Your purchase, acceptance, or use of the Software will constitute your acceptance of such End-User Licenses.

By purchase, use or acceptance of the Software you further agree to comply with all export laws and regulations of the United States as such laws and regulations may exist from time to time.

Software Support

Components of the supplemental Software and any offers associated with them may be supported by the specific Owner(s) of that material but they are not supported by SYBEX. Information regarding any available support may be obtained from the Owner(s) using the information provided in the appropriate read.me files or listed elsewhere on the media.

Should the manufacturer(s) or other Owner(s) cease to offer support or decline to honor any offer, SYBEX bears no responsibility. This notice concerning support for the Software is provided for your information only. SYBEX is not the agent or principal of the Owner(s), and SYBEX is in no way responsible for providing any support for the Software, nor is it liable or responsible for any support provided, or not provided, by the Owner(s).

Warranty

SYBEX warrants the enclosed media to be free of physical defects for a period of ninety (90) days after purchase. The Software is not available from SYBEX in any other form or media than that enclosed herein or posted to www.sybex.com. If you discover a defect in the media during this warranty period, you may obtain a replacement of identical format at no charge by sending the defective media, postage prepaid, with proof of purchase to:

SYBEX Inc.
Customer Service Department
1151 Marina Village Parkway
Alameda, CA 94501
(510) 523-8233
Fax: (510) 523-2373
e-mail: info@sybex.com
WEB: HTTP://WWW.SYBEX.COM

After the 90-day period, you can obtain replacement media of identical format by sending us the defective disk, proof of purchase, and a check or money order for $10, payable to SYBEX.

Disclaimer

SYBEX makes no warranty or representation, either expressed or implied, with respect to the Software or its contents, quality, performance, merchantability, or fitness for a particular purpose. In no event will SYBEX, its distributors, or dealers be liable to you or any other party for direct, indirect, special, incidental, consequential, or other damages arising out of the use of or inability to use the Software or its contents even if advised of the possibility of such damage. In the event that the Software includes an online update feature, SYBEX further disclaims any obligation to provide this feature for any specific duration other than the initial posting.

The exclusion of implied warranties is not permitted by some states. Therefore, the above exclusion may not apply to you. This warranty provides you with specific legal rights; there may be other rights that you may have that vary from state to state. The pricing of the book with the Software by SYBEX reflects the allocation of risk and limitations on liability contained in this agreement of Terms and Conditions.

Shareware Distribution

This Software may contain various programs that are distributed as shareware. Copyright laws apply to both shareware and ordinary commercial software, and the copyright Owner(s) retains all rights. If you try a shareware program and continue using it, you are expected to register it. Individual programs differ on details of trial periods, registration, and payment. Please observe the requirements stated in appropriate files.

Copy Protection

The Software in whole or in part may or may not be copy-protected or encrypted. However, in all cases, reselling or redistributing these files without authorization is expressly forbidden except as specifically provided for by the Owner(s) therein.

In memory of Douglas Adams, 1952–2001. So long, and thanks for all the laughter.

Acknowledgments

A book doesn't just happen. At every point along the way from project conception to finished product, many people other than the author have their influence. Elizabeth Hurley, the Acquisitions and Developmental Editor, helped guide the book's development, especially for the critical first few chapters. Shannon Murphy, as Production Editor, coordinated the work of the many others who contributed their thoughts to the book. Rebecca Rider, the Editor, provided suggestions and helped keep the prose readable. The team of technical editors scrutinized the text for technical errors, and to be sure its coverage was complete. Also, my thanks go to Emily Hsuan, Nelson Kim, Laurie O'Connell, Yariv Rabinovitch, and Suzanne Stein, the Proofreaders for this book; Nila Nichols, the Electronic Publishing Specialist; and to the entire CD team at Sybex for working together to produce the final product. I'd also like to thank Neil Salkind at Studio B; as my agent, he helped connect me with Sybex to write this book.

Introduction

Why should you learn about Linux? It's a fast-growing operating system, and it is inexpensive and flexible. Linux is also a major player in the small and mid-sized server field, and it's an increasingly viable platform for workstation and desktop use, as well. By understanding Linux, you'll increase your standing in the job market. Even if you already know Windows or MacOS and your employer uses these systems exclusively, understanding Linux will give you an edge when you are looking for a new job or if you are looking for promotion. For instance, this knowledge will allow you to make an informed decision about if and when you should deploy Linux.

The Computing Technology Industry Association (CompTIA) has developed its Linux+ exam as an introductory certification for people who want to enter careers involving Linux. The exam is meant to certify that an individual has the skills necessary to install, operate, and troubleshoot a Linux system, and is familiar with Linux-specific concepts and basic hardware.

The purpose of this book is to help you pass the Linux+ exam. Because this exam covers basic Linux installation, use, configuration, administration, and hardware interactions, those are the topics that are emphasized in this book. You'll learn enough to get a Linux system up and running and how to configure it for many common tasks. Even after you've taken and passed the Linux+ exam, this book should remain a useful reference.

What Is Linux?

Linux is a clone of the Unix OS that has been popular in academia and many business environments for years. Formerly used exclusively on large mainframes, Unix and Linux can now run on small computers—which are actually far more powerful than the mainframes of just a few years ago. Because of its mainframe heritage, Unix (and hence also Linux) scales well to perform today's demanding scientific, engineering, and network server tasks.

Linux consists of a kernel, which is the core control software, and many libraries and utilities that rely upon the kernel to provide features with which users interact. The OS is available in many different distributions, which are bundlings of a specific kernel with specific support programs. These concepts are discussed at greater length in Chapters 1–3.

Why Become Linux+ Certified?

There are several good reasons to get your Linux+ certification. The CompTIA Candidates Information packet lists five major benefits:

Provides proof of professional achievement Certifications are quickly becoming status symbols in the computer service industry. Organizations, including members of the computer service industry, are recognizing the benefits of certification, such as Linux+ or A+. Organizations are pushing for their members to become certified. Every day, more people are putting the CompTIA official certification logo on their business cards.

Increases your marketability Linux+ certification makes individuals more marketable to potential employers. Also, the Linux+ certified employees might receive a higher salary base because employers won't have to spend as much money on vendor-specific training.

Provides an opportunity for advancement Most raises and advancements are based on performance. Linux+ certified employees work faster and more efficiently. The more productive employees are, the more money they will make for their company. And, of course, the more money they make for the company, the more valuable they will be to the company. So, if employees are Linux+ certified, their chances of getting promoted will be greater.

Fulfills training requirements Each year, more and more major computer hardware vendors, including (but not limited to) IBM, Hewlett-Packard, and Compaq, are recognizing CompTIA's certifications as prerequisites in their own respective certification programs. The use of outside certifications like Linux+ has the side benefit of reducing training costs for employers. Because more and more small companies are deploying the flexible and inexpensive OS we call Linux, the demand for experienced users is growing. CompTIA anticipates that the Linux+ exam, like the A+ exam, will find itself integrated into various certification programs as well.

Raises customer confidence As the IT community, users, small business owners, and the like become more familiar with the Linux+ certified professional moniker, more of them will realize that the Linux+ professional is more qualified to work in their Linux environment than is a non-certified individual.

How to Become Linux+ Certified

The Linux+ certification is available to anyone who passes the test. You don't have to work for a particular company. It's not a secret society. It is, however, an elite group.

The exam is administered by Prometric and can be taken at any Prometric Testing Center. If you pass, you will get a certificate in the mail from CompTIA saying that you have passed, and you will also receive a lapel pin and business cards. To find the Prometric training center nearest you, call (800) 755-EXAM (755-3926).

To register for the exam, call Prometric at (800) 776-MICRO (776-4276) or register online at `http://www.2test.com`. You'll be asked for your name, your Social Security number (an optional number may be assigned if you don't wish to disclose your Social Security number), mailing address, phone number, employer, when and where you want to take the test (i.e., which Prometric testing center), and your credit card number (arrangement for payment must be made at the time of registration).

Who Should Buy This Book

Anybody who wants to pass the Linux+ exam may benefit from this book. If you're new to Linux, this book covers the material you will need to learn the OS from the beginning, and it continues to provide the knowledge you need up to a proficiency level sufficient to pass the Linux+ exam. You can pick up this book and learn from it even if you've never used Linux before, although you'll find it an easier read if you've at least casually used Linux for a few days. If you're already familiar with Linux, this book can serve as a review and as a refresher course for information with which you might not be completely familiar. In either case, reading this book will help you to pass the Linux+ exam.

This book is written with the assumption that you know at least a little bit about Linux (what it is, and possibly a few Linux commands). This book also assumes that you know some basics about computers in general, such as how to use a keyboard, how to insert a floppy disk into a floppy drive, and so on. Chances are you have used computers in a substantial way in the past—perhaps even Linux, as an ordinary user, or maybe you have used Windows or MacOS. This book does *not* assume that you have extensive knowledge of Linux system administration, but if you've done some system administration, you can still use this book to fill in gaps in your knowledge.

How This Book Is Organized

This book consists of nine chapters plus supplementary information: a glossary, this Introduction, and the Assessment Test after the Introduction. The chapters are organized as follows:

- Chapter 1, "Planning the Implementation," covers things you should consider *before* you install Linux on a computer. This chapter compares Linux to other OSs, it discusses Linux's hardware requirements and its disk partition requirements, it describes the various Linux distributions, and it explores the software licenses found in the Linux world.
- Chapter 2, "Installing Linux," covers the Linux installation process. Because Linux is available in several variant forms, this chapter focuses on just one (Linux Mandrake 8.0), but other Linux distributions must perform the same fundamental tasks, so much of this information is directly applicable to other distributions. This chapter also covers the post-installation configuration of one particularly critical Linux component: the X Window System (or X for short), which provides Linux's GUI environment.
- Chapter 3, "Software Management," covers how to install and configure software. Much of this discussion is devoted to the two major package management systems in Linux, the Red Hat Package Manager (RPM) and Debian packages. This chapter also covers kernel issues and boot loaders (which are used to boot a Linux kernel).
- Chapter 4, "Users and Security," covers how to create and maintain user accounts; it also covers the security issues surrounding users and Linux more generally. Because Linux is a clone of Unix, it includes extensive support for multiple users, and understanding Linux's model for user accounts is critical to many aspects of Linux's operation.
- Chapter 5, "Networking," covers how to use Linux on a network. This chapter includes an overview of what a network is, including the popular TCP/IP networking tools upon which the Internet is built. Several popular Linux network client programs are discussed, as is the subject of how to control access to a Linux computer.

- Chapter 6, "Managing Files and Services," covers many of the important Linux configuration files and some miscellaneous administrative and user tasks, such as how you should use a GUI environment and how to write a shell script. Most of these tasks aren't very glamorous, but they're critically important for you to know if you want to keep a system running properly.
- Chapter 7, "Managing Partitions and Processes," covers two things: filesystems (disk partitions and the data they contain) and processes (running programs). Specific topics include how to create and manage filesystems, how to back up and restore a computer, how to run programs at specific scheduled times, and how to manipulate running processes.
- Chapter 8, "Hardware Issues," covers various hardware topics. These include configuring printers, using kernel modules (drivers for specific hardware devices), adding new hardware, using laptop computers, and diagnosing hardware problems. Some of these issues are the same as in other OSs, but Linux handles some hardware devices in fundamentally different ways than do many other OSs.
- Chapter 9, "Troubleshooting," is devoted to the question of what to do when things go wrong. This chapter includes information on how to narrow down the problem space to a manageable size, and it includes advice on how to proceed when you see many common problem symptoms.

Each chapter begins with a list of the CompTIA Linux+ objectives that are covered in that chapter. (The book doesn't cover objectives in the same order as CompTIA lists them, so don't be alarmed when you notice gaps in the sequence.) At the end of each chapter, there are several elements you can use to help prepare for the exam:

Exam Essentials This section summarizes important information that was covered in the chapter. You should be able to perform each of the tasks or convey the information requested.

Commands in This Chapter Most chapters include discussion of several Linux commands. (Chapter 1 is an exception to this rule.) You should be familiar with these commands before taking the exam. You might not need to know every option for every command, but you should know what the command does and be familiar with its major options. (Chapter 3 begins with a discussion of how to perform basic tasks in a Linux command shell.)

Key Terms The key terms are italicized throughout the text. They're important terms with which you should be familiar before you take the exam. The Glossary provides definitions for all of the key terms. They're also defined in the text in which they're first discussed extensively.

Review Questions Each chapter concludes with twenty review questions. You should answer these questions and check your answer against the one provided after the questions. If you can't answer at least 80 percent of these questions correctly, go back and review the chapter, or at least those sections that seem to be giving you difficulty.

The Review Questions, Assessment Test, and other testing elements included in this book are *not* derived from the CompTIA Linux+ exam questions, so don't memorize the answers to these questions and assume that doing this will let you pass the Linux+ exam. You should learn the underlying topic, as described in the text of the book. This will let you answer the questions provided with this book *and* pass the exam. Learning the underlying topic is also the approach that will serve you best in the workplace—the ultimate goal of a certification like Linux+.

To get the most out of this book, you should read each chapter from start to finish, then check your memory and understanding with the chapter-end elements. Even if you're already familiar with a topic, you should skim the chapter; Linux is complex enough that there are often multiple ways to accomplish a task, so you may learn something even if you're already competent in an area.

Bonus CD-ROM Contents

This book comes with a CD-ROM that contains both the book's features and several additional elements. Items available on the CD-ROM include the following:

Book contents as a PDF file The entire book is available as an Adobe Portable Document Format (PDF; aka Acrobat) file. This allows you to take the book with you on the road or use a PDF reader's search function to find a word or phrase you remember reading but can't quite find.

Electronic "flashcards" The CD-ROM includes 150 questions in "flashcard" format (a question followed by a single correct answer). You can use these to review your knowledge of the Linux+ exam objectives.

Sample Tests All of the questions in this book appear on the CD-ROM—both the 30-question Assessment Test at the end of this Introduction and the 180 questions that consist of the nine 20-question Review Question sections for each chapter. In addition, there are two 65-question Bonus Exams.

You can use a PDF reader like Adobe Acrobat or any Ghostscript-based viewer in Linux to read the PDF files on the CD-ROM. The sample tests use a Java applet that works with Java-enabled Web browsers in Linux, Windows, or other OSs. Look for a file called `test.htm` in the test engine directory on the CD-ROM and double-click it in a file browser, or load it using a file selector in your Web browser. Chapter 7, "Managing Partitions and Processes," discusses mounting disks, including CD-ROMs, if you want to access these files from Linux.

Conventions Used in This Book

This book uses certain typographic styles in order to help you quickly identify important information and to avoid confusion over the meaning of words such as on-screen prompts. In particular:

- *Italicized text* indicates key terms that are discussed at length for the first time in a chapter. (Italics are also used for emphasis.)
- A `monospaced font` is used to indicate the contents of configuration files, messages displayed at a text-mode Linux shell prompt, filenames, and Internet URLs.
- *`Italicized monospaced text`* indicates a variable—information that differs from one system or command run to another, such as the name of a client computer or a process ID number.

- **`Bold monospaced text`** is information that you're to type into the computer, usually at a Linux shell prompt. This text can also be italicized to indicate that you should substitute an appropriate value for your system. (When isolated on their own lines, commands are preceded by non-bold monospaced `$` or `#` command prompts.)

In addition to these text conventions, which can apply to individual words or entire paragraphs, there are a few conventions that I use to highlight segments of text:

A Note indicates information that's useful or interesting, but that's somewhat peripheral to the main discussion. A Note might be relevant to a small number of networks, for instance, or it may refer to an outdated feature.

A Tip provides information that can save you time or frustration and that may not be entirely obvious. A Tip might describe how to get around a limitation, or how to use a feature to perform an unusual task.

Warnings describe potential pitfalls or dangers. If you fail to heed a Warning, you may end up spending a lot of time recovering from a bug, or you may even end up restoring your entire system from scratch.

Sidebars

A Sidebar is like a Note but is longer. Typically, a Note is one paragraph or less in length, but Sidebars are longer than this. The information in a Sidebar is useful, but it doesn't fit into the main flow of the discussion.

Real World Scenario

Real World Scenario

A Real World Scenario is a type of sidebar that describes some task or example that's particularly grounded in the real world. This may be a situation I or somebody I know has encountered, or it may be advice on how to work around problems that are common in real, working Linux environments.

The Exam Objectives

Behind every computer industry exam you can be sure to find exam objectives—the broad topics in which exam developers want to ensure your competency. The official CompTIA objectives for the Linux+ exam are listed here.

Exam objectives are subject to change at any time without prior notice and at CompTIA's sole discretion. Please visit the Linux+ Certification page of CompTIA's Web site (`http://www.comptia.com/certification/linuxplus/index.htm`) for the most current listing of exam objectives.

Domain 1.0 Planning the Implementation

1.1 Identify purpose of Linux machine based on predetermined customer requirements (e.g., appliance, desktop system, database, mail server).

1.2 Identify all system hardware required and validate that it is supported by Linux (e.g., CPUs, RAM, graphics cards, storage devices, network interface cards, modem).

1.3 Determine what software and services should be installed (e.g., client applications for workstation, server services for desired task), check requirements and validate that it is supported by Linux.

1.4 Determine how storage space will be allocated to file systems (e.g., partition schemes).

1.5 Compare and contrast how major Linux licensing schemes work (e.g., GNU/GPL, freeware, shareware, open source, closed source, artistic license).

1.6 Identify the function of different Linux services (e.g., Apache, Squid, SAMBA, Sendmail, `ipchains`, BIND).

1.7 Identify strengths and weaknesses of different distributions and their packaging solutions (e.g., tar ball vs. RPM/DEB).

1.8 Describe the functions, features, and benefits of Linux solutions as compared with other operating systems (e.g., Linux players, distributions, available software).

1.9 Identify how the Linux kernel version numbering works.

1.10 Identify where to obtain software and resources.

1.11 Determine customer resources for a solution (e.g., staffing, budget, training).

Domain 2.0 Installation

2.1 Determine appropriate method of installation based on the environment (e.g., boot disk, CD-ROM, Network (HTTP, FTP, NFS, SMB)).

2.2 Describe the different types of Linux installation interaction and determine which to use for a given situation (e.g., GUI, text, network).

2.3 Select appropriate parameters for Linux installation (e.g., language, time zones, keyboard, mouse).

2.4 Select packages based on the machine's "role" (e.g., Workstation, Server, Custom).

2.5 Select appropriate options for partitions based on pre-installation choices (e.g., FDISK, third party partitioning software).

2.6 Partition according to your pre-installation plan using fdisk (e.g., `/boot`, `/`, `/usr`, `/var/home`, `SWAP`).

2.7 Configure file systems (e.g., (ext2) or (ext3) or REISER).

2.8 Select appropriate networking configuration and protocols (e.g., modems, Ethernet, Token-Ring).

2.9 Select appropriate security settings (e.g., Shadow password, root password, umask value, password limitations and password rules).

2.10 Create users and passwords during installation.

2.11 Install and configure XFree86 server.

2.12 Select Video card support (e.g., chipset, memory, support resolution(s)).

2.13 Select appropriate monitor manufacturer and settings (e.g., custom, vertical, horizontal, refresh).

2.14 Select the appropriate window managers or desktop environment (e.g., KDE, GNOME).

2.15 Explain when and why the kernel will need to be recompiled.

2.16 Install boot loader (e.g., LILO, MBR vs. first sector of boot partition).

2.17 Install and uninstall applications after installing the operating system (e.g., RPM, `tar`, `gzip`).

2.18 Read the Logfiles created during installation to verify the success of the installation.

2.19 Validate that an installed application is performing correctly in both a test and production environment.

Domain 3.0 Configuration

3.1 Reconfigure the Xwindow System with automated utilities (e.g., Xconfigurator, XF86Setup).

3.2 Configure the client's workstation for remote access (e.g., ppp, ISDN).

3.3 Set environment variables (e.g., `PATH`, `DISPLAY`, `TERM`).

3.4 Configure basic network services and settings (e.g., `netconfig`, `linuxconf`; settings for TCP/IP, DNS, DHCP).

3.5 Configure basic server services (e.g., X, SMB, NIS, NFS).

3.6 Configure basic Internet services (e.g., HTTP, POP, SMTP, SNMP, FTP).

3.7 Identify when swap space needs to be increased.

3.8 Add and configure printers.

3.9 Install and configure add-in hardware (e.g., monitors, modems, network interfaces, scanners).

3.10 Reconfigure boot loader (e.g., LILO).

3.11 Identify the purpose and characteristics of configuration files (e.g., BASH, `inittab`, `fstab`, `/etc/*`).

3.12 Edit basic configuration files (e.g., BASH files, `inittab`, `fstab`).

3.13 Load, remove, and edit list modules (e.g., `insmod`, `rmmod`, `Ismod`, `modprobe`).

3.14 Document the installation of the operating system, including configuration.

3.15 Configure access rights (e.g., `rlogin`, NIS, FTP, TFTP, SSH, Telnet).

Domain 4.0 Administration

4.1 Create and delete users.

4.2 Modify existing users (e.g., password, groups, personal information).

4.3 Create, modify, and delete groups.

4.4 Identify and change file permissions, modes, and types by using `chmod`, `chown`, and `chgrp`.

4.5 Manage and navigate the Linux hierarchy (e.g., `/etc`, `/usr`, `/bin`, `/var`).

4.6 Manage and navigate the standard Linux file system (e.g., `mv`, `mkdir`, `ls`, `rm`).

4.7 Perform administrative tasks while logged in as `root`, or by using the `su` command (e.g., understand commands that are dangerous to the system).

4.8 Mount and manage filesystems and devices (e.g., `/mnt`, `/dev`, `du`, `df`, `mount`, `umount`).

4.9 Describe and use the features of the multi-user environment (e.g., virtual terminals, multiple logins).

4.10 Use common shell commands and expressions.

4.11 Use network commands to connect to and manage remote systems (e.g., `telnet`, `ftp`, `ssh`, `netstat`, transfer files, redirect Xwindow).

4.12 Create, extract, and edit file and tape archives using `tar`.

4.13 Manage runlevels using `init` and `shutdown`.

4.14 Stop, start, and restart services (daemons) as needed (e.g., init files).

4.15 Manage print spools and queues.

4.16 Create, edit, and save files using `vi`.

4.17 Manage and navigate the Graphical User Interface (e.g., menus, xterm).

4.18 Program basic shell scripts using common shell commands (e.g., `grep`, `find`, `cut`, `if`).

Domain 5.0 System Maintenance

5.1 Create and manage local storage devices and file systems (e.g., `fsck`, `fdisk`, `mkfs`).

5.2 Verify user and root cron jobs and understand the function of cron.

5.3 Identify core dumps and remove or forward as appropriate.

5.4 Run and interpret `ifconfig`.

5.5 Download and install patches and updates (e.g., packages, `tgz`).

5.6 Differentiate core services from non-critical services (e.g., ps, PID, PPID, `init`, timer).

5.7 Identify, execute, and kill processes (`ps`, `kill`, `killall`).

5.8 Monitor system log files regularly for errors, logins, and unusual activity.

5.9 Document work performed on a system.

5.10 Perform and verify backups and restores.

5.11 Perform and verify security best practices (e.g., passwords, physical environments).

5.12 Assess security risks (e.g., location, sensitive data, file system permissions, remove/disable unused accounts, audit system services/programs).

5.13 Set daemon and process permissions (e.g., SUID – SGID – Owner/groups).

Domain 6.0 Troubleshooting

6.1 Identify and locate the problem by determining whether the problem is hardware, operating system, application software, configuration, or the user.

6.2 Describe troubleshooting best practices (i.e., methodology).

6.3 Examine and edit configuration files based on symptoms of a problem using system utilities.

6.4 Examine, start, and stop processes based on the signs and symptoms of a problem.

6.5 Use system status tools to examine system resources and statuses (e.g., `fsck`, `setserial`).

6.6 Use systems boot disk(s) and root disk on workstation and server to diagnose and rescue file system.

6.7 Inspect and determine cause of errors from system log files.

6.8 Use disk utilities to solve file system problems (e.g., `mount`, `umount`).

6.9 Resolve problems based on user feedback (e.g., rights, unable to login to the system, unable to print, unable to receive or transmit mail).

6.10 Recognize common errors (e.g., package dependencies, library errors, version conflicts).

6.11 Take appropriate action on boot errors (e.g., LILO, bootstrap).

6.12 Identify backup and restore errors.

6.13 Identify application failure on server (e.g., Web page, `telnet`, `ftp`, `pop3`, `snmp`).

6.14 Identify and use troubleshooting commands (e.g., `locate`, `find`, `grep`, `|`, <, >, >>, `cat`, `tail`).

6.15 Locate troubleshooting resources and update as allowable (e.g., Web, man pages, howtos, infopages, LUGs).

6.16 Use network utilities to identify network connectivity problems (e.g., `ping`, `route`, `traceroute`, `netstat`, `lsof`).

Domain 7.0 Identify, Install, and Maintain System Hardware

7.1 Identify basic terms, concepts, and functions of system components, including how each component should work during normal operation and during the boot process.

7.2 Assure that system hardware is configured correctly prior to installation (e.g., IRQs, BIOS, DMA, SCSI settings, cabling) by identifying proper procedures for installing and configuring ATA devices.

7.3 Assure that system hardware is configured correctly prior to installation (e.g., IRQs, BIOS, DMA, SCSI settings, cabling) by identifying proper procedures for installing and configuring SCSI and IEEE 1394 devices.

7.4 Assure that system hardware is configured correctly prior to installation (e.g., IRQs, BIOS, DMA, SCSI settings, cabling) by identifying proper procedures for installing and configuring peripheral devices.

7.5 Assure that system hardware is configured correctly prior to installation (e.g., IRQs, BIOS, DMA, SCSI settings, cabling) by identifying available IRQs, DMAs, and I/O addresses and procedures for device installation and configuration.

7.6 Remove and replace hardware and accessories (e.g., cables and components) based on symptoms of a problem by identifying basic procedures for adding and removing field replaceable components.

7.7 Remove and replace hardware and accessories (e.g., cables and components) based on symptoms of a problem by identifying common symptoms and problems associated with each component and how to troubleshoot and isolate problems.

7.8 Identify basic networking concepts, including how a network works.

7.9 Identify proper procedures for diagnosing and troubleshooting ATA devices.

7.10 Identify proper procedures for diagnosing and troubleshooting SCSI devices.

7.11 Identify proper procedures for diagnosing and troubleshooting peripheral devices.

7.12 Identify proper procedures for diagnosing and troubleshooting core system hardware.

7.13 Identify and maintain mobile system hardware (e.g., PCMCIA, APM).

Assessment Test

1. Which of the following tools is it *most* important to have available on an emergency recovery disk?

   A. `fdformat`

   B. WordPerfect

   C. `mkfs`

   D. `traceroute`

2. The output of `free` shows that a system with 256MB of RAM is using 191MB of RAM and 12MB of an available 350MB of swap space. These values don't fluctuate much over time. Which of the following is true?

   A. The computer would experience substantial speedup by doubling its RAM.

   B. The swap space may be safely eliminated.

   C. The administrator should use the `swapon` command to activate more use of the existing swap space.

   D. Available swap space and RAM are adequate for the system's current uses.

3. Which of the following tasks can `/etc/modules.conf` entries perform? (Choose all that apply.)

   A. They can specify hardware parameters, such as IRQs, to be used by a kernel module.

   B. They can indicate a command to be performed whenever the kernel loads a module.

   C. They can indicate the conditions under which the kernel should recompile a module.

   D. They can specify the module to be loaded for a particular type of hardware.

4. Which of the following are power management protocols? (Choose all that apply.)
   - **A.** ACPI
   - **B.** PPP
   - **C.** SMTP
   - **D.** APM

5. What does the `-t` parameter to `telinit` control?
   - **A.** The time between a polite shutdown of unneeded servers (via `SIGTERM`) and a forceful shutdown (via `SIGKILL`)
   - **B.** The time between issuing the `telinit` command and the time the runlevel change takes place
   - **C.** The runlevel that's to be entered upon completion of the command
   - **D.** The message sent to users before the runlevel change is enacted

6. Which of the following programs might you want to remove on a system that's to function solely as a firewall? (Choose all that apply.)
   - **A.** `init`
   - **B.** The Telnet client
   - **C.** The Linux kernel
   - **D.** The Apache server

7. Which of the following is it wise to do when deleting an account with `userdel`?
   - **A.** Ensure that the user's password isn't duplicated in `/etc/passwd` or `/etc/shadow`.
   - **B.** Search the computer for stray files owned by the former user.
   - **C.** Change permissions on system files to prevent the user from accessing them remotely.
   - **D.** Delete the user's files with a utility that overwrites former file contents with random data.

8. Which of the following is true of Debian-based distributions?

   A. They all use kernels optimized for Intel Pentium CPUs.

   B. They are all derived from Debian GNU/Linux but diverge in various ways from the original.

   C. They cannot use software shipped in RPM format.

   D. They are extremely rare because of the popularity of RPM- and tarball-based distributions.

9. An `ls -l` command reveals that the `loud` file has a permission string of `crw-rw----` and ownership by the user `root` and group `audio`. Which of the following is a true statement about this file?

   A. Only `root` and the account that created it may read or write the file.

   B. The file is a directory, as indicated by the leading `c`.

   C. Anybody in the `audio` group may read from and write to the file.

   D. The command **`chmod 660 loud`** will make it accessible to more users.

10. Which of the following is commonly found in `/etc/inetd.conf` entries for servers but not in the equivalent entries in `/etc/xinetd.conf` or a file in `/etc/xinetd.d`?

    A. A call to `tcpd`

    B. A specification of the protocol, such as `tcp`

    C. A specification of the user, such as `nobody`

    D. Arguments to be passed to the target server

11. Why might a script include a variable assignment like `CC="/usr/bin/gcc"`?

A. To ensure that the script uses `gcc` rather than some other C compiler.

B. Because some programs can't be called from scripts except when referred to by variables.

C. The variable assignment allows the script to run the program even if it lacks execute permission.

D. The variable can be easily changed or assigned different values, increasing the utility of the script.

12. Which of the following symptoms is more common in kernel bugs than in application problems?

A. Programs consume an inordinate amount of CPU time.

B. An error message containing the word `oops` appears in your log files.

C. A program refuses to start and complains of a missing library file.

D. The problem occurs for some users but not for others.

13. Which of the following are potential problems when using a partition resizing utility like `resize2fs` or PartitionMagic? (Choose all that apply.)

A. A power failure or crash during the resize operation could result in substantial data loss.

B. Linux may not recognize a resized partition because resizers often change the partition ID code.

C. No resizing programs exist for the most common Linux filesystem, ext2fs.

D. If the resizer moves the Linux kernel, you'll need to reinstall LILO.

14. In which of the following circumstances is it *most* appropriate to run XFree86 3.3.6 over a 4.0.*x* version of the server?

A. Never; XFree86 4.0.*x* does everything 3.3.6 does, and better.

B. When you need support for multiple simultaneous monitors to display an oversized desktop.

C. When 3.3.6 includes a separate accelerated server for your card.

D. When 4.0.*x* provides unaccelerated support for your chipset but 3.3.6 provides acceleration.

15. You want to set up a firewall on a Linux computer. Which of the following tools might you use to accomplish this task?

A. Apache

B. `iptables`

C. `wall`

D. TCP Wrappers

16. What is the purpose of the `setserial` command?

A. It configures a series of actions to be performed automatically by typing one command.

B. It configures Universal Serial Bus (USB) port parameters.

C. It disables multitasking, forcing Linux to perform only one command at a time.

D. It queries or configures the status of an RS-232 serial port.

17. Which of the following is the purpose of the `rc.local` or `boot.local` startup script?

A. It sets the system's time zone and language defaults.

B. It holds startup commands created for its specific computer.

C. It displays startup messages to aid in debugging.

D. It verifies that all other startup scripts are operating correctly.

18. Which of the following is a protocol that can help automate configuration of SCSI devices?

A. SCAM

B. SMB

C. ASPI

D. ATAPI

19. Which of the following is true of emergency restore procedures?

A. You should test your emergency recovery tools, no matter what they are, to be sure they work and you know how to use them, before an emergency arises.

B. Emergency disks provided with distributions are guaranteed to be able to restore a system, provided they can boot a system initially.

C. The only way to recover a Linux system to a fresh hard disk is to do a partial installation and then recover the backup system using the partial system's tools.

D. You can't completely restore a system from a CD-R backup; you must have a tape backup to create a bootable Linux system.

20. Which of the following is *not* one of the responsibilities of `lpd`?

A. Maintaining the printer queues

B. Accepting print jobs from remote systems

C. Informing applications of a printer's capabilities

D. Sending data to printers

21. Which of the following commands displays the contents of a tarball, including file sizes and time stamps?

A. `tar xzf theprogram-1.2.3.tgz`

B. `tar tzf theprogram-1.2.3.tgz`

C. `tar tvzf theprogram-1.2.3.tgz`

D. `tar x theprogram-1.2.3.tgz`

22. Which of the following does a switch allow that a hub does not permit?

A. 100Mbps operation

B. Linking more than five computers

C. Full-duplex operation

D. Use with 10-Base5 cabling

23. A Linux system administrator is using Nedit (process name `nedit`) to edit a configuration file on a system that hosts several users, but the editor has hung. Because of this, the administrator types **`killall nedit`**. Why might this action be a mistake?

A. To work properly, you must specify the signal type with the *`-SIGNAL`* parameter.

B. It's necessary to locate the PID with `ps` and pass that to `killall`.

C. The command will kill all Nedit processes, even those owned by other users.

D. Without the `-n` parameter, `killall` interprets its first parameter as a username, not a process name.

24. How can you specify the medium used for installation?

A. You can type the codes **`c`** for CD-ROM or **`n`** for network at the `lilo:` prompt when you first boot the installer.

B. You can use individualized boot floppies for each medium, or you can choose the medium during the installation process, depending upon the distribution.

C. Each distribution supports just one installation medium, so the choice is implicit in your choice of distribution.

D. The installer auto-detects the installation medium, so there's no need to explicitly provide this information to the installer.

25. What types of devices may be attached via the USB port? (Choose all that apply.)

A. Keyboards

B. Modems

C. RAM

D. Printers

26. A user whose desktop environment is KDE reports an inability to log in to the computer in graphics mode. Other users (even those who also use KDE) have no such problem. Which of the following is most likely to help resolve this situation?

A. Reinstalling KDE

B. Reconfiguring the `XF86Config` file's `Modeline` statements

C. Deleting the user's `.icewm` directory, which controls KDE's window manager

D. Deleting the user's `.kde` directory, in which KDE's preferences are stored

27. How would you direct the output of the `uptime` command to a file called `uptime-stats.txt`?

A. **`echo uptime uptime-stats.txt`**

B. **`uptime > uptime-stats.txt`**

C. **`uptime | uptime-stats.txt`**

D. **`uptime < uptime-stats.txt`**

28. How do you create a system cron job?

A. You copy a script into a directory specified in `/etc/crontab`, such as `/etc/cron.d/Hourly` or `/etc/cron.daily`.

B. You type **`crontab -u system -e`** to edit the crontab for the `system` user.

C. You type **`crontab -u system`** ***`cron-file`*** to turn *`cron-file`* into the system cron file.

D. You can't; system cron jobs are fixed by the distribution and cannot be altered.

29. Which of the following best describes the relative advantages of Linux and Windows NT/2000?

A. Linux is best used on networks; Windows NT/2000 is best used in stand-alone installations.

B. Linux better supports Unix applications and servers; Windows NT/2000 better supports legacy DOS applications.

C. Linux is best configured through its GUI tools; Windows NT/2000 is easily configured through text-based tools.

D. Linux supports the most popular office productivity applications; Windows NT/2000 supports the most popular Internet servers.

30. Why might you want to use both a firewall and server options to restrict access based on the IP address of a client computer?

A. Without both types of options, access will *not* be restricted.

B. The redundancy provides protection in case one access control mechanism is buggy or misconfigured.

C. Server-based controls are good for protections based on Internet IP addresses, while firewalls are better for protections based on LAN IP addresses.

D. Server-based controls are ineffective and should never be used.

Answers to Assessment Test

1. C. `mkfs` is a tool for creating a new filesystem, which is something you're likely to need to do in an emergency recovery situation. `fdformat` does a low-level format on a floppy disk, WordPerfect is a word processor, and `traceroute` helps diagnose network connectivity problems. You're unlikely to need to use any of these tools from an emergency disk. See Chapter 9 for more information.

2. D. Swap space is being used lightly, and so it isn't degrading system performance. The available swap space is large enough that an unexpected spike in memory usage probably won't overwhelm it. The swap space should *not* be eliminated in case such a spike arrives, though. Adding RAM might improve performance somewhat, but most likely, such an action won't improve it substantially. The `swapon` command won't improve performance, but it could be used to add more swap space if memory demands increased. See Chapter 8 for more information.

3. A, B, D. `/etc/modules.conf` includes parameters to specify all of the indicated information, but the Linux kernel never automatically recompiles a kernel module; that's a task for you as an administrator. See Chapter 6 for more information.

4. A, D. The Advanced Configuration Power Interface (ACPI) and Advanced Power Management (APM) are power management protocols. The Point-to-Point Protocol (PPP) forms TCP/IP network links over serial or telephone lines, and the Simple Mail Transfer Protocol (SMTP) handles e-mail exchanges. See Chapter 8 for more information.

5. A. When shutting down certain servers, `telinit` first tries asking them to shut themselves down by sending a `SIGTERM` signal. The server can then close open files and perform other necessary shutdown housekeeping. If the servers don't respond to this signal, `telinit` becomes more forceful, and passes a `SIGKILL` signal, which is more likely to work but doesn't give the server a chance to shut itself down in an orderly fashion. See Chapter 6 for more information.

6. B, D. You're unlikely to need to use a Telnet client on a firewall, but an intruder who breaks into the firewall could use it to access your internal systems. A firewall shouldn't run any servers that aren't absolutely required, and an Apache server is almost certainly not required. `init` is the master process on a Linux system, and cannot be removed without damaging the system. Likewise, the Linux kernel controls everything else; without it, the computer isn't a Linux computer at all. See Chapter 4 for more information.

7. B. Tracking down and removing or changing the permissions of a former user's files can prevent confusion or possibly even spurious accusations of wrongdoing in the future. Unless the user was involved in system cracking, there's no reason to think that the user's password will be duplicated in the password database. No system file's ownership or permissions should need changing when deleting a user. Although overwriting deleted files with random data may be useful in some high-security environments or with unusually sensitive data, it's not a necessary practice on most systems. See Chapter 4 for more information.

8. B. In 2001, all major Debian-based distributions use not just the Debian package system but many Debian component packages as a starting base. Debian is available for non-*x*86 CPUs, so Pentium optimization is not universal (although Corel Linux does use a Pentium-optimized kernel). RPM is available on Debian and its derivatives, and it can be used, although it's generally not recommended. The Debian package format is the second most popular in the Linux world, behind RPM but ahead of tarballs—at least as a basis for distributions. See Chapter 3 for more information.

9. C. The second set of permission bits (`rw-`) indicates that the file's group (`audio`) may read from and write to the file. This permission string ensures that, if `sound` has more than one member, multiple users may access the file. The leading `c` indicates that the file is a character device file, not a directory. **`chmod 660 loud`** will not change the file's permissions; `660` is equivalent to `rw-rw----`. See Chapter 4 for more information.

10. A. `tcpd` is the TCP Wrappers program. This program provides some security features that are largely provided directly by `xinetd`, so most systems that use `xinetd` don't call `tcpd` from `xinetd`. The other options appear in both types of files, although arguments for the server aren't required for either super server. See Chapter 6 for more information.

11. D. You can easily edit that line to change the program run by the `$CC` variable, or you can assign different values to the variable within a conditional in support of different system configurations. Specifying the program directly will as easily ensure that it's run. Any program that can be called from a variable can be called directly. Variable assignment doesn't allow the script to call programs for which the user lacks execute permission. See Chapter 6 for more information.

12. B. Kernel bugs often manifest themselves in the form of kernel oopses, in which an error message including the word `oops` appears on the console and in log files. Although a program might conceivably trigger a kernel oops, the bug is fundamentally in the kernel. (Kernel oopses also often indicate hardware problems.) See Chapter 9 for more information.

13. A, D. The biggest problem with resizers is the potential for data loss in the event of a crash or power failure during the resize operation. They also can render a system unbootable because of a moved kernel. This latter problem can be overcome by reinstalling LILO. Linux doesn't use partition ID codes except during installation, and resizing programs don't touch these codes. PartitionMagic and `resize2fs` are two programs commonly used to resize ext2 filesystems. See Chapter 7 for more information.

14. D. XFree86 4.0.*x* includes a new driver architecture, so some of 3.3.6's accelerated drivers haven't been ported to the new system as of version 4.0.3. In such cases, using the old server can provide a snappier display. It's 4.0.*x* that provides support for multiple monitors. The presence of a separate accelerated driver in 3.3.6 does not necessarily mean that the 4.0.*x* support is slower. See Chapter 2 for more information.

15. B. `iptables` is the tool for configuring the 2.4.*x* Linux kernel's firewall features. (`ipfwadm` and `ipchains` perform these tasks for the 2.0.*x* and 2.2.*x* kernels, respectively.) Apache is a Web server and `wall` sends messages to all currently logged-on users. TCP Wrappers controls access to specific servers but it isn't a firewall per se. See Chapter 1 for more information.

16. D. `setserial` returns information on the RS-232 serial port's hardware and current operating status, such as the port speed. See Chapter 9 for more information.

17. B. These scripts hold startup commands individualized for their host ("local") computer, as opposed to provided with the distribution. In principle, these scripts *could* be used for any of the other listed purposes, but this isn't their usual function. See Chapter 6 for more information.

18. A. The SCSI Configured Automatically (SCAM) protocol, if supported by the host adapter and SCSI devices connected to it, autoconfigures those devices. The Server Message Block (SMB) is a protocol used in Windows file sharing and implemented by Samba in Linux. The Advanced SCSI Programming Interface (ASPI) is a method common in DOS and Windows for programs to interface with SCSI devices. The Advanced Technology Attachment Packet Interface (ATAPI) is a protocol used by many EIDE devices. See Chapter 1 for more information.

19. A. Testing your emergency tools can save you time when the pressure is on during a restore. In extreme cases, testing tools and finding problems with them may allow you to correct problems that might cause hours of effort in an emergency. Distributions' emergency disks may or may not contain the exact tools you need to restore a system. They're unlikely to contain commercial backup tools. Recovering by doing a slim Linux installation and then using that to recover a backed-up system is one approach to the problem of doing a complete restore, but it's not the only solution. Assuming you've made a complete backup to CD-R, it's possible to completely restore a system from CD-R. See Chapter 7 for more information.

20. C. `lpd` is a multifunction tool that accepts print jobs from local and remote systems, maintains print queues, and sends data to printers (both local and remote). It does *not*, however, feed back information on a printer to applications. See Chapter 8 for more information.

21. C. Option A extracts files from the archive without displaying their names. Option B lists the files in the archive; but without the `--verbose` (`v`) option, it doesn't list file sizes or time stamps. Option D will cause `tar` to attempt to extract the named file from its standard tape device. See Chapter 3 for more information.

22. C. Switches allow full-duplex operation and reduce the chance of collisions on a network relative to hubs. Both devices come in 100Mbps models and models supporting both fewer than and greater than 5 devices. Neither type of device normally supports 10-Base5 cabling; they're both intended for use with twisted-pair network cables. See Chapter 5 for more information.

23. C. In Linux, `killall` kills all processes of the specified name. When the superuser issues the command, it will kill processes owned by normal users, as well as `root`'s processes of the specified name. It can take a signal name or number as a parameter, but if that's omitted, it defaults to a `TERM` signal. The advantage of `killall` over `kill` is that you don't need to look up a PID to use `killall`. There is no `-n` parameter for `killall`. See Chapter 7 for more information.

24. B. Some distributions use particular boot floppies (or other boot media) for specific installation media. Others allow you to select the installation medium from a list early in the installation process. None require you to enter this information at the `lilo:` prompt. Most distributions support multiple installation media. The installer cannot auto-detect your installation medium, except insofar as an installer can be written to support just one, with different boot floppies for different media. See Chapter 2 for more information.

25. A, B, D. The USB port has been used for connecting just about every type of external peripheral, including keyboards, mice, modems, printers, scanners, cameras, and removable-media drives. USB is inadequate for driving a monitor, but some monitors include USB hubs or USB speakers. RAM is always installed internally, never via an external port like USB. See Chapter 1 for more information.

26. D. A login problem isolated to one user is almost certainly related to something in the user's configuration files. One possible source of the problem is the file or directory controlling the window manager or desktop environment. IceWM isn't the default window manager for KDE, so `.icewm` isn't the appropriate directory to delete; `.kde` is. See Chapter 2 for more information.

27. B. The output redirection operator is `>`, so option B sends the output of `uptime` to `uptime-stats.txt`. The `echo` command displays information on the screen, so option A simply causes `uptime uptime-stats.txt` to appear. Option C uses a pipe. If `uptime-stats.txt` were a program, it would process the output of `uptime`, but the result of this command will probably be a `file not found` or `permission denied` error. Option D uses an *input* redirection operator, so `uptime` receives the contents of `uptime-stats.txt` as its input. See Chapter 9 for more information.

28. A. System cron jobs are controlled through `/etc/crontab`, which normally specifies several directories whose contents are run at varying intervals, so copying a script to one of these directories turns it into a system cron job. The `crontab` program is used to create user cron jobs. Both options B and C might work, *if* the computer has a user called `system`, which isn't a standard account name. Cron jobs created in this way would work with the `system` user's privileges, but they wouldn't be system cron jobs in the sense discussed in Chapter 7. See Chapter 7 for more information.

29. B. Most Unix applications can be recompiled on Linux to function, and Windows includes support for DOS programs. Linux and Windows NT/2000 are both well suited to use on networks. Linux, not Windows, is the OS that's best suited to configuration via text-based tools. Windows, not Linux, supports the most popular office productivity applications, such as Microsoft Office. See Chapter 1 for more information.

30. B. A firewall is normally a first line of defense, either on the network as a whole or on an individual computer. If the firewall doesn't block access (because of a bug, misconfiguration, or other problem), subsequent controls may do the job. Server options are one such subsequent control. See Chapter 5 for more information.

Planning the Implementation

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.1 Identify purpose of Linux machine based on predetermined customer requirements (e.g., appliance, desktop system, database, mail server).**
- ✓ **1.2 Identify all system hardware required and validate that it is supported by Linux (e.g., CPUs, RAM, graphics cards, storage devices, network interface cards, modem).**
- ✓ **1.3 Determine what software and services should be installed (e.g., client applications for workstation, server services for desired task), check requirements and validate that it is supported by Linux.**
- ✓ **1.4 Determine how storage space will be allocated to file systems (e.g., partition schemes).**
- ✓ **1.5 Compare and contrast how major Linux licensing schemes work (e.g., GNU/GPL, freeware, shareware, open source, closed source, artistic license).**
- ✓ **1.6 Identify the function of different Linux services (e.g., Apache, Squid, SAMBA, Sendmail, `ipchains`, BIND).**
- ✓ **1.8 Describe the functions, features, and benefits of a Linux solution as compared with other operating systems (e.g., Linux players, distributions, available software).**
- ✓ **1.10 Identify where to obtain software and resources.**
- ✓ **1.11 Determine customer resources for a solution (e.g., staffing, budget, training).**

- ✓ **7.1 Identify basic terms, concepts, and functions of system components, including how each component should work during normal operation and during the boot process.**
- ✓ **7.2 Assure that system hardware is configured correctly prior to installation (e.g., IRQs, BIOS, DMA, SCSI settings, cabling) by identifying proper procedures for installing and configuring ATA devices.**
- ✓ **7.3 Assure that system hardware is configured correctly prior to installation (e.g., IRQs, BIOS, DMA, SCSI settings, cabling) by identifying proper procedures for installing and configuring SCSI and IEEE 1394 devices.**
- ✓ **7.4 Assure that system hardware is configured correctly prior to installation (e.g., IRQs, BIOS, DMA, SCSI settings, cabling) by identifying proper procedures for installing and configuring peripheral devices.**
- ✓ **7.5 Assure that system hardware is configured correctly prior to installation (e.g., IRQs, BIOS, DMA, SCSI, cabling) settings by identifying available IRQs, DMAs, and I/O addresses and procedures for device installation and configuration.**

Most computers are not designed or sold with Linux in mind. This means that Linux doesn't always run properly on them, or it may not take full advantage of the computer's hardware. Therefore, if you need to buy or build a new computer, it's important to understand what Linux needs with respect to hardware so that you can buy a computer with appropriate specifications.

Just as you should understand Linux's hardware requirements, you need to know something about the Linux software world. When you are determining what operating system (OS) to install on a computer, one of the most critical questions you should ask yourself is whether the software you need is available on the OS in question. Locating Linux software and understanding its licensing terms are also important aspects of software requirements for Linux.

Understanding these fundamental hardware and software features will help you in every subsequent aspect of Linux configuration and use because they lay the groundwork for additional Linux layers. Many of your installation choices (discussed in Chapter 2, "Installing Linux") depend upon your hardware, for instance, and many details of system configuration and administration (discussed throughout the rest of the book) rely upon your choice of Linux vendor.

Evaluating Computer Requirements

If you're building or buying a new computer, one of the first steps you must take is to lay out the system's general hardware requirements—the

amount of RAM, the approximate *central processing unit (CPU)* speed, the amount of disk space, and so on. These characteristics are determined in large part by the role or roles the computer will play. For instance, a workstation for a graphics designer will require a large monitor and good video card, but an Internet server needs neither. Once you've decided the general outline of the hardware requirements, you can evaluate your resource limitations (such as your budget) and arrive at more specific hardware selections—specific brands and models for the individual components, or for a pre-built computer.

Workstations

A *workstation* is a computer that is used primarily or exclusively from that computer's own *console* (the keyboard and monitor attached directly to the computer). Workstations are sometimes also referred to as *desktop computers*, although some people apply the latter term to somewhat lower-performance computers without network connections, reserving the term "workstation" for systems with network connections.

Because they're used by individuals, workstations typically require fairly good input/output devices—a large display (typically 17-inch or larger), a high-quality keyboard, and a good 3-button mouse. (Linux, unlike Windows, uses all three buttons, so a 2-button mouse is suboptimal.) Workstations also frequently include audio hardware (a sound card, speakers, and sometimes a microphone) and high-capacity removable media drives (Zip or LS-120 drives, perhaps CD-R or CD-RW burners, and often a DVD-ROM drive).

CPU speed, memory, and hard disk requirements vary from one application to another. A low-end workstation that's to be used for simple tasks such as word processing can get by with less of each of these values than is available on new computers today. A high-end workstation that will be used for video rendering, heavy-duty scientific simulations, or the like may need the fastest CPU, the most RAM, and the biggest hard disk available. Likewise, low-end workstations are likely to have less cutting-edge network hardware than are high-end workstations, and the differing hard disk requirements dictate less in the way of backup hardware for the low-end workstation.

Servers

The word *server* can mean one of two things: a program that responds to network requests from other computers, or the computer on which the server program runs. When designing a computer, the latter is the appropriate definition. Servers usually have little or no need for user-oriented features like large monitors or sound cards. Most servers make heavy use of their hard disks, however, so large and high-performance disks are desirable in servers. For the same reason, *Small Computer System Interface (SCSI*) disks are preferred to *Enhanced Integrated Device Electronics (EIDE)* disks—SCSI disks tend to perform better, particularly when multiple disks are present on a single computer. (This issue is discussed more later in the chapter, in the section entitled "Hard Disk Space.")

Small servers, such as those handling a few users in a small office, don't need much in the way of CPU speed or RAM, but larger servers need more of these quantities, especially RAM. Linux automatically buffers disk accesses, meaning that Linux keeps recent disk accesses in memory, and reads more than it requested from disk. These practices mean that when subsequent requests come in, Linux can deliver them from memory, which is faster than going back to the disk to obtain the data. Thus, a server with lots of RAM can often outperform an otherwise similar server with only a modest amount of RAM.

It's important to realize that server needs fall along a continuum; a very low-demand Web site might not require a very powerful computer, but a very popular Web site might need an extraordinarily powerful system. There are also many other types of servers available, including Usenet news servers, database servers, time servers, and more. (News and database servers are particularly likely to require very large hard disks.)

Dedicated Appliances

Some Linux systems function as dedicated appliances—as routers, print servers for just one or two printers, the OS in small robots, and so on. In some cases, as when the computer functions as a small router, Linux can enable recycling of old hardware that's otherwise unusable. Dedicated applications like these often require little in the way of specialized hardware. Other times, the application demands very specialized hardware, such as custom motherboards or touch-panel input devices. Overall, it's difficult to make sweeping generalizations concerning the needs of dedicated appliances.

Special Needs

Sometimes, the intended use of the computer requires specialized hardware of one variety or another. Common examples include the following:

Video input If the computer must digitize video signals, such as those from a television broadcast or a videotape, you will need a video input board. The Linux kernel includes drivers for several such products, and a variety of programs are available to handle such inputs. The Video4Linux project (`http://roadrunner.swansea.linux.org.uk/v4l.shtml`) supports these efforts.

Scientific data acquisition Many scientific experiments require real-time data acquisition. This requires special timing capabilities, drivers for data acquisition hardware, and software. The Linux Lab Project (`http://www.llp.fu-berlin.de`) is a good starting point from which to locate appropriate information for such applications.

USB devices The *Universal Serial Bus (USB)* is a multipurpose external hardware interface. It's seeing increased use as an interface method for keyboards, mice, modems, scanners, digital cameras, printers, removable-media drives, and other devices. Linux added USB support in the 2.2.18 and 2.4.*x* kernels. This support is good for some devices but weak or non-existent for others. Check `http://www.linux-usb.org` to learn about support for specific devices. You'll also have to be sure to use a distribution with USB support, or at least upgrade the kernel to include this support.

Determining Available Resources

Before you decide on specific hardware to be used for a new system, you should consider the resources available to you. These include any existing hardware that you can reuse, or with which a new system must integrate; the budget under which you must work; and the expertise available to you, both as it exists now and as it might exist after it has been obtained by training personnel to use new hardware and software.

Utilizing Existing Hardware

One resource you may have available is that of existing hardware. One of the reasons for Linux's popularity is that it can be stripped of graphical user interface (GUI) configuration tools and other resource hogs and still accomplish a great deal. This makes Linux an excellent OS with which to stretch the life of old hardware. Before you purchase a new system, you should also consider how it will integrate with your current infrastructure, such as your current network, existing printers, and so on.

Reusing Old Hardware

If you have much in the way of old hardware, you may be able to use it with Linux, particularly in specialized ways. For instance, an early Pentium, or even a 486 system, can make a good dedicated firewall for a small- or medium-sized office. Such an application requires little CPU power, almost no disk space (some products for this purpose fit on a single floppy disk), and little RAM. Such a system can also be turned into a print server for two or three printers (this may require adding parallel port or USB cards); or, with the addition of more disk space and possibly RAM, it can be used as a light-duty file server. With a big enough monitor, such a system can function as a terminal (even a graphical X terminal) to other Linux or Unix computers.

Another approach for reusing old hardware is to scavenge parts for inclusion in an otherwise new system. Components like monitors, speakers, mice, keyboards, removable-media drives, hard disks, and many add-in cards can be moved from an old, decommissioned system to an otherwise new one, thus saving the cost of the new components. Of course, this is best done with components that are in good condition. Also, some components improve substantially with time, so old components may not be useful in modern systems. A 1995 hard disk isn't likely to be large enough to be worth salvaging, for instance. Some technologies, such as those for RAM and CPUs, change so much that old components can't be used in new hardware after more than a year or two.

Integrating With Existing Infrastructure

When planning a new system, you must be aware of the environment in which it will be placed. This environment includes many components. Sometimes there's little you can do to make Linux compatible with this infrastructure;

at other times, you can buy appropriate hardware or install software to make the system compatible. Here are a few examples:

Network connections In many offices, networking is critically important, so a Linux system must be able to work on the local network. If your office uses a Token Ring network, for instance, you'll need to track down and install Token Ring cards for which Linux drivers exist rather than using the more common Ethernet cards.

Printers If the computer must link directly to a printer, the two devices need compatible interfaces. Most *x*86 PCs sold today include one parallel, one or two serial, and two USB ports. You may need to add extra ports or buy a USB hub if you need to connect more printers than this. (Printer driver configuration is another important issue, which is discussed in Chapter 8, "Hardware Issues.")

Modems Linux works with most external RS-232 serial modems. External USB modems work if they follow the Communication Device Class Abstract Control Model (CDC-ACM) protocol. Internal models may or may not work, depending upon the model. Broadband (cable and DSL) modems work if they use Ethernet interfaces, but internal and USB devices both require explicit driver support, so check on that detail.

Scanners Check on `http://panda.mostang.com/sane` for information on Linux's support for scanners. As a general rule, SCSI-interfaced models work well, while USB and parallel-port scanners may or may not work, depending upon the model in question.

Removable-media devices Linux works well with most removable-media devices. Whether your office uses Zip, LS-120, magneto-optical, Jaz, CD-RW, or other media, Linux can use them. The main caveat relates to the interface. SCSI and EIDE devices work well, as do many parallel-port and USB devices—but a few of the latter two types will cause problems. Also, some Windows CD-RW software creates discs that Linux can't read.

As a general rule, if you expect Linux to interface directly with a device, you should check on compatibility. You can usually find some way to get Linux working with your infrastructure, but you may need to buy extra hardware or use unusual drivers.

Balancing Budgetary Limitations

Whether you're working in an organization or buying a computer for yourself, it's a good idea to set a budget for a new computer before you buy. If you fail to do this, it's easy to fall into a pattern of feature inflation, in which you decide to spend "just" $20 more on one upgrade, and "just" $30 more on another. Before long, you've added $500, $1000, or more to the cost of the computer. When setting your budget, remember that you may need to make some unexpected purchases after the fact. For instance, you may find that you need an adapter to connect an external SCSI device to your computer's external SCSI port.

There's a good chance you'll find that your ideal computer costs more than you've budgeted for it. If this happens, you have two choices: revise your budget or settle for a lesser computer. (You can, of course, do a little of both.) If you choose to trim, here are some suggestions:

- Today's hard disks are measured in the tens of gigabytes in capacity, which is more than enough for most workstations and even many servers. You may be able to make do with a smaller hard disk than you'd planned. If necessary, you can add another hard disk in the future. Also, look for the hard disk capacity sweet spot—the point that carries the lowest price per gigabyte of storage. If you'd planned to buy at just above the sweet spot, reducing disk size by a little can reduce the cost by a lot.
- It's easy to list floppy, DVD-ROM, CD-RW, Zip, and Jaz drives as being required, but are they? You might be able to omit the DVD-ROM drive and use a CD-RW to both read and write CDs. An LS-120 drive can read ordinary floppy disks.
- Large monitors are very nice, but they may be overkill, particularly for low-end workstations and servers. I do *not*, however, recommend that you cut costs by purchasing an unknown low-cost monitor, particularly for a workstation. A monitor with a blurry screen, or one that flickers a lot, can cause eyestrain and headaches. Similarly, cheap keyboards can be a false economy on heavily used workstations if they contribute to carpal-tunnel syndrome.

- CPU speed increases rapidly in the computer industry, and in 2001, CPUs are faster than required for many applications. If you truly need a fast CPU, by all means get one, but for many workstations, a mid-range CPU will do well. You may also want to look for the CPU price sweet spot. This often occurs just below the top of the line, so even if you need a speedy CPU, getting the model that's one or two notches below the top-speed model may make sense.
- Many of today's top video cards are targeted at game players, who want high-speed, 3-dimensional effects. 3D effects are computationally expensive, and they also require a lot of RAM. For traditional office applications, a card with minimal or no 3D effects and 8MB of RAM is perfectly adequate. Some high-end graphics applications may need more, though.

One additional consideration, particularly if you're buying multiple Linux computers, is that you may be able to invest additional resources in a single computer rather than distribute the resources among a group of systems. For instance, a single computer with a large hard disk can function as a file server for dozens of computers with anemic hard disks. You can even give users accounts on a central system and have them run programs on that computer, using their own systems as little more than terminals. In fact, this approach can be a great way to reuse old hardware—even a 486 can function as an adequate terminal (even for X-based GUI programs).

Considering Available Expertise

Ideally, once Linux is installed and running, the details of your hardware selections will be unimportant to the computer's ultimate users. The average user doesn't really care if you use an ATI or Matrox video card, and they won't need to manually reconfigure these components. Of course, the end user may care that you've selected a 17-inch rather than a 19-inch monitor to save money.

Local expertise becomes important in a few hardware devices, however. If you're installing a new network, you may need to arrange for training so that users can use the necessary network applications. At anything but the smallest sites, it's also important to have somebody on hand to troubleshoot network problems as they arise. Removable disks occasionally require explanations, particularly warnings concerning proper care of the media. In Linux, CD-R and CD-RW drives don't operate like other removable-media

devices, so users must be told how to use appropriate Linux software. End users may also require training on tape backup devices.

End-user training and expertise are at least as important for Linux software as for hardware. Users who are used to Microsoft Windows or MacOS can probably adapt to Linux without too much difficulty, but they'll find this task much easier if they're given a complete desktop environment, such as the K Desktop Environment (KDE) or the GNU Network Object Model Environment (GNOME). Both come with most Linux distributions, and are described briefly in Chapter 2. If you're migrating a large number of users from another OS to Linux, you may want to organize some introductory Linux orientation sessions in which you demonstrate Linux and highlight some of the differences between Linux and the old OS.

Deciding What Hardware to Use

Once you've decided on the approximate specifications for a computer and you've set a budget, you can begin deciding on exact specifications. If you possess the necessary knowledge, I recommend indicating manufacturer and model numbers for every component, along with one or two backups for each. (RAM, however, is close to being a commodity; few people shop for RAM by brand, although the *type* of RAM is important.) You can then take this list to a store and compare it to the components included in particular systems, or you can deliver your list to a custom-build shop to obtain a quote. If you don't have enough in-depth knowledge of specific components, you can omit the make and model numbers for some components, such as the hard disk, CD-ROM drive, monitor, and motherboard. You should definitely research Linux compatibility with video cards, network cards, SCSI host adapters (if you decide to use SCSI components), and sound cards (if the computer is to be so equipped). These components can cause problems for Linux, so unless you buy from a shop that's experienced in building Linux systems, a little research now can save you a lot of aggravation later when you try to get a component working in Linux.

A Rundown of PC Hardware

Computers are built from several components that must interact with each other in highly controlled ways. If a single component misbehaves or if the interactions go awry, the computer as a whole will malfunction in subtle or obvious ways. Major components in computers include the following:

Motherboard The *motherboard* (also sometimes called the mainboard) holds the CPU, RAM, and plug-in cards. It contains circuitry that "glues" all these components together. The motherboard determines what type of memory and CPU the computer can hold. It also includes the BIOS, which controls the boot process, and it usually has built-in support for hard disks, floppy disks, serial ports, and other common hardware.

CPU The CPU is the computer's brain—it performs most of the computations that result in a system's ability to crunch numbers in a spreadsheet, lay out text in a word processor, transform PostScript to printer-specific formats for a print queue, and so on. To be sure, some computations are performed by other components, such as some video computations by a video card, but the CPU does the bulk of the computational work.

Memory Computers hold various types of memory; the most common general classes of these are random access memory (RAM) and read-only memory (ROM). There are several varieties of each of these. Memory holds data, which can include Linux software and the data upon which that software operates. Memory varies in access speed and capacity.

Disk storage Disk storage, like memory, is used to retain data. Disk storage is slower than memory, but usually higher in capacity. Typically, Linux itself resides on disk storage, and when the system boots, parts of Linux are loaded into RAM. In addition to the common hard disks, there are lower-capacity removable disks, CD-ROMs, and so on. Disks are controlled through EIDE or SCSI circuitry on the motherboard or separate cards. As a general rule, Linux doesn't need specific drivers for disks, but Linux does need drivers for the controller.

Video hardware Video hardware includes the video card and the monitor. The video card may or may not literally be a separate card; sometimes it's built into the motherboard. Collectively, video hardware provides the primary means for a computer to communicate with its user, but Linux has the ability to do so through other computers' video hardware. Linux's video support is provided in two ways: through standard text-mode features in the kernel that work with just about any video card; and through drivers in XFree86, Linux's GUI package, that work with most cards, but not absolutely all of them.

Input devices The keyboard and mouse allow you to give commands to the computer. These devices are well standardized, although there are a few variants of each type. Linux requires no unusual drivers for most common keyboards and mice (including trackballs and similar mouse alternatives), but if you use USB devices, you may need to use a recent kernel—2.2.18 or 2.4.0 or later.

Network devices In most business settings, network hardware consists of an *Ethernet* card or a card for a similar type of computer network. Such networks link several computers together over a few tens or hundreds of feet, and they can interface to larger networks. Even many homes now use such a network. It's also possible to link computers via *modems*, which use telephone lines to create a low-speed network over potentially thousands of miles. These devices are usually quiescent until late in the boot process, when Linux may launch programs to begin network interactions. There are ways to boot a computer via network connections, though.

Audio hardware Many workstations include audio hardware, which lets the system create sounds and digitize sounds using microphones or other audio input devices. These aren't critical to basic system functioning, though; Linux will boot quite well without a sound card.

To understand how these components interact, consider Figure 1.1, which shows a simplified diagram of the relationship between various system components. Components are tied together with lines that correspond to traces on a circuit board, chips on a circuit board, and physical cables. These are known as *busses*, and they carry data between components. Some busses are contained within the motherboard, but others are not. Components on a single bus can often communicate directly with one another, but components on different busses require some form of mediation, such as from the CPU. (Although not shown in Figure 1.1, there are lines of communication between the memory and PCI busses that don't directly involve the CPU.) A lot of what a computer does is coordinate the transfer of data between components on different busses. For instance, to run a program, data must be transferred from a hard disk to memory, and from there to the CPU. The CPU then operates on data in memory, and may transfer some of it to the video card. Busses may vary in speed (generally measured in megahertz, MHz) and width (generally measured in bits). Faster and wider busses are better than slower and narrower ones.

FIGURE 1.1 A computer is a collection of individual components that connect together in various ways.

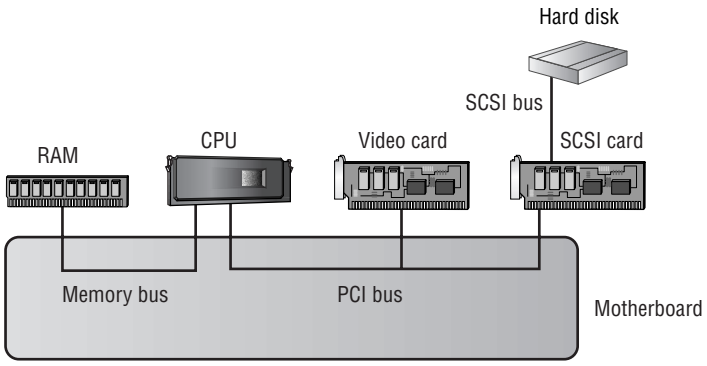


Figure 1.1 is *very* simplified. For instance, the link between the CPU and RAM passes through the motherboard's chipset and various types of cache, as described briefly in the upcoming section, "RAM."

The next few sections examine several critical system components in more detail.

CPU

Linux was originally developed for Intel's popular 80*x*86 (or *x*86 for short) line of CPUs. In particular, a 386 was the original development platform. (Earlier CPUs in the line lack features required by Linux.) Linux also works on subsequent CPUs, including the 486, Pentium, Pentium MMX, Pentium Pro, Pentium II, Pentium III, Pentium 4, and Celeron.

In addition to working on Intel-brand CPUs, *x*86 versions of Linux also work on competitors' *x*86-compatible chips. Today, the most important of these are the AMD K6 series, Athlon, and Duron. VIA also sells a line of CPUs originally developed by Cyrix and IDT, but in 2001, these lag substantially behind the offerings from Intel and AMD in speed. A few other companies have sold *x*86-compatible CPUs in the past, but these companies have failed or been consumed by others. (IBM and some other firms sell Cyrix or AMD designs under their own names, sometimes as part of a package to

upgrade an existing motherboard beyond its originally-designed maximum CPU speed.)

As a general rule, Linux has no problems with CPUs from any of the *x*86 CPU manufacturers. When a new CPU is introduced, Linux distributions occasionally have problems booting and installing on it, but such problems are usually fixed quickly.

In addition to *x*86 CPUs, Linux runs on many other CPUs, including the Apple/IBM/Motorola PowerPC (PPC), Compaq's (formerly DEC's) Alpha, and the SPARC CPU in Sun workstations. Linux is most mature on *x*86 hardware, and that hardware tends to be less expensive than hardware for other architectures, so it's generally best to buy *x*86 hardware for Linux.

The best CPUs of some non-*x*86 lines sometimes perform slightly better than the best *x*86 CPUs, particularly in floating-point math, so you might favor alternative architectures for these reasons. You might also want to dual-boot between Linux and an OS that's available for some other architecture, such as MacOS.

To date, *x*86 systems use 32-bit internal registers, although Pentium systems and above have 64-bit links to memory. Some non-*x*86 systems use 64-bit internal registers, and both Intel and AMD are developing 64-bit variants of the *x*86 architecture. The Intel variant is known as IA-64 and has been implemented in Intel's Itanium CPU. IA-64 works best with code that has been specially designed for the IA-64 architecture. The Linux kernel works on IA-64, and some IA-64 Linux distributions are available. AMD is developing a different 64-bit version of the *x*86 architecture, known as *x*86-64. The code name for AMD's 64-bit CPU is Hammer, and the company hopes to release this CPU by the end of 2001.

When comparing CPU performance, most people look at the chips' speeds in megahertz (MHz) or gigahertz (GHz; 1GHz is 1,000MHz). This measure is useful when comparing CPUs of the same type; for instance, a 750MHz Athlon is slower than a 900MHz Athlon. Comparing across CPU models is trickier, though, because one model may be able to do more in a single CPU cycle than another can. What's worse, this comparison may differ according to the nature of the computation. For instance, in general, *x*86 CPUs have a reputation for poor floating-point math performance, although they've been improving on this measure in recent years. Thus, an Intel CPU might be the equal of an Alpha in most tasks, but the Alpha might have a substantial

advantage in applications that require floating-point math, such as ray tracing and certain scientific applications. When comparing different CPUs (for instance, Pentium 4 to Athlon), you should look at a measure such as MIPS (millions of instructions per second) or a benchmark test that's relevant to your intended application. (The Linux kernel uses a measure called Bogo-MIPS as a calibration loop when it boots, but this is *not* a valid measure of CPU performance; it's used only to calibrate some internal timing loops.) The best measure is how quickly the software *you* use runs on both CPUs.

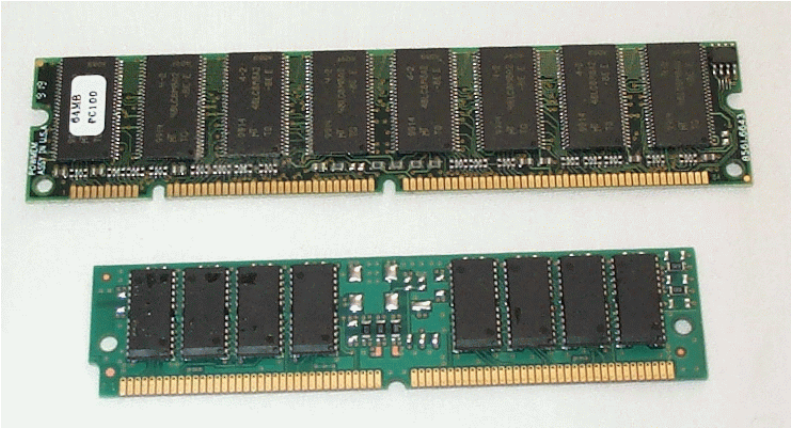
CPUs plug into specific motherboards, which are the main (and sometimes the only) major circuit board in a computer. The motherboard contains a *chipset*, which implements major functions such as an EIDE controller, an interface between the CPU and memory, an interface to the keyboard, and so on. Linux works with most motherboards, although on occasion, Linux doesn't support all of a motherboard's features. For instance, a motherboard may include an integrated video or audio chipset for which Linux drivers are immature or non-existent. The key consideration in choosing a motherboard is that it is compatible with the CPU you buy—both its model and its speed. If you buy a preassembled system, this won't be a concern.

RAM

RAM comes in several forms, the most common of which in 2001 is the *dual inline memory module (DIMM)*. Older motherboards and some other components use the *single inline memory module (SIMM)* format, which comes in both 30-pin and 72-pin varieties. Figure 1.2 shows a DIMM and a 72-pin SIMM. A few motherboards use *RDRAM inline memory modules (RIMMs)*, which physically resemble DIMMs, but use a special type of RAM known as *RAMbus dynamic RAM (RDRAM)*.

Motherboards host sockets for particular types of memory—30-pin SIMM sockets in many 486 and older motherboards, 72-pin SIMM sockets in some 486- and Pentium-class motherboards, DIMM sockets in some Pentium-class and later motherboards, and RIMM sockets in some Pentium II and later motherboards. Depending upon the module and CPU type, you may need to add modules singly, in pairs, or in groups of four. Pentium and later systems take 72-pin SIMMs in pairs and DIMMs or RIMMs singly.

FIGURE 1.2 Currently, DIMMs (top) are used in many computers; SIMMs (bottom) are used in older computers and some peripherals.



In addition to differences in physical interfaces, RAM varies in its electronic characteristics. RAM today is largely derived from *dynamic RAM (DRAM)*, which has spawned many improved variants, such as fast page mode (FPM) DRAM, extended data out (EDO) DRAM, synchronous DRAM (SDRAM), double data rate (DDR) SDRAM, and RDRAM. Most motherboards accept just one or two types of RAM, and with the exception of RDRAM and RIMMs, the physical format of the memory does not clearly indicate the RAM's electronic type. In 2001, most motherboards accept some combination of SDRAM, DDR SDRAM, or RDRAM, and possibly one or two lesser varieties. DDR SDRAM and RDRAM are the speed champions today. Each has its adherents. DDR SDRAM uses fairly conventional improvements to regular DRAM, delivering fast memory access by using a wide (64-bit) and moderately fast (66–133 MHz) bus. RDRAM uses a more unusual design in which the RIMM uses a narrow (16-bit) but unusually fast (800 MHz) bus externally and a separate bus within the RIMM that uses a more conventional configuration.

RAM also varies in how well it copes with errors. Computer memory is composed of individual *bits*, which are *binary* (base 2) numbers—each digit is either 1 or 0. A *byte* is composed of eight bits. If a single bit changes its value, say because of a cosmic ray hitting the memory, the data becomes corrupt. This can cause subtle or extreme errors in computations or it can result in other data being corrupted. Some memory modules incorporate a ninth bit in each byte as an error-detection bit. This bit is encoded to indicate

whether an even or odd number of bits in the other eight bits in the byte are set. If an error occurs, the motherboard's memory controller can detect this fact. Unfortunately, the usual result is a system crash, the idea being that it's better to crash the computer than to propagate bad data.

A more sophisticated approach is error correction. Like the error detection process described above, error correction also requires one extra bit per byte (or, more precisely, eight extra bits for every 64 bits in a Pentium-class or above system). In this case, though, a motherboard that supports error correction can correct 98 percent of the errors that occur, resulting in no disruption to the computer's operation. This is clearly a desirable characteristic, particularly in mission-critical systems such as important servers.

All of these characteristics apply to *main memory*, which, as you might imagine, is the main type of memory in a computer. Motherboards or CPUs also support another type of memory, though—*cache memory*. A computer has much less cache memory than main memory (typically under 1MB), but the cache memory is much faster. The system stores frequently used memory in the cache, which results in a substantial performance increase. Typically, two caches exist. The first, known as the L1 cache, resides in the main part of the CPU and is a few kilobytes in size. On Pentium-class and earlier systems, the second cache, known as L2, is on the motherboard and can sometimes be upgraded. On Pentium Pro, Athlon, and later systems, the L2 cache is on the CPU package, but it's not part of the same chip as the CPU. A few motherboards that take CPUs with an on-board L2 cache also provide a cache on the motherboard. In this configuration, the motherboard's cache is known as the L3 cache.

Linux itself is unconcerned with these details. To Linux, memory is memory, and the OS doesn't particularly care about what physical or electronic form the memory takes or whether it supports any form of error detection or correction. All these details are handled by the motherboard, which is why it's so important that your memory match the motherboard's requirements.

When upgrading a computer's memory, try to buy from a retailer that has a memory cross-reference tool. Such a tool may be a Web-based form or a printed book. You look up or enter your motherboard or computer model and find a specific model of memory that's compatible with your computer. If such a tool is unavailable, check your motherboard's manual for detailed specifications concerning the types of memory it accepts, and use those specifications when shopping.

Hard Disk Space

The great divide in hard disks is between EIDE and SCSI devices. Both of these busses come in a variety of speeds, ranging from less than 10MBps to 160MBps, with higher speeds on the way. In order to achieve a given speed, both the hard disk and its interface must support the same speed. For instance, using an old 10MBps Fast SCSI drive with an 80MBps Ultra2 Wide SCSI host adapter will yield only 10MBps speeds, not 80MBps speeds.

It's important to distinguish between the speed of the *interface* and the speed of the *device*. Manufacturers typically emphasize the speed of the interface, but the mechanical device usually can't support these speeds. A hard disk might have an 80MBps Ultra2 Wide SCSI interface but be capable of only 35MBps sustained transfer rates. Manufacturers express the device's true maximum speed as an *internal transfer rate*, as opposed to the *external transfer rate* (of the interface). To further confuse matters, many manufacturers give the internal transfer rate in mega*bits* per second (Mbps), but the external rate in mega*bytes* per second (MBps). If you fail to do the appropriate conversion (dividing or multiplying by 8), you'll erroneously believe that the interface is the bottleneck in data transfers to and from the device. Disks can transfer data at their external transfer rate only when they've previously stored data from the disk in their internal caches. For this reason, external speeds substantially higher than internal speeds can produce modest speed benefits, and disks with large caches are preferable to those with small caches.

As a general rule, SCSI devices are preferred in computers in which disk performance is important. There are several reasons for this:

- Depending upon the variety of SCSI, each SCSI host adapter can support 7–15 devices on one hardware interrupt. (There are only 15 interrupts available in the *x*86 architecture, and many are reserved for critical hardware like the keyboard.) EIDE, by contrast, supports just two devices per cable (and hence per interrupt), although most motherboards include support for two chains (using two interrupts), for a total of four devices.
- SCSI devices multitask better than do EIDE devices. Given sufficient capacity on the SCSI host adapter, multiple SCSI devices can be engaged in data transfers at full speed. EIDE, by contrast, dedicates its full capacity to one device per chain, even if that device can't use the EIDE controller's full capacity.

- Hard disk manufacturers tend to release their fastest and highest-capacity drives in SCSI format. EIDE drives tend to be slower and smaller.

These advantages are substantial, but for many situations, they're overwhelmed by one advantage of EIDE: It's less expensive. As just mentioned, modern *x*86 motherboards ship with support for two EIDE chains, so there's no need to buy an EIDE controller. EIDE hard disks are also typically less expensive than SCSI devices of the same capacity, although the EIDE drives are often slower.

On the whole, SCSI is worthwhile when disk performance is important or when you need to support a large number of storage devices (including CD-ROM, DVD-ROM, removable disk, and tape drives). For most low-end and even mid-range workstations, though, EIDE's lower cost makes it appealing, and EIDE performance is adequate for many such systems.

Fortunately, Linux's support for both EIDE and SCSI adapters is excellent. Most EIDE controllers can be run in an old-style (and slow) mode using generic drivers, but faster speeds often require explicit driver support. Therefore, you may want to check on Linux's EIDE drivers for your motherboard or EIDE controller. There is no generic SCSI host adapter support, so you must have support for your specific SCSI host adapter.

Once you configure Linux to work with an EIDE controller or a SCSI host adapter, you don't need to worry about support for specific models of disk. (If you recompile your kernel, you need to explicitly include support for hard disks or any other devices attached to your adapter, but this support is present by default in all major Linux distributions.) You can purchase hard disks and other storage devices on the basis of capacity, speed, and the reputation for quality of a manufacturer or model.

Hard disks consist of spinning platters with read/write heads reading or writing data from those platters as the platters move under the heads. You'll usually see ads that list the rotational velocity of platters, such as 7,200 revolutions per minute (rpm); and the latency or seek time, such as 9 milliseconds (ms). A faster rotational velocity translates into data passing under the heads faster. The latency is the time it takes to move a head to a new position from the center of the disk, so a lower latency is better than a higher one. Unfortunately, these two figures aren't the only important ones in determining the speed of a hard disk. The data density—how much data can be packed into a given amount of space—interacts with the rotational velocity to determine disk speed. A 7,200rpm disk might actually be faster than a

10,000rpm disk, if the former has a substantially higher data density. For this reason, you should look for the actual transfer rates, expressed in MBps or Mbps.

The data transfer rate varies between inner and outer tracks—outer tracks contain more data than inner tracks do, so outer tracks produce higher transfer rates.

Network Hardware

Ethernet is the most common type of network in 2001. There are several different varieties of Ethernet, including 10Base-2 and 10Base-5 (which use thin and thick coaxial cabling, respectively); 10Base-T, 100Base-T, and 1000Base-T (which use twisted-pair cabling similar to telephone wires); and 1000Base-SX (which uses fiber-optic cabling). In any of these cases, the first number (10, 100, or 1000) represents the maximum speed of the network, in Mbps. Of these classes of Ethernet, 100Base-T is currently the most popular choice for new installations.

Most 100Base-T network cards also support 10Base-T speeds. This fact can help you migrate a network from 10Base-T to 100Base-T; you can install dual-speed cards in new systems and eventually replace older 10Base-T hardware with dual-speed hardware to upgrade the entire network. Similarly, many 1000Base-T cards also support 100Base-T and even 10Base-T speeds.

Linux's support for Ethernet cards is, on the whole, excellent. Linux drivers are written for particular chipsets, rather than specific models of network card. Therefore, the driver names often bear no resemblance to the name of the card you've bought, and you may use the same driver for boards purchased from different manufacturers. Fortunately, most distributions do a good job of auto-detecting the appropriate chipset during installation, so you probably won't have to deal with this issue if the card is installed when you install Linux. Chapter 8 covers adding new hardware, should you need to add a network card after the fact.

If you're faced with the choice, purchase a *Peripheral Component Interconnect (PCI)* card rather than an *Industry Standard Architecture (ISA)* card. PCI cards tend to be easier to configure, and they support higher transfer rates. The ISA bus tops out at a theoretical maximum speed of 64Mbps—less than that of 100Base-T Ethernet. A 32-bit PCI card has a theoretical maximum speed of 1056Mbps, which is barely enough for gigabit Ethernet.

In practice, PCI can't handle this full speed because the needs of other devices and deviations from theoretical maximum performance will degrade performance. (A rare 64-bit PCI variant is better able to sustain full gigabit Ethernet speeds.)

Linux supports networking standards other than Ethernet, but these devices are less well supported overall. Linux includes support for some Token Ring, Fiber Distributed Data Interface (FDDI), LocalTalk, Fibre Channel, and wireless products, among others. If your existing network uses one of these technologies, you should carefully research Linux's support for specific network cards before buying one.

Networking hardware outside of the computer doesn't require Linux-specific drivers. Network cables, hubs, switches, routers, and so on are all OS-independent. They also generally work well with each other no matter what their brands, although brand-to-brand incompatibilities occasionally crop up.

One partial exception to the rule of needing no specific Linux support is in the case of network-capable printers. If you buy a printer with a network interface, you must still have appropriate Linux printer drivers to use the printer, as described in Chapter 8. Fortunately, network-capable printers usually understand PostScript, which is ideal from a Linux point of view.

Video Hardware

Linux works in text mode with just about any video card available for *x*86 systems. This means you can log in, type commands, use text-based utilities, and so on. Such operation is probably adequate for a system intended to function as a server, so selection of a video card for a server need not occupy too much of your time. Workstations, though, usually operate in GUI mode, which means they run XFree86 or a commercial X Window System (X for short) server.

Unlike most other drivers, the drivers necessary to operate a video card in the bitmapped graphics modes used by X do not reside in the kernel; they're part of the X server. Therefore, you should research the compatibility of a video card with XFree86 (`http://www.xfree86.org`) or the commercial X servers, Accelerated-X (`http://www.xig.com`) and Metro-X (`http://www.metrolink.com`). Because XFree86 ships with all major Linux distributions, it's best to use a board it supports. As a general rule of thumb, it's

best to avoid the most recent video cards because drivers for XFree86 tend to lag a few months behind the release of the hardware. A few manufacturers do provide XFree86 servers for their products, though, and the commercial X servers sometimes introduce drivers more rapidly than does the XFree86 team.

The Linux kernel is beginning to acquire a number of video drivers, known as frame buffer drivers. XFree86 includes a driver to interface to these kernel-level drivers. This approach is particularly common outside of the *x*86 world.

One important question when deciding on a video card is how much memory it should contain. The video card uses on-board memory to store a copy of the image displayed on the screen. Because of this, the video card must have enough memory to hold this image. The formula for determining this value is

$$R = x \times y \times b \div 8{,}388{,}608$$

In this equation, R is the RAM in megabytes, x and y are the width and height of the screen, respectively, and b is the color depth in bits (typically 8, 16, 24, or 32). For instance, to support a 1024×768 display at 16-bit color depth requires 1.5MB of RAM.

Most video cards available in 2001 have at least 8MB of RAM, which is more than enough to handle a 1600×1200 display with a 32-bit color depth—a very high resolution and color depth. Cards with more memory than this typically use it in conjunction with 3D effects processors, which are useful in games and certain types of 3D rendering packages. 3D acceleration is still rare in Linux, and few Linux programs take advantage of these effects. If you need them, you should research 3D support carefully before settling on a product to buy.

Miscellaneous Hardware

Some hardware is so well standardized that there's no reason to give it much thought for Linux compatibility. The following are included in this category:

Cases Computer cases are hunks of plastic and metal shaped to hold other components. Usually, they also include power supplies, fans, and a few wires. Cases do vary in quality—check for rough edges, a good fit, and easy access. There's nothing OS-specific about them, though.

Floppy drives Standard floppy drives are very standardized. There are a few variant technologies, though, like LS-120 drives, which typically interface via the EIDE port. These may need to be treated like hard disks in the `/etc/fstab` configuration file (described in Chapter 6, "Managing Files and Services").

CD-ROM drives Today, most CD-ROM drives use either the EIDE (aka AT Attachment Packet Interface, or ATAPI) or the SCSI interface, and the devices are very well standardized. The main exceptions are USB-interfaced drives. Even DVD-ROM drives are well standardized. Recordable and rewriteable CDs (CD-R and CD-RW drives) are also becoming well standardized.

Tape drives Most tape drives use a standard EIDE/ATAPI or SCSI interface. These drives almost always respond to a standardized set of commands, and so they don't require a special configuration in Linux. There are a few older floppy-interfaced drives that work with the Linux ftape drivers, which are part of the kernel. Some old parallel-interfaced drives can cause problems, and newer USB-interfaced drives are as yet rare and not well tested.

Keyboards Standard PC keyboards are well supported by Linux and require no special configuration. Some keyboards include special keys that may not be recognized by Linux, though, like volume-control keys or keys to launch specific applications. There are also USB keyboards available. These are supported in 2.4.*x* kernels, but they aren't as well tested.

Mice Most mice today use USB or PS/2 interfaces, but some older mice used RS-232 serial or various exotic interfaces. All are well supported, although USB support prior to the 2.4.*x* kernels was poor. Note that the tracking technology (conventional wheeled mouse, optical mouse, trackball, touchpad, and so on) is unimportant; it's only the interface protocols and the type of hardware interface that are important. Mice using USB or PS/2 hardware use the PS/2 protocol or a variant of it that supports wheels.

Serial and parallel ports If you need to add extra serial or parallel ports, you can do so with plug-in cards. These cards are fairly well standardized, so they'll seldom pose serious problems with Linux itself, although they can sometimes conflict with other hardware.

Monitors Monitors don't require drivers, although you may need to know certain features of a monitor to configure it in XFree86. Specifically, you may need to know the monitor's maximum horizontal and vertical refresh rates (expressed in kHz and Hz, respectively). With XFree86 4.0 and later, the X server can sometimes obtain this information from the monitor. (Chapter 2 covers X configuration in detail.)

Some other types of hardware require special consideration. These devices may require unusual drivers or configuration in Linux. Examples include the following:

USB devices Linux needs drivers for each USB device; a single USB driver isn't sufficient to handle all USB devices. The 2.2.18 and 2.4.*x* kernels add support for many—but by no means all—USB devices. Check `http://www.linux-usb.org` for information on what's currently supported.

Internal modems In years gone by, internal modems seldom caused problems in Linux, because they were essentially composed of ordinary modem hardware linked to an ordinary serial port, all on one card. Today, though, internal modems are more likely to be *software modems*—devices that rely upon the CPU to do some of the modem's traditional chores. Such devices require special drivers, which sometimes don't exist for Linux. Check `http://www.linmodems.org` for information on what's supported and what's not.

Sound cards Linux supports most sound cards. The standard kernel includes drivers for many cards. Commercial variants of these drivers (often called the Open Sound System, or OSS) are available from `http://www.4front-tech.com`. An entirely separate project, the Advanced Linux Sound Architecture (ALSA; `http://www.alsa-project.org`) aims to replace the standard kernel drivers, and supports a different (but overlapping) set of cards. You can also check to see if the sound card vendor provides drivers, which may be unique or work along with the kernel or ALSA core.

Video acquisition boards Video acquisition hardware includes cameras (which typically interface via the parallel, USB, or RS-232 serial ports) and internal cards that accept television input signals. The Video4Linux project (`http://www.exploits.org/v4l`) is devoted to developing tools for such devices, and the standard kernel includes many of the requisite drivers—but be sure to check for supported hardware if this is important.

Aside from trivial components like cables, you should be cautious about adding hardware to a Linux computer without checking its compatibility with Linux. It's easy to forget that computer hardware often requires drivers, and if nobody has written appropriate drivers for Linux, that hardware simply will not work. These drivers can also vary in quality, which is part of why one device may work well while another works poorly.

Unreliable drivers can be a major cause of system instability. Most drivers have privileged access to the computer's hardware as well as to kernel data structures. As a result, a bug in a driver is unusually likely to crash the system or cause other major problems.

Checking Hardware Configuration before Installation

One of the reasons to buy a preassembled computer is so that you won't have to worry about all the pesky little details of hardware configuration. Unfortunately, life doesn't always work out that way. Pre-built computers often come with one or more components configured suboptimally, so even if you buy such a system, it's wise to review the hardware configuration before you install Linux. Some settings, if incorrect, can cause problems during Linux installation, or soon thereafter.

Because most hardware is inside the computer's case, you must open that case to check the hardware's status. This poses two dangers. First, you might suffer an electrical shock if the computer is plugged into a wall outlet. Some power supplies have power switches independent of the computer's main switch; turning these off can reduce this risk. Second, static charges built up in your own body (say, from shuffling across a carpet in dry weather) can damage computer components. You can reduce this risk by grounding yourself frequently—for instance, by wearing a wrist strap designed for that purpose or by frequently touching a water faucet, radiator, or the computer's power supply if it's plugged into the wall.

Checking Cabling

Several types of devices use cables, typically to link a device to the motherboard or to a controller card of some type. These cables can be entirely internal or external, depending upon the device type. Particular types of cable have specific requirements, which are discussed below.

Power Cables

The most obvious power cable to most users is the one that stretches from a wall outlet, power strip, or uninterrupted power supply (UPS) to the computer. This cable is much like power cables on many other devices, and it should be fully inserted into the computer and its power source.

A second class of power cables resides inside the computer case. These cables stretch from the power supply (a rectangular metal box inside the computer to which the external power cable attaches) to the motherboard and various disk devices (hard disk, floppy disk, CD-ROM drive, and so on). There are several different types of internal power connectors. Most power supplies have about half a dozen connectors of various forms, each of which connects to just certain types of devices—the motherboard, hard disk devices, or floppy devices. You should check that power connectors are all inserted firmly in their respective devices because they sometimes work loose during shipping.

So-called AT-style motherboards (used on many Pentium and earlier computers) used two motherboard power connectors, rather than the integrated connector used in later ATX systems. These AT connectors *must* be inserted side-by-side, with the black wires next to each other. These connectors can be inserted in each other's sockets, which will *destroy* the motherboard!

Internal Data Cables

The second major form of internal cabling is data cables. These carry data between components—typically between a disk or tape device and a motherboard or controller. The most common form of data cable is a *ribbon cable*, so called because the cable resembles a ribbon. Ribbon cables differ in their widths and in the exact forms of their connectors. Some also have unique characteristics, such as a twisted portion on floppy cables.

You should check that all cable connectors are inserted firmly and correctly. Most cables include notches or asymmetrical connectors so that they cannot be inserted backwards, but some cheap cables lack these safeguards. If some of your cables are so crippled, pay careful attention to the cable's orientation. Most cables include a colored stripe on one edge, which indicates the location of the first signal line. The matched connector on the device or board should indicate the location of pin #1, probably in tiny type by the connector. Be sure to plug the cable in so that the stripe is next to pin #1.

Some types of ribbon cable can have more connectors than devices. For instance, it's possible to use a SCSI cable with four connectors when you have just two SCSI drives, leaving one connector unused (two connectors attach to the SCSI drives and one to the host adapter). For most types of cable, you should ensure that the end connectors are both used. Normally, one of these attaches to the motherboard or controller card, and the other end attaches to one of the devices.

Particularly on older systems, ribbon cables sometimes link internal to external connectors. For instance, a motherboard might have an internal connector for its parallel port, so a ribbon cable ties this to an external parallel-port connector. Such cables are rare on modern motherboards, which integrate the connector into the motherboard in a standard location so that it's directly accessible from outside the case. You might still find such cables on a few designs, for instance if they are being used to link a USB port to a front-panel USB connector.

Ribbon cables aren't the only type of internal data cable. CD-ROM drives frequently sport 3-wire cables to tie the CD-ROM drive's audio output to a sound card. There are also 2–4-wire connectors that link the motherboard to front-panel computer components, such as the power button, the reset button, and the hard disk activity LEDs.

LED cables must be connected in the correct orientation, but the cables aren't keyed, so you've got a 50/50 chance of getting it wrong unless you pay careful attention to the positive and negative markings on the cables and motherboard. This detail isn't important for the power or reset switches on modern computers.

External Cables

External cables connect the computer to its keyboard, mouse, and monitor. Printers, scanners, network connections, and so on also use cables. (A few wireless devices exist, but even these often use short cables to link from a standard port to a radio or infrared transmitter.)

In all cases, for a device to function properly it's important that the associated cable be inserted firmly into its matching socket. Some cable types, such as Ethernet cables, snap into place and cannot be removed unless you push a small lever or similar locking mechanism. Others, such as parallel, RS-232 serial, and some varieties of external SCSI connectors, have thumbscrews that can be tightened to ensure a snug connection (some of these require an actual screwdriver to tighten and loosen). Others, such as USB and keyboard connectors, have no locking or tightening mechanism, so you must be sure these connectors are fully and firmly inserted.

Some cable types should not be routinely connected or disconnected when the computer is in operation. These include SCSI, RS-232 serial, and parallel connectors. When attaching or detaching such a cable, a short can damage the device or the computer. Other connectors, such as those for USB and Ethernet, are designed for *hot swapping*—attachment and detachment when the computer is in operation..

Because you'll be plugging external devices in yourself, you should be sure you do this job correctly. It's easy to mistakenly connect a device to the wrong port. This is particularly true for RS-232 serial devices since many computers have two such ports; for speakers, microphones, and audio inputs on sound cards; and for PS/2-style mice and keyboards. USB ports are interchangeable on most computers; it doesn't matter which one you use.

Some connectors are electrically compatible but come in different sizes or shapes. This is particularly true of RS-232 serial connectors (which come in 9- and 25-pin varieties), keyboard connectors (which come in large AT-style and small PS/2-style connectors), and external SCSI connectors (which come in several varieties). Adapters for these are available, but be cautious with them—the adapters can add enough weight to the connector so that it's likely to fall out. This is particularly true of one-piece keyboard adapters and some types of SCSI adapters.

Checking IRQs, DMA, and I/O Settings

Most plug-in boards use various hardware resources that are in limited supply in the *x*86 architecture. Of particular interest are the board's *interrupt request (IRQ)*, its *direct memory access (DMA)* channel, and its *input/output (I/O)* port. The *x*86 architecture supports just 15 interrupts (0–15, with IRQs 2 and 9 being the same). There are also just a handful of DMA channels. I/O ports are in less short supply, but still occasionally produce conflicts. Boards use an interrupt to tell the CPU that something important is happening that requires the CPU's attention. DMA channels and I/O ports are used to transfer data from the board to the computer's memory or CPU.

With ISA, it's important that two devices don't attempt to use the same IRQ, DMA channel, or I/O port. Doing so can result in one board being unavailable, and in extreme cases, it can crash the computer. PCI boards may be able to share an IRQ with another PCI board, but even this sometimes causes the hardware to work slowly or behave strangely.

The motherboard uses several IRQs for its own devices. For instance, EIDE drives normally use IRQs 14 and 15, and the keyboard takes IRQ 1.

If you have any old ISA boards, you can check their IRQs by examining jumper settings on the boards themselves. Consult the board's documentation for details. Newer ISA boards use software configuration, as described in Chapter 8. PCI boards are auto-configured by the computer's BIOS or by the Linux kernel. In both of these latter cases, it's impossible to tell what hardware resources a board will use without booting the computer. Unfortunately, if the resources cause a conflict, the computer may not boot completely. If you suspect this may be happening, consult Chapter 8 for hardware troubleshooting information.

Checking EIDE Devices

Most *x*86 computers use EIDE for hard disks, CD-ROMs, and often other types of disk and tape devices. There are several variants on EIDE available, ranging in speed from 8MBps to 100MBps, with faster speeds in the works. In 2001, 33MBps is considered low-end, 66MBps is common, and 100MBps is the interface of choice. These different interface types are referred to by various names, which usually include the speed, such as "UltraDMA 33" or

"ATA/66." The more capable interfaces can communicate with less-capable devices, and vice-versa, so you can mix and match if you need to—but each chain runs at just one speed, so you can seriously degrade a fast disk's performance by attaching it to the same cable as a slow CD-ROM or the like.

Each EIDE chain can support the controller and up to two devices. Traditionally, you must configure each device to be either the *master* or the *slave*. In Linux, the master device takes on a lower device letter in its `/dev/hdx` device filename, where *x* is the device letter. Configuring master/slave status is done through a switch or jumper on the device itself; consult your documentation for details. Some modern devices and controllers support an auto-configuration protocol.

If you need more than two devices, or if you want to separate fast and slow devices, you must use multiple EIDE chains, each of which corresponds to one physical EIDE cable. Most motherboards support two chains (hence four devices total), and you can add more by adding plug-in EIDE controller cards. You can use similar cards to upgrade to faster forms of EIDE.

Normally, one EIDE device will be the master on the first (or primary) chain. A second device might be the slave on the same chain or the master on a second chain. The former configuration preserves IRQs, which may be desirable if you have lots of other devices, but the second is likely to produce better performance.

Checking SCSI Devices

SCSI is an unusually capable and complex interface bus. For this reason, SCSI busses can sometimes be difficult to configure correctly, particularly when they're loaded down with many devices. There are several factors you should consider when planning or checking a SCSI bus:

SCSI Variant There are many versions of SCSI available, ranging from the original 5MBps SCSI-1 to the 160MBps Ultra3 Wide SCSI. Most of these versions are compatible with one another, but adding a less-capable device to an otherwise more-capable SCSI chain can degrade performance. Also, the more different two devices are, the less likely they are to get along on one chain. Adding a SCSI-1 device to an Ultra3 SCSI chain, for instance, is likely to cause problems.

SCSI IDs SCSI devices are differentiated by their ID numbers. Older SCSI variants (those that use a bus that's 8 bits wide) use ID numbers that range from 0 to 7, while Wide variants (which use 16-bit busses) have IDs that range from 0 to 15. The SCSI host adapter itself consumes one number, so this is the source of the 7- or 15-device limit on SCSI chains. SCSI IDs are generally set with jumpers on internal devices, or via some sort of switches or dial on external devices. Check your documentation for details. If two devices share an ID, it's likely that one will mask the other, or they'll both malfunction quite seriously. New devices can often use the SCSI Configured Automatically (SCAM) protocol, which allows devices to acquire IDs automatically.

Termination A SCSI bus can be thought of as a one-dimensional chain of devices. The devices on both ends of the chain must be terminated, which keeps signals from bouncing back from the end of the chain. There are several different types of termination associated with different SCSI variants, ranging from passive to active to low-voltage differential (LVD). Most SCSI devices include termination that can be activated by setting a jumper, or even automatically. Sometimes you need to add a separate SCSI terminator. Be sure this detail is set correctly because incorrect termination can lead to bizarre errors, which can crash a Linux system.

Cable quality SCSI—and particularly high-speed SCSI variants—is quite susceptible to problems caused by low-quality cables. Particularly if your SCSI chain has many devices, it can be worthwhile to purchase high-quality cables. These are, unfortunately, likely to be expensive—often $50 or more.

Cable length Maximum SCSI cable lengths range from 1.5 to 12 meters (m), depending upon the SCSI version. SCSI cable length limits apply to the *entire* SCSI chain. If you've got two external SCSI devices, for instance, you sum the lengths of the external cables, along with any internal cables, to determine your SCSI chain's cable length.

Chapter 8 includes information on troubleshooting a SCSI chain.

Checking BIOS Settings

The *Basic Input/Output System (BIOS)* is the lowest-level software component in a computer. The CPU runs BIOS code as part of its startup procedure. As a result, the BIOS configures many fundamental aspects of the computer

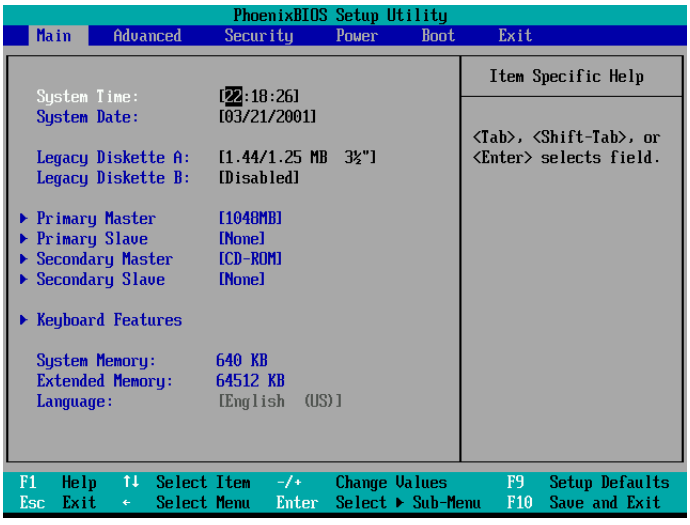
before Linux has a chance to boot. The BIOS also provides tools that the computer uses to load the Linux kernel into memory.

Although the *x*86 BIOS provides some standard features, it's not entirely standardized. In particular, modern BIOSes provide a setup tool, often referred to as the *Complementary Metal Oxide Semiconductor (CMOS) setup utility*, that you can use to set various low-level options. The options available in a computer's CMOS setup utility differ from one computer to another, both because of differences in hardware and because of different BIOS designs.

Most computers display a prompt at boot time that tells you how to get into the CMOS setup utility. This is usually done by hitting a key, such as Delete or F2, at a critical point during the boot process. Once you've done this, you'll see a BIOS setup screen, such as the one shown in Figure 1.3. This screen allows you to select and set various options, typically by moving through menus by pressing the arrow keys on the keyboard.

SCSI host adapters often include their own BIOSes and setup utilities, which are separate from the motherboard BIOS. The SCSI setup utilities usually have setup options you can adjust by pressing a key sequence at a particular point in the boot process. Watch your boot displays or consult your SCSI adapter's documentation for details.

FIGURE 1.3 CMOS setup utilities use menu-driven displays to let you adjust a computer's built-in hardware.



Most systems come with reasonable default BIOS settings, but you may want to check, and possibly adjust, a few. These include the following:

Disk settings There are two common hard disk settings you may need to adjust. The first specifies the size of the disk. An auto-detection feature normally works well for this. The second setting determines how the BIOS interprets the disk's cylinder/head/sector (CHS) addresses. On most BIOSes, a linear block addressing (LBA) mode is the best choice. If you use SCSI hard disks, the main motherboard BIOS won't detect them. This is normal; the SCSI BIOS provides the necessary support.

On-board ports Modern motherboards include RS-232 serial, parallel, USB, EIDE, and frequently other types of ports. You can enable or disable these or change their settings (for instance, you can change the IRQs used by the devices). Disabling unused ports can free up resources for other devices.

PCI settings Some BIOSes allow you to specify how the system treats PCI devices. Most commonly, you can choose from two or more rules for how the BIOS assigns IRQs to PCI devices. Sometimes, one rule results in IRQ conflicts and another doesn't, so such a setting is worth investigating if you have problems booting and suspect IRQ conflicts.

Passwords In a high-security environment, you may want to set a BIOS password. This prevents the system from booting unless the correct password is entered. It can slow down intruders who have physical access to the computer and boot with their own boot disk, but if intruders have physical access to the computer, they can bypass this feature in various ways. Setting a BIOS password also prevents automatic reboots in the event of a power failure. Nonetheless, slowing down an intruder may be worthwhile in some environments.

Memory settings BIOSes can be configured to copy parts of themselves, or of BIOSes stored on other devices, to RAM. This practice, which is known as shadowing, speeds up access to the BIOS, and it is useful in DOS, which relies on the BIOS for input/output. Linux doesn't use the BIOS as much, so it's generally best to disable all shadowing in Linux, which can result in slightly more memory available in Linux. Some BIOSes also allow you to control one or more memory holes—regions of the CPU's memory map that are unusable. These sometimes cause Linux to misdetect the amount of RAM installed in the computer, so you may want to experiment with different memory hole settings.

Boot Devices Modern BIOSes support booting from a wide variety of disk and disk-like devices, including floppy disks, EIDE disks, SCSI disks, CD-ROM drives, and high-capacity removable disks like Zip or LS-120 disks. You can usually set the system to boot from some subset of these devices in any order you like. The BIOS tries each medium in turn, and if it's not present or isn't bootable, it tries the next one. For highest security, set the system to boot from your EIDE or SCSI hard disk first; for convenient booting of installation or emergency media, set it to boot from a CD-ROM, floppy, or other removable media drive first.

In practice, you may need to experiment with a particular computer's CMOS settings to determine which work best. It's generally not a good idea to try random changes on a working system, though; experiment with these settings only if you're having trouble. Making changes without cause can produce an unbootable system, although if you remember what you changed, you can usually recover your system to a working state.

Most CMOS setup utilities include an option to restore the settings to the factory default values. These may not always produce optimal results, but they'll usually work

Planning Disk Partitioning

Hard disks can be broken into logical chunks known as *partitions*. In Windows, partitions correspond to drive letters (`C:`, `D:`, and so on). In Linux, partitions are mounted at particular points in the Linux directory tree, so they're accessible as subdirectories. Before actually installing Linux, it's a good idea to give some thought to how you'll partition your hard disk. A poor initial partitioning scheme can become awkward because you'll run out of space in one partition when another has lots of available space or because the partition layout ties your hands in terms of achieving particular goals.

Understanding the PC Partitioning System

The original *x*86 partitioning scheme allowed for only four partitions. As hard disks increased in size and the need for more partitions became apparent, the original scheme was extended in a way that retained compatibility

with the old scheme. The new scheme uses three partition types: *primary partitions*, which are the same as the original partition types; *extended partitions*, which are a special type of primary partition that serves as a placeholder for the next type; and *logical partitions*, which reside within an extended partition.

For any one disk, you're limited to four primary partitions, or three primary partitions and one extended partition. Many OSs, such as DOS, Windows, and FreeBSD, *must* boot from primary partitions, and because of this, most hard disks include at least one primary partition. Linux, however, is not so limited, so you could boot Linux from a disk that contains no primary partitions, although in practice few people do this.

The *x*86 partitioning scheme isn't the only one around. Linux includes support for many alternatives, but *x*86-based Linux systems generally use the PC partitioning scheme. Linux systems running on other architectures tend to use the partitioning systems native to those architectures. From an administrative point of view, these systems are almost always simpler than the PC system because there aren't any distinctions between primary, extended, and logical partitions.

A disk's primary partition layout is stored in a data structure known as the *partition table*, which exists on the first sector of the hard disk. This sector is known as the *master boot record (MBR)* because it also contains some of the first code to be run by the computer after the BIOS initializes. The locations of the logical partitions are stored within the extended partition, outside of the MBR. Although they are not a part of the MBR, these data are sometimes considered to be part of the partition table because they do define partition locations.

Linux Partition Requirements

To Linux, there's very little difference between the partition types. Linux numbers partitions on a disk, and the primary and extended partitions get the numbers from 1 to 4 (such as `/dev/hda1` or `/dev/sdc3`), while logical partitions get numbers from 5 up. This is true even if there are fewer than four primary and extended partitions, so partitions might be numbered 1, 2, 4, 5, and 6 (omitting partition 3). Primary partition numbers are like fixed slots, so when a disk uses just 1–3 of these slots, any of the four numbers may

go unused. Logical partitions, by contrast, are always numbered sequentially, without any missing numbers, so a system with precisely three logical partitions *must* number them 5, 6, and 7.

Some administrators use a primary Linux boot partition because a conventional *x*86 MBR can boot only from a primary partition. Therefore, it's possible to put the Linux Loader (LILO; the program Linux uses to boot) in the Linux boot partition to boot the computer. Alternatively, LILO can reside directly in the MBR, which is more direct but leaves LILO more vulnerable to being wiped out should some other utility rewrite the MBR. (Chapter 3, "Software Management," discusses the boot process and LILO in more detail.)

As a bare minimum, Linux needs a single partition to install and boot. This partition is referred to as the *root partition*, or as `/`. This partition is so called because it lies at the "root" of the directory "tree"—all files on the system are identified relative to the base of this partition. The root partition also stores directories, such as `/etc` and `/usr`, in which other files reside. Some of these directories can serve as *mount points*—directories to which Linux attaches other partitions. For instance, you might mount a partition on `/home`.

One important directory in Linux is `/root`, which serves as the system administrator's home directory—the system administrator's default program settings and so on go here. The `/root` directory is not to be confused with the root directory (`/`).

Common Optional Partitions

In addition to the root partition, many system administrators like creating additional partitions. Several of the advantages that come from splitting an installation into multiple partitions rather than leaving it as one monolithic root partition follow:

Multiple disks When you have two or more hard disks, you *must* create separate partitions—at least one for each disk. For instance, one disk might host the root directory and the second might hold `/home`. Also, removable disks (floppies, CD-ROMs, and so on) must be mounted as if they were separate partitions.

Better security options By breaking important directories into separate partitions, you can apply different security options to different partitions. For instance, you might make `/usr` read-only, which reduces the chance of accidental or intentional corruption of important binary files.

Data overrun protection Some errors or attacks can cause files to grow to huge sizes, which can potentially crash the system or cause serious problems. Splitting key directories into separate partitions guarantees that a runaway process in such a directory won't cause problems for processes that rely on the ability to create files in other directories. This makes it easier to recover from such difficulties. On the down side, splitting partitions up makes it more likely that a file will legitimately grow to a size that fills the partition.

Disk error protection Disk partitions sometimes develop data errors, which are data structures that are corrupt, or a disk that has developed a physically bad sector. If your system consists of multiple partitions, such problems will more likely be isolated to one partition, which can make data recovery easier or more complete.

Backup If your backup medium is substantially smaller than your hard disk, breaking up your disk into chunks that fit on a single tape can simplify your backup procedures.

Ideal filesystems A *filesystem* is a set of low-level data structures that regulate how the computer allocates space on the disk for individual files, as well as what types of data are associated with files, such as file creation times and filenames. Sometimes, one filesystem works well for some purposes but not for others. You might therefore want to break the directory tree into separate partitions so that you can use multiple filesystems.

So, what directories are commonly split off into separate partitions? Table 1.1 summarizes some popular choices. Note that typical sizes for many of these partitions vary greatly depending upon how the system is used. Therefore, it's impossible to make recommendations on partition size that will be universally acceptable. For more information, consult "Understanding the Linux Filesystem Hierarchy" in Chapter 7 ("Managing Partitions and Processes").

TABLE 1.1 Common Partitions and Their Uses

Partition (mount point)	Typical size	Use
Swap (not mounted)	1.5–2 times system RAM size	Serves as an adjunct to system RAM; is slow, but allows the system to run more or larger programs. Discussed in more detail in Chapter 8.
`/home`	200MB–100GB	Holds users' data files. Isolating it on a separate partition preserves user data during a system upgrade. Size depends on number of users and their data storage needs.
`/boot`	5–20MB	Holds critical boot files. Creating as a separate partition allows for circumventing limitations of certain BIOSes and boot loaders on hard disks over 8GB.
`/usr`	500MB–4GB	Holds most Linux program and data files; this is frequently the largest partition.
`/usr/local`	100MB–2GB	Holds Linux program and data files that are unique to this installation, particularly those that you compile yourself.
`/opt`	100MB–2GB	Holds Linux program and data files that are associated with third-party packages, especially commercial ones.
`/var`	100MB–100GB	Holds miscellaneous files associated with the day-to-day functioning of a computer. These files are often transient in nature. Most often split off as a separate partition when the system functions as a server that uses the `/var` directory for server-related files like mail queues.

TABLE 1.1 Common Partitions and Their Uses *(continued)*

Partition (mount point)	Typical size	Use
/tmp	100MB-10GB	Holds temporary files created by ordinary users.
/mnt	N/A	/mnt isn't itself a separate partition; rather, it or its subdirectories are used as mount points for removable media like floppies or CD-ROMs.

Some directories should *never* be placed on separate partitions. These directories are /etc, /bin, /sbin, /lib, and /dev. These directories host critical system configuration files or files without which a Linux system cannot function. For instance, /etc contains /etc/fstab, the file that specifies what partitions correspond to what directories; and /bin contains the mount utility that's used to mount partitions on directories.

The 2.4.*x* kernels include experimental support for a dedicated /dev file-system, which obviates the need for files in an actual /dev directory, so in some sense, /dev can reside on a separate filesystem, although not a separate partition.

Real World Scenario

When to Create Multiple Partitions

One problem with splitting off lots of separate partitions, particularly for new administrators, is that it can be difficult to settle on appropriate partition sizes. As noted in Table 1.1, the appropriate size of various partitions can vary substantially from one system to another. For instance, a workstation is likely to need a fairly small /var partition (say, 100MB), but a mail or news server might need a /var partition that's gigabytes in size. Guessing wrong isn't fatal, but it is annoying. You'll need to resize your partitions (which is tedious and dangerous) or set up symbolic links between partitions so that subdirectories on one partition can be stored on other partitions.

For this reason, I generally recommend that new Linux administrators try simple partition layouts first. The root (/) partition is required, and swap is a very good idea. Beyond this, `/boot` can be very helpful on hard disks of more than 8GB with older distributions or BIOSes. An appropriate size for `/home` is often relatively easy for new administrators to guess, so splitting it off generally makes sense. Beyond this, I recommend that new administrators proceed with caution.

As you gain more experience with Linux, you may want to break off other directories into their own partitions on subsequent installations, or when upgrading disk hardware. You can use the `du` command to learn how much space is used by files within any given directory.

Linux Filesystem Options

Linux supports many filesystems. The most popular in 2001 for Linux partitions is the *second extended filesystem (ext2* or *ext2fs)*, which is the default filesystem for most distributions. Ext2fs supports all the features required by Linux (or by Unix-style OSs in general), and is well tested and robust.

Ext2fs has one major problem, though: If the computer is shut down improperly (because of a power outage, system crash, or the like), it can take several minutes for Linux to verify an ext2fs partition's integrity when the computer reboots. This delay is an annoyance at best, and it is a serious problem on mission-critical systems such as major servers. The solution is implemented in what's known as a *journaling filesystem*. Such a filesystem keeps a record of changes it's about to make in a special journal log file. Therefore, after an unexpected crash, the system can examine the log file to determine what areas of the disk might need to be checked. This design makes for very fast checks after a crash or power failure—a few seconds at most, typically.

Four journaling filesystems are being developed for Linux in 2001. The most usable of these in mid-2001 is ReiserFS (`http://www.namesys.com`), which was added as a standard component to the 2.4.1 kernel. The other journaling filesystems are ext3fs (`ftp://ftp.uk.linux.org/pub/linux/sct/fs/jfs`), which is an extension of ext2fs; XFS (`http://linux-xfs.sgi.com/projects/xfs`), which was originally designed for Silicon Graphics' IRIX OS; and Journaled Filesystem (JFS) (`http://oss.software.ibm.com/developerworks/opensource/jfs`), which IBM developed for

its AIX and OS/2. Of these four, XFS and JFS are the most advanced, but ReiserFS is the most stable and is usable in mid-2001.

The Linux swap partition doesn't use a filesystem per se. Linux does need to write some basic data structures to this partition in order to use it as swap space (as described in Chapter 8), but this isn't technically a filesystem because no files are stored within it.

Linux also supports many non-Linux filesystems, including the File Allocation Table (FAT) filesystem used by DOS and Windows; the New Technology Filesystem (NTFS) used by Windows NT and 2000; the High-Performance Filesystem (HPFS) used by OS/2; the Unix Filesystem (UFS; also known as the Fast Filesystem, or FFS) used by various versions of Unix; the Hierarchical Filesystem (HFS) used by MacOS; and the ISO-9660 and Joliet filesystems used on CD-ROM. Most of these filesystems are useful mainly in dual-boot configurations—say, to share files between Linux and Windows. Some—particularly FAT, ISO-9660, and Joliet—are useful for exchanging files between computers on removable media. As a general rule, these filesystems can't hold critical Linux files because they lack necessary filesystem features. There are exceptions, though—Linux sports extensions to cram necessary information into FAT and HPFS partitions, UFS was designed for storing Unix filesystem features in the first place, and the Rock Ridge extensions add the necessary support to ISO-9660.

It's often best to use ext2fs for Linux partitions, although ReiserFS is a good choice if you have large partitions and want to avoid lengthy filesystem checks on system startup. ReiserFS isn't as well tested as ext2fs, though; so as of kernel 2.4.2, ext2fs is still safer. XFS, JFS, and ext3fs may become viable options for production systems by the end of 2001 or 2002. XFS and JFS have the advantage of supporting larger file sizes than the 4GB maximum allowed by ext2fs and ReiserFS. All Linux distributions support ext2fs out of the box, and most released in 2001 support ReiserFS, as well. Others may require modifying the kernel, so you must initially install Linux to a supported filesystem and then convert or add the other filesystem types. Use non-Linux filesystems for data exchange with non-Linux systems.

Partitioning Tools

In order to create partitions, you use a partitioning tool. There are dozens of such tools available, but only a few make reasonable choices when installing a Linux system:

DOS's `FDISK` Microsoft's DOS and Windows ship with a simple partitioning tool known as `FDISK` (for fixed disk). This program is inflexible and uses a crude text-based user interface, but it's readily available and can create partitions that Linux can use. (You'll probably have to modify the partition type codes using Linux tools in order to use DOS-created partitions, though.)

Linux's `fdisk` Linux includes a partitioning tool that's named after the DOS program, but the Linux tool's name is entirely lowercase, whereas the DOS tool's name is usually written in uppercase. Linux's `fdisk` is much more flexible than is DOS's `FDISK`, but it also uses a text-based user interface. If you have an existing Linux emergency disk, you can use it to create partitions for Linux before installing the OS.

Linux install-time tools Most Linux installation utilities include partitioning tools. Sometimes the installers simply call `fdisk`, but other times they provide GUI tools that are much easier to use . If you're installing a Linux-only system, using the installer's tools is probably the best course of action.

PowerQuest's PartitionMagic PowerQuest (`http://www.powerquest.com`) makes an unusually flexible partitioning program, known as PartitionMagic. This commercial program provides a GUI interface and can create partitions that are prepared with FAT, NTFS, HPFS, or ext2 filesystems. This makes it an excellent tool for configuring a disk for a multi-OS computer. PartitionMagic can also resize a partition without damaging its contents. The main program is Windows-based, but the package comes with a DOS version that can run from a floppy, so it's possible to use it on a system without Windows.

FIPS The First Nondestructive Interactive Partition Splitting (FIPS) program comes with many Linux distributions. It's a fairly specialized partitioning tool that splits a single primary FAT partition into two partitions. It's designed to make room for Linux on computers that already have Windows installed—you run FIPS, delete the second (empty) partition that FIPS creates, and create Linux partitions in that empty space.

In theory, partitions created by any tool may be used in any OS, provided the tool uses the standard *x*86 partition table. In practice, though, OSs sometimes object to unusual features of partitions created by certain partitioning tools. Therefore, it's usually best to take one of two approaches to disk partitioning:

- Use a cross-platform partitioning tool like PartitionMagic. Such tools tend to create partitions that are inoffensive to all major OSs.
- Use each OS's partitioning tool to create that OS's partitions.

Chapter 2 includes a discussion of partitioning during installation of a Linux Mandrake system. Although other distributions' partitioning tools differ somewhat, the basic principles remain the same across distributions. Before you invest too much effort in partitioning, though, you need to study Linux software issues, not the least of which is whether Linux is the proper tool to use.

Linux and Non-Linux Solutions

Why use Linux? This may seem like an odd question to ask in a book on Linux, but it's an important one, and one that you should ask yourself whenever you begin configuring a new Linux computer. If you find that another OS is better suited to a particular task, you can save yourself time and effort in the long run by using that other OS. When Linux is really the best choice, asking yourself why you should use it will give you a ready answer should somebody else ask you the question. By weighing Linux against other OSs, you will also increase your confidence that you're doing the right thing by choosing Linux. This activity may also lead you to think about the computer's purpose in a way that helps you decide precisely how to configure it.

Another important aspect to consider is *which* flavor of Linux to use. Many companies and organizations package Linux. Each of the resulting distributions has its own mix of features. In the end, you can configure any distribution to do what any of the others does, but some distributions are better suited to particular tasks right out of the box. Therefore, it's helpful to know something about the range of what's available before you begin configuring a new Linux system.

Linux vs. Proprietary OSs

Linux competes, to a greater or lesser extent, with many OSs. One of the great divides is between Linux and *proprietary* OSs—operating systems that use a source code base that's closed to public scrutiny. In particular, Linux competes against Microsoft Windows in many peoples' minds. In many respects, though, Linux is more properly pitted against commercial versions of Unix.

Most proprietary OSs are at a cost disadvantage when compared to Linux. Price tags on commercial OSs range from around $50 up to thousands of dollars. Most Linux distributions can be downloaded for free from the Internet, obtained on CD-ROM for less than $10 if you don't need official support or a printed manual or for $100 or less with these extras. Some extra-deluxe Linux packages cost more than this, often because they come bundled with some costly commercial packages or extended support options.

Linux vs. Microsoft OSs

Microsoft makes two OS product lines: Windows 9*x*/Me and Windows NT/2000. Each OS is targeted at a particular market, although they overlap to some extent. Linux competes against both, although more directly against the NT/2000 line.

The Windows 9*x*/Me line is targeted at home and relatively low-powered business desktop systems. This OS is derived from the old MS-DOS and Windows 3.1 products, and this legacy continues to influence Windows 9*x*/Me, in terms of filesystem support, multitasking power (the ability to run multiple programs at once), and so on. One of the drawbacks of Windows 9*x*/Me is that it's saddled with the need for backward compatibility with its outmoded predecessors. This fact limits the stability of Windows 9*x*/Me and its suitability for advanced network server functions. Many desktop users, however, aren't particularly bothered by these limits, and they are drawn to the easy-to-use Windows user interface. Although Linux is far more stable than Windows 9*x*/Me, Linux is playing catch-up in the user interface department. Therefore, if a computer will be used as a personal productivity workstation, Windows Me deserves consideration. This is particularly true if the individual who'll be using the machine is already familiar with Windows and doesn't want to learn something else.

Microsoft markets Windows NT/2000 as the competition to Unix and Linux. This OS branch uses a newer kernel with better support for features that are important today, such as filesystem security and multitasking. This makes Windows NT/2000 much more suited to function as a network server than Windows 9*x*/Me. One of the key differences between Windows NT/2000 and Linux is that the former is much more tightly tied to its graphical user interface (GUI). This fact makes Windows NT/2000 easier for new administrators to pick up, but at the same time, it reduces the OS's flexibility. With Linux, on the other hand, you can customize configuration files in ways a GUI doesn't allow, and you can do other things that are difficult or impossible to do using Windows NT/2000. The fact that Linux can be configured through text-based tools also means that it's easy to administer remotely, using nothing more than common remote login tools such as Secure Shell (SSH), which allows you to run text-based Linux programs from another computer.

Although Microsoft no longer markets it, OS/2 is another OS in the same family as DOS and Windows. This OS is still sold by IBM, but it's been sliding in market share since the mid-1990s. In many respects, OS/2 is similar to Windows NT—it uses its own updated kernel that abandons much of the DOS baggage still carried by Windows 9*x*/Me. OS/2 lacks compatibility with today's popular Windows software, however. OS/2 may still be worth considering in environments where it's still heavily used, such as many banks, but its declining market share argues against its adoption in new environments without some compelling cause.

One of the most compelling arguments in favor of any Microsoft or Microsoft-related OS is the application base for desktop use. In particular, the Microsoft Office application group is extraordinarily popular, and it is not available for Linux (or OS/2, for that matter). Although some Linux programs can import and export Office files, these operations are necessarily imperfect. Likewise, emulators like WINE, VMware, and Win4Lin allow you to run Windows software under Linux, but if the primary reason to use a computer is to use Windows software, there's seldom an advantage to running that software in an emulator under Linux, as opposed to running it directly.

Likewise, the principal advantage for Linux in a Linux-vs.-Windows comparison is its software base. Linux supports a wide range of open source (and therefore generally low-cost) applications. Because Linux is modeled after Unix, administrators with Unix experience should have no trouble handling

a Linux system—and a person who learns Linux can pick up other flavors of Unix quite quickly.

Linux vs. Unix

Originally, Linux was developed as a clone of Unix. Linus Torvalds set out to write an open source kernel around which existing open source Unix replacement parts could converge; the result was the Linux distribution as we know it today. Because of this history, modern Linux systems bear a very strong resemblance to modern Unix systems, and in fact, the two can often be used similarly.

In many cases, the main difference between Linux and a commercial Unix is cost. As noted earlier in this section, Linux is a very low-cost OS, but commercial Unixes cost much more. (The most common *x*86 Unixes are now available at low cost for personal use, but most commercial users must still pay hundreds of dollars for a license.) Most Linux software is available in open source form, and it can be compiled on commercial Unix machines—indeed, most Linux software is developed as Unix software generically, with Linux as just one of many Unix-like platforms on which it works. Many commercial Unix programs have been ported to (that is, recompiled on) Linux.

Where commercial Unixes hold an edge is in very high-performance computing. OSs such as Silicon Graphics' (SGI's) IRIX and Sun's Solaris run on very fast non-*x*86 hardware and support advanced features that Linux supports poorly, if at all. For instance, IRIX's XFS is still considered beta quality on Linux. Also, the hardware used by high-end systems is often superior to that used on the *x*86 PCs on which Linux usually runs. Linux has been ported to many non-*x*86 platforms, including many of those on which its Unix "big brothers" run, but these ports often lag behind the *x*86 version in terms of overall polish and general usability.

As Linux improves, the gap between Linux and commercial versions of Unix is likely to shrink. Even today, Linux is an excellent platform for workstations and small to mid-size servers. Because of its similarity to more advanced systems, it's possible to deploy Linux today and move to a higher-end commercial Unix system in the future, with minimal changes to configuration and administrators' training. This is certainly an advantage of the Linux/Unix family as a whole over Windows.

Linux vs. Other Open Source OSs

Linux is not the only open source OS in existence. Most competing open source OSs are, like Linux, clones of Unix. In fact, the main competing family—FreeBSD, NetBSD, and OpenBSD—is derived directly from mainstream Unix.

BSD stands for *Berkeley Standard Distribution*. BSD grew as a component-by-component open source replacement for AT&T's original Unix. During and after this process, the development forked several times, producing several variant products, including the three major open source BSDs. Today, the term BSD refers either to an entire OS that shares the Berkeley heritage or to specific OS components that are so derived, such as the BSD printing system discussed in Chapter 8.

In most situations, one of the open source BSDs will function as well as Linux. These OSs have more-or-less the same base of applications, and they're administered in largely the same ways. The main differences between Linux and its BSD cousins come down to three factors:

Kernel licenses The Linux kernel is released under the General Public License (GPL), which tends to encourage greater public participation in the development of the kernel than does the BSD license used for the BSDs. The culture that's emerged around each OS has furthered this distinction. The end result is that the Linux kernel has developed more quickly than the BSD kernels, and it has more support for more hardware, more filesystems, and so on.

Commercial software availability Commercial software developers seem to be more willing to port software to Linux than to the open source BSDs. The BSDs include ways to run Linux programs, and most software for both systems is open source and available on both, so this factor isn't critical for most people, but it is, nonetheless, a plus in the Linux column.

Support network The Linux support network, as embodied in the Linux newsgroups, Web sites, and support from commercial Linux vendors, is generally more active than is the support network for the BSDs. This helps new Linux users get started with Linux, and it helps even experienced administrators resolve problems. Some support forums, though, are

OS-neutral, such as mailing lists and newsgroups devoted to programs that run on both platforms.

These three factors produce a positive feedback cycle—more users creates a better volunteer support network, more potential kernel developers, and more incentive for commercial software vendors to port their products to Linux. Each of these factors in turn creates an environment that will attract even more users.

In the end, you're probably best off using whichever Unix-like open source OS is most familiar to you. For new users, the broader support network for Linux can be a major point in its favor. If you can access personal support for a BSD more easily, though, or if a BSD supports hardware that you need that's not supported in Linux, then a BSD may be a better choice for you. In either case, moving from Linux to a BSD or vice-versa is fairly straightforward, so the time you spend learning one system is not wasted should you decide to change.

A Rundown of Linux Distributions

Within the Linux world, there exist several *distributions*. A distribution is a compilation of a Linux kernel, startup scripts, configuration files, and critical support software. Distributions also include some type of installation routine so that you can get a working Linux system. Any two distributions may use different versions of any or all of these components, which will produce distinctly different feels. Critical components, though, such as the kernel and certain support software, come from the same line in all distributions. For instance, one distribution might use the 2.4.2 Linux kernel and another might ship with 2.4.3, but they're both Linux kernels.

One important distinguishing characteristic of Linux distributions is which packaging methods they use. Red Hat Package Manager (RPM), Debian packages, and tarballs are the three most common package formats. The details of using these three package formats are covered in Chapter 3.

Depending upon your definition of "major," there are anywhere from two or three to a dozen or more major Linux distributions. In addition, there are less popular and specialized distributions. Many Linux distributions are

derived from either Debian or Red Hat. Some common Linux distributions include the following:

Caldera eDesktop and eServer These distributions, from Caldera (`http://www.caldera.com`), are targeted at workstation and server use, respectively. Both spring from the earlier OpenLinux product. These distributions are RPM-based and include moderately sophisticated GUI configuration tools. Although Caldera is RPM-based, its distributions aren't directly derived from Red Hat Linux. Caldera's distributions are available only for *x*86 CPUs.

Corel Linux Corel (`http://linux.corel.com`) based its distribution on Debian GNU/Linux, but it added a very user-friendly installation routine and GUI configuration tools. In doing so, though, Corel made its distribution less easily configured through traditional Linux command-line methods. This distribution is targeted at new Linux users who want to use the OS as a desktop OS to replace Windows. Corel is an *x*86-only distribution.

Debian GNU/Linux This distribution, headquartered at `http://www.debian.org`, is built by a non-profit organization, rather than by a for-profit company, as are most other distributions. Debian eschews many of the GUI configuration tools used by most other distributions, and instead it aims to be a very stable and flexible distribution. For these reasons, it's well liked by open source hard-liners and those who like tinkering with the underlying text-based configuration files. Debian is available on a very wide array of CPUs, including *x*86, PowerPC, Alpha, SPARC, and 680*x*0.

Linux Mandrake This distribution is a French-based offshoot of Red Hat Linux. Originally developed as a Red Hat with integrated K Desktop Environment (KDE), Mandrake has since developed more of its own personality, which includes a good GUI installer and some unusual choices in standard server software, such as Postfix rather than the more popular sendmail for a mail server. Its English Web page is `http://www.linux-mandrake.com/en`. Mandrake is available for *x*86, IA-64, SPARC, Alpha, and PowerPC CPUs.

LinuxPPC This distribution is a Red Hat derivative for PowerPC (PPC) processors—the CPUs at the heart of modern Macintoshes. LinuxPPC is very similar to Red Hat, but the GUI installation routines are unique, designed for the Macintosh market. The LinuxPPC Web site is `http://www.linuxppc.com`.

Red Hat Linux Red Hat (`http://www.redhat.com`) is one of the older major distributions today, and one of the most influential. Red Hat developed the RPM format that's used by many other distributions, including some that aren't otherwise based on Red Hat. The distribution includes GUI installation and configuration tools that are unusually complete. Red Hat is or has been available on *x*86, IA-64, SPARC, and Alpha CPUs, although Red Hat has ceased SPARC development with version 6.2.

Slackware Linux Slackware is the oldest of the surviving Linux distributions. Like Debian, Slackware favors manual text-based configuration over GUI configuration tools, so it's often recommended for those who want the "Unix experience" without GUI "crutches." Slackware is the only major distribution to rely upon tarballs for package management. You can read more at `http://www.slackware.com`. This distribution is available for *x*86, Alpha, and SPARC CPUs.

Storm Linux This distribution, from Stormix (`http://www.stormix.com`), is another Debian variant. Like Corel Linux, Storm Linux adds GUI installation and configuration tools to Debian's core, but Storm Linux is less tightly tied to these tools. Storm Linux is available only on *x*86 CPUs.

SuSE Linux The German company SuSE (`http://www.suse.com`) produces a distribution that's particularly popular in Europe. SuSE uses RPMs, but it's not otherwise based on Red Hat. Some SuSE packages use a DVD-ROM for software distribution, which is very helpful if your system has a DVD-ROM drive—SuSE ships with an unusually large number of packages, so juggling the half-dozen CD-ROMs can be awkward, compared to using a single higher-capacity DVD-ROM. This distribution includes GUI installation and configuration tools. Versions of SuSE for *x*86, IA-64, PPC, and Alpha are all available.

TurboLinux This distribution (`http://www.turbolinux.com`) is a Red Hat derivative. As of early 2001, the TurboLinux installation routines are somewhat simple, but effective. This distribution includes unusually strong support for Asian languages, and is targeted at the server market. TurboLinux is available for *x*86, IA-64, and Alpha CPUs.

Yellow Dog Linux Like LinuxPPC, this is another PPC distribution based on Red Hat. Yellow Dog (at `http://www.yellowdoglinux.com`) uses its own unique installer, but once set up, it and LinuxPPC are quite similar to one another—and to Red Hat.

When deciding on a Linux distribution, some of these will fall out of the running for very basic reasons. For instance, there's no point in considering Yellow Dog for an *x*86 system, or Corel for an Alpha CPU. The RPM and Debian package management systems are, on the whole, quite similar in overall features and capabilities, so if you're not already familiar with either, there's little reason to favor one over the other. (Chapter 3 covers both systems in more detail.) Any of these distributions can be configured to do anything that any other can do, with the exception of running on an unsupported CPU.

As a practical matter, you *do* need to decide between distributions. As a general rule, Caldera eDesktop, Corel, and Mandrake are probably the best suited as delivered to function as workstations, particularly for new Linux users. SuSE ships with an unusually wide array of software (particularly the Professional package, which ships with a DVD-ROM and half a dozen CD-ROMs). Red Hat is unusually popular, so finding support for it on newsgroups and the like is particularly easy. Caldera eServer and TurboLinux are specifically marketed for the server market, but others can fill that role just as easily. Some distributions come in variants that include additional software, such as secure servers, third-party partition managers, and so on.

If you have a fast Internet connection, a CD-R drive, and you want to experiment with several Linux distributions, check out the Linux ISO Web site, `http://www.linuxiso.org`. This site includes links to CD-R image files for most Linux distributions. You can also obtain distributions on no-frills CD-ROMs (with no manual and no support) for less than $10 from the likes of Linux Mall (`http://www.linuxmall.com`), Linux System Labs (`http://www.lsl.com`), or CheapBytes (`http://www.cheapbytes.com`). Official boxed sets typically cost $20 to $100, or occasionally more for the most feature-packed versions. The boxed sets typically include printed manuals, support, and occasionally, a commercial software product or two.

Determining Software Needs

When you plan a Linux installation, it's important that you know what software you'll need on the system. For each program class, you'll need to decide what particular package you want to run. For instance, if you want to configure a word processing workstation, you'll need to decide if you want to use Corel's WordPerfect, Sun's StarOffice, Applix's ApplixWare, the open source LyX, or something else. Some of these packages come with certain

distributions; others must be obtained independently. In the case of downloadable software, if it doesn't come with the distribution you use, you may want to download it before installing Linux. Depending upon your available hardware, you can usually put a package on floppy disk, a high-capacity removable disk (like a Zip or LS-120 disk), or a CD-R to have it ready for installation once you've installed the main distribution. Doing this from Windows works just fine, if this is your first Linux installation.

Common Workstation Programs

Workstations don't usually need much in the way of server software. Workstations may include such software to provide local services, though—for instance, Linux workstations usually include mail servers to handle mail for the administrator that is generated by automatic scripts and the like. The most important workstation programs are designed to help an individual get work done.

The X Window System

The X Window System (or X for short) is Linux's GUI environment. It's usually implemented through the XFree86 package. Although Linux can be used without this GUI, most workstation users expect a GUI environment, and an increasing number of workstation programs require X in order to function.

X itself is a fairly spare environment, so it's frequently supplemented by additional tools, such as *window managers* (which provide borders and controls around windows) and *desktop environments* (which include a window manager and an assortment of utility programs to help make for a comfortable working environment). In particular, the K Desktop Environment (KDE; `http://www.kde.org`) and the GNU Network Object Model Environment (GNOME; `http://www.gnome.org`) are two popular desktop environments for Linux. Most Linux distributions in 2001 ship with both, but some install one or the other by default. Red Hat, for instance, favors GNOME, whereas Mandrake favors KDE.

Office Tools

Office tools are the workhorses of computer use in offices; they are primarily made up of word processors, spreadsheets, and databases, but they may also contain various other applications, such as personal contact managers, calendar programs, and so on. Corel's WordPerfect Office and Sun's (`http://www.sun.com`) StarOffice are both available in both Linux and Windows, and so they can be good choices in a mixed Linux/Windows environment.

Applix's (http://www.applix.com) ApplixWare is another competing office suite. All three are commercial products, although Sun has released StarOffice as a free download and a stripped-down version of the old version 8 of the WordPerfect word processor is also available for free. All three of these products also include import/export filters for Microsoft Office documents, but as noted earlier, this approach is imperfect at best. (StarOffice is generally considered to have the best of these filters.)

In the open source arena, various packages are available, mostly as singleton programs rather than integrated office suites. For instance, LyX (http://www.lyx.org) and AbiWord (http://www.abisource.com) are two popular "what you see is what you get" (WYSIWYG) Linux word processors. There are also markup languages like TeX and LaTeX (http://www.latex-project.org) that, in conjunction with editors like Emacs, can do much the same job. Gnumeric (http://www.gnome.org/projects/gnumeric) is a popular Linux spreadsheet. Both the GNOME (http://www.gnome.org) and KDE (http://www.kde.org) projects are building open source office suites, largely by working on integrating existing products.

Network Clients

Users run network client programs to access network resources. Examples include Web browsers like Netscape (http://www.netscape.com) and Opera (http://www.opera.com), mail readers like Mutt (http://www.mutt.org) and KMail (part of KDE, http://www.kde.org), and FTP clients like gFTP (http://gftp.seul.org). All major Linux distributions ship with a wide variety of network clients, but if you need a specific program, you should check whether it's included in your distribution. If it's not, track it down and install it. Most Linux network clients are open source, but there are a few that aren't. Opera stands out in this respect.

For more information on network clients, please refer to Chapter 5, "Networking".

Audio/Visual Programs

Audio/Visual programs cover quite a wide range of products. Examples include graphics viewers and editors like XV (http://www.trilon.com/xv) and The GIMP (http://www.gimp.org); ray tracing programs like POV-Ray (http://www.povray.org); multimedia players like XAnim (http://xanim.va.pubnix.com); audio/video editors like Broadcast

(http://heroines.sourceforge.net/bcast2000.php3) and Linux Video Studio (http://ronald.bitfreak.net); and games like Civilization: Call to Power and Alpha Centauri (both from Loki, http://www.lokigames.com). Some audio/visual programs are serious tools for work and are on par with office utilities for some users. Somebody whose work involves graphics design, for instance, may need tools like The GIMP or POV-Ray. Other audio/visual programs fall more in the realm of entertainment, like games.

Linux's support for audio/visual programs has traditionally been weak. This has changed substantially since the mid-1990s, however, with the development of powerful programs like The GIMP and increasingly sophisticated multimedia players and editors. Even Linux games have come a long way, thanks largely to companies like Loki that specialize in porting other companies' games to Linux.

Personal Productivity Tools

Personal productivity tools are programs that individuals use to better their own lives. Examples include personal finance programs like GnuCash (http://www.gnucash.org) and slimmer versions of office programs (word processors for writing letters, for instance). As with audio/visual programs, personal productivity applications have traditionally been lacking in Linux, but that situation is improving. GnuCash, in particular, fills a niche that many users find important for personal use of Linux.

Personal productivity tools need not be restricted to the home, however. For instance, although big word processors like StarOffice and WordPerfect are very useful in some situations, many office users don't need anything nearly so powerful. Slimmer tools like Maxwell (http://www.eeyore-mule.demon.co.uk) suit some users' needs just fine. By foregoing the resource requirements of a larger package, using such programs can help save money by allowing employees to use less powerful computers than might otherwise be required.

Scientific Programs

Unix systems have long been used in scientific research, and Linux has inherited a wealth of specialized and general scientific tools. These include data plotting programs such as the GNU plotutils package (http://www.gnu.org/software/plotutils/plotutils.html) and SciGraphica (http://scigraphica.sourceforge.net), data processing programs like

Stata (`http://www.stata.com`), and many very specialized programs written for specific studies or purposes. Linux's software development tools (described shortly, in "Compilers") let you or your users write scientific programs, or compile those written by others.

Common Server Programs

A *server program* is one that provides some sort of service, usually to other systems via a network connection. Typically, a server runs in the background, unnoticed by the computer's users. In fact, many computers that run server programs don't have ordinary login users; instead, the system's users are located at other systems, and they use the computer only for its servers. A Web server computer, for instance, may not have any local users aside from those who maintain the computer and its Web pages.

The term server is sometimes applied to an entire computer, as in "the Web server needs a bigger hard disk." Context is usually sufficient to distinguish this use from the use of the term in reference to a specific software product.

Web Servers

One very popular use of Linux is as a platform for running a Web server. This software uses the *Hypertext Transfer Protocol (HTTP)* to deliver files to users who request them with a Web client program, more commonly known as a Web browser. The most popular Web server for Linux by far is Apache (`http://www.apache.org`), which is an open source program included with Linux. There are other Linux Web servers available, however, including Roxen (`http://www.roxen.com/products/webserver`) and `thttpd` (`http://www.acme.com/software/thttpd`). Roxen is a high-powered commercial Web server, whereas `thttpd` is a minimalist open source program suitable for small Web sites or those that don't need advanced features.

Some Linux distributions install Web servers even on workstations because the distributions use the Web servers to deliver help files to the local users. Such a configuration chews up resources, though, and can at least potentially be a security problem.

Mail Servers

Mail servers handle e-mail delivery. All major Linux distributions ship with a mail server, such as sendmail (`http://www.sendmail.org`), Exim (`http://www.exim.org`), or Postfix (`http://www.postfix.org`). These servers all handle the Simple Mail Transfer Protocol (SMTP), which is used to deliver mail between mail servers on the Internet at large, and can also be used as part of a local network's e-mail system. All major Linux distributions also ship with Post Office Protocol (POP) and Internet Message Access Protocol (IMAP) servers. These are used to deliver mail to end-user mail reader programs, which typically reside off of the mail server. Most Linux SMTP, POP, and IMAP servers are open source, although there are commercial servers available as well.

Disabling the SMTP server on a system that doesn't function as a mail server may seem like a good idea, but many Linux systems rely upon this functionality to deliver important system status reports to the system administrator. Because of this, it's generally best to ensure that the mail server is configured in a secure way, which it normally is by default, and leave it running.

Remote Login Servers

A remote login server allows a user to log into the computer from a remote location. The most ubiquitous remote login protocol is Telnet, which is handled by a server called `telnetd` or `in.telnetd` in Linux. This server is open source and comes with all Linux distributions, although it's not always active by default.

Unfortunately, Telnet is an insecure protocol. Data passing between the Telnet client and server can be intercepted at points in-between the two, leading to compromised data. For this reason, it's best to disable the Telnet server on any Linux system and instead use a more secure protocol. Secure Telnet variants are available, but an alternative protocol, known as the *Secure Shell (SSH),* is more popular. SSH encrypts all data passing between two systems, making intercepted data useless. The most popular SSH implementation for Linux is the open source OpenSSH (`http://www.openssh.com`).

Telnet and SSH are basically text-based tools. SSH can be configured to tunnel X sessions through its connections, however. When so configured, you can run X programs remotely. You can do the same by setting various parameters from a Telnet login, as described in the section entitled "Using X Programs Remotely," in Chapter 5. More direct GUI remote login tools (the X Display Manager [XDM], GNOME Display Manager [GDM], and K

Display Manager [KDM]) are also available and come with all major distributions. Finally, the VNC package (`http://www.uk.research.att.com/vnc`) allows direct remote X logins, as well. With the exception of VNC, these tools all come with all major Linux distributions.

File Access Servers

A file access server lets users read, write, and otherwise manipulate files and directories from a remote location. The traditional remote access protocol is the File Transfer Protocol (FTP), which is still in common use. Many local networks use *file sharing protocols*, which allow programs on one computer to treat files on another system as if those files were local. Sun's Network Filesystem (NFS) is used for file sharing between Linux or Unix systems; the *Server Message Block (SMB)*, also known as the *Common Internet Filesystem (CIFS)*, is used to share files with DOS, Windows, and OS/2 systems; Novell's NetWare is another PC file sharing protocol; and Apple's AppleShare is the protocol used for Macintosh file sharing. Linux supports all of these protocols—NFS with standard kernel tools and various NFS servers; SMB/CIFS with the Samba package; NetWare with the `mars_nwe` and `lwared` packages; and AppleShare through Netatalk (`http://www.umich.edu/~rsug/netatalk`).

Most of these file sharing servers have printer sharing features, as well, so you can provide network access to printers connected to Linux. NFS is an exception to this rule, but NFS's lack of printer sharing is offset by the fact that Linux's standard printing tools include this feature themselves.

Because of its excellent support for so many different file sharing protocols, Linux makes an outstanding file and printer sharing platform in a cross-platform office. In an office that supports Windows, MacOS, OS/2, and Unix or Linux desktop systems, for instance, a single Linux computer can provide file and printer sharing services for all of these OSs, allowing users to move freely from one client platform to another or to collaborate with users of other platforms.

Miscellaneous Servers

The preceding list covers many of the most popular server types, but it's far from complete. Many servers fall into less-used categories or simply defy categorization. Examples include proxy servers, such as Squid (`http://www.squid-cache.org`), which improve network performance or security by buffering Internet access attempts; Dynamic Host Configuration Protocol

(DHCP) servers, which keep track of network configurations and help automate the configuration of DHCP client systems; Domain Name System (DNS) servers, such as BIND (aka `named`), which convert between numeric IP addresses and hostnames; and remote configuration tools like Red Hat's `linuxconf`, which allow you to change a system's configuration from another computer. Most Linux distributions ship with a wide range of such servers, some of which are active by default and some of which aren't.

Although not a server per se, the `ipchains` and `iptables` tools are extremely useful when configuring a system as a firewall, or in protecting an individual workstation with firewall-like rules. These programs can block access to your system based on IP addresses or network ports (numbers associated with specific servers or runs of client programs). `ipchains` fills this role with the 2.2.*x* kernel series, while `iptables` works with the newer 2.4.*x* kernels.

Useful Software on Any System

Whether a computer is to be used as a workstation or a server, certain classes of programs are extremely useful. These programs help users handle common user tasks and help administrators administer a system. Libraries are particularly important because they're the foundation upon which most other programs are built.

Text Editors

A *text editor*, as you might imagine, is a program used to edit text. Most system administrators need to be familiar with Vi, which is a small and ubiquitous Unix and Linux text editor. (Chapter 7 includes an overview of Vi operation.) If you need to do emergency maintenance, there's a good chance your emergency tools will include Vi as the text editor, or a close relative such as Vi Improved (VIM). `jed` and `pico` are a couple of other small text editors. These tools are designed to be similar to the popular Emacs program, which is an extremely large and flexible text editor.

Vi, `jed`, `pico`, and Emacs are all text-based programs, although some of them have at least some X extensions. In particular, XEmacs (`http://www.xemacs.org`) is an X-enhanced version of Emacs. Other text editors, such as Nedit (`http://www.nedit.org`), gEdit (part of GNOME), and KEdit (part of KDE), are designed from the ground up as GUI text editors. Although you may prefer to use one of these in day-to-day operation, you *will* occasionally need to use a text-based editor, so you should become familiar with at least one of them, as well.

Programming Tools

A *compiler* is a tool for converting a program's source code (its human-readable form, written by a programmer) into binary form (the machine-readable form, which users run). All major Linux distributions ship with a wide array of compilers, the most important of these being the GNU C Compiler (GCC). The Linux kernel is written mostly in C, as are many Linux programs. Some installations require other programming languages. If your users will be doing programming, ask them what tools they'll need. Most programming languages are available with major Linux distributions, and the rest can be found in open source and, occasionally, commercial forms.

Some programming languages aren't compiled; they're interpreted. In an interpreted language, the computer translates from human-readable form to machine code on the fly. This reduces execution speed, but it can speed development since there's no need to explicitly compile the software. Many interpreted languages are known as *scripting languages*, because they're used to create simple programs known as scripts. Java, Python, and Perl are popular interpreted languages.

Many developers like to work with an integrated development environment (IDE). IDEs provide GUI front-ends to editors, compilers, linkers, debugging utilities, and other programming tools. Some software companies make money selling IDEs for Linux development, such as Metrowerks CodeWarrior (`http://www.metrowerks.com/desktop/linux`). Other IDEs are open source projects, such as Code Crusader (`http://www.newplanetsoftware.com/jcc`), and KDevelop (`http://www.kdevelop.org`).

It's generally unwise to leave programming tools on a server system. If the system is ever compromised by crackers (those who break into computer systems), the programming tools can be turned against you to compile the cracker's own utilities. Nonetheless, compilers are useful in administering servers. Typically, you'll compile software on a system that's configured much like the server, and then you'll transfer the compiled software to the server system.

Libraries

A *library* isn't a program per se; rather, it's a collection of software routines that may be used by programs. Placing commonly used code in libraries saves both disk space and RAM. All Linux systems rely upon a library known as the *C library (libc)* because it provides routines that are necessary

for any C program to run in Linux. (The version of libc shipped with major distributions in 2001 is known as *glibc*.) Any but the most trivial Linux system will use a number of additional libraries, as well. You must ensure that you install the appropriate libraries. If you fail to do so, your package system will probably tell you about the problem, expressed as a *failed dependency* (dependencies are described in more detail in Chapter 3).

Validating Software Requirements

Computer software is highly interdependent. Programs rely upon others, which in turn rely on still others. This cycle ultimately leads to the Linux kernel—the "heart" of a Linux system. Even the kernel relies on other software—namely, the Basic Input/Output System (BIOS), which the kernel needs to start up, as described in Chapter 3. This web of dependencies and requirements sometimes poses a problem because you may need to install a dozen new programs in order to install a single package you want to use.

If a program comes with your Linux distribution, that program will most likely work well with that distribution. In some cases, you may need to install additional packages. Most distributions use package management systems that support dependency checking, as described in Chapter 3, so you'll be told what files or packages you're missing when you try to install a new program.

For programs that don't ship with a distribution—and even for those that do—you can usually find a list of requirements on the program's Web site or in its documentation. This requirement list may include several components:

Supported OSs Most Linux software works on many Unix-like OSs. It's usually best to check that a package explicitly supports Linux. This is particularly true of binary-only packages, such as those that are common in the commercial world. A binary package for IRIX won't do you any good in Linux, for instance. Unix programs that come with source code can often be compiled without trouble on Linux, but the larger the program, the more likely you'll run into a snag if the author doesn't explicitly support Linux.

Supported distributions Some packages' documentation refers to specific Linux distributions. As a general rule, what works on one distribution can be made to work on another. Sometimes the conversion process is trivial, but sometimes you'll need to wade through a tangled mess of unfulfilled dependencies to get a program working on a distribution its author doesn't explicitly support.

CPU requirements Software that comes in source code form can usually be compiled on any type of CPU. Binary-only programs, though, usually work only on one CPU family, such as *x*86 or PowerPC. This problem afflicts many commercial packages. Even some programs that come with source code don't compile properly on all CPUs, although this problem is rare.

Library requirements The vast majority of programs rely upon specific libraries, such as libc and GTK+. Check the requirements list and try to determine if the libraries are installed in your system. If your distribution uses the RPM or Debian package system, you can usually check for a library of the specified name. Chapter 3 discusses software management, including RPM and Debian package utilities.

Development tools and libraries If you intend to compile a program yourself, pay attention to any development tools or libraries the package uses. For instance, if a program is written in C++, you'll need a C++ compiler. Also, many libraries have matching development libraries. These include additional files needed to compile programs that use the libraries, but that aren't needed merely to run such programs once compiled.

If your system seems to meet all the requirements specified by the program's author, try installing the package according to the provided instructions. If you have trouble, read any error messages you get when you try to install or run the program; these often contain clues. You may also want to check Chapter 3, "Software Management," for information on Linux packages, and Chapter 9, "Troubleshooting" (particularly the section entitled "Package Dependencies and Conflicts"), for package installation troubleshooting tips.

Understanding Software Licenses

Most computer software is copyrighted. This gives the copyright owner the legal right to restrict distribution of the software, and even to limit how it may be used. In the Linux world, *open source* licenses dominate the landscape. These give users unusually broad rights. Commercial software for Linux is also available, though, and some products fall somewhere in-between the two.

One of the problems with Linux software licensing is that the culture surrounding open source is often perceived as hostile towards commercial software and even commercial users of software. Although this perception is often overblown, it's important that you be aware of it and how you may and may not use software with varying licenses in Linux. Understanding the issues can help you to arrive at a decision regarding the correct software licenses for your environment, and it may help you overcome any misconceptions you may encounter among your co-workers.

Open Source Software Licenses

Open source software and its culture have evolved substantially over time. One early influence was the Free Software Foundation (FSF) and its GNU's Not Unix (GNU) project, which developed many critically important components that are now used in Linux, such as GCC. The FSF developed the *General Public License (GPL)*, which Linus Torvalds used for the Linux kernel. The FSF is a leader in what's known as the free software movement, which advocates software freedom—the ability of users to distribute and modify program source code.

By the mid-1990s, the free software community began to see the need for some changes. For one thing, some felt that the term "free software" was inappropriate because it tended to deter development by for-profit companies. In fact, the primary meaning of *free* in free software refers to the freedom to do what one likes with the code; it does not mean that it is a zero-cost distribution policy. In 1997, a collection of movers in the free software community (notably lacking Richard Stallman, the founder of the FSF) created a formal definition of what they termed open source software. This definition includes nine components, summarized here:

1. **Free redistribution—**The user must have the right to redistribute the software without paying royalties.
2. **Source availability—**The program must have source code, either as part of the main package or readily available via the Internet.
3. **Derived works—**The license must allow the user to modify the source code and distribute these modifications under the same terms as the original.
4. **Source code integrity—**An open source license may restrict distribution of modified code, but only if patch files (files used to modify, or

"patch," original source code) are permitted. This condition is essentially a weakening of point #3 in order to let the software's original author control a primary package, but still allow third-party modifications.

5. **No discrimination against persons or groups—**The license may not discriminate against any person or group.
6. **No discrimination against fields of endeavor—**The license may not restrict rights based on fields of endeavor (such as business use or genetic research).
7. **License distribution—**The license terms must apply automatically, without requiring signing a form or the like.
8. **License must not be specific to a product—**The license terms may not be contingent upon the program being part of another product.
9. **No-contamination—**The license must not "contaminate" other software by placing restrictions on other software distributed with the product.

Today, the Linux community as a whole has embraced the open source definition, although some influential individuals and groups still prefer to use other terms. In particular, the FSF continues to use the term "free software" for software distributed under the GPL and some other licenses. The FSF's GPL includes language stipulating that any changed version of a program must be distributed under the GPL. Such a requirement is certainly allowed by the open source definition, but it's not required by this definition. In fact, some open source licenses allow a person to modify the source code and distribute it under another license. (Note that the open source requirement #3 allows, but does not require, modifications to be distributed under the original license.)

Open source software, and particularly software distributed under the GPL, is frequently referred to as falling under *copyleft*. This play on the word "copyright" suggests a use of copyright laws to achieve goals that are in many ways the opposite of copyright—to ensure the free availability of software, rather than to limit the right to copy.

Every open source license has its own unique characteristics. These are mostly of interest to developers who might want to contribute to a software project, but on occasion they may be important to a system administrator. The major open source licenses include the following:

GNU GPL and LGPL The GNU GPL is the license used by the Linux kernel. As noted above, it contains language that requires modifications to be made available under the GNU GPL. This ensures that there will never be a proprietary version of the Linux kernel, which may be a good or bad thing, depending upon your point of view. A variant on this license is the Lesser GPL (LGPL), formerly known as the Library GPL. This is intended to be applied to libraries. The LGPL explicitly allows software that uses LGPLed code to do so even if the software does not follow the LGPL or GPL. This loophole is very important for libraries; a library licensed with the GPL would require all programs that use the library to also use the GPL.

BSD The BSD license is used by the open source BSD OSs, and by various software components developed for them. Unlike the GPL, the BSD license allows modifications to be distributed under other licenses. The latest versions of this license are very similar to the MIT license.

MIT The Massachusetts Institute of Technology (MIT) was the original moving force behind the X Window System, and the MIT license (sometimes called the X11 license) continues to be used for XFree86—the implementation of X included with all major Linux distributions. The MIT license is unusually short.

Artistic The Artistic license was originally developed for the Perl programming language, but it has been used with other programs. It's filled with requirements and loopholes for those requirements. Most software that uses the Artistic license is shipped with the stipulation that this license is optional; the user may elect to follow the terms of some other license (usually the GPL) instead.

The Qt Public license Trolltech (`http://www.trolltech.com`) developed a cross-platform GUI library, Qt, that was used as the core of KDE, among other programs. Qt was originally licensed in a manner that did not qualify it as open source, although it was freely available in Linux. Trolltech has since modified its license so that it does qualify as open source, although some free software purists dislike it because it retains more rights for the owner than do most open source licenses.

NPL and MPL The Netscape Public License (NPL) and Mozilla Public License (MPL) were developed by Netscape when they brought their Netscape Web browser into the open source field. Like the Qt Public license, the NPL reserves some rights for the copyright holder, but the MPL is more open.

There are additional licenses that meet the open source requirements. You can find a complete list, and additional discussion of just what an open source license is, on the Open Source Initiative Web site, `http://www.opensource.org`.

The details of the various open source licenses are probably not terribly important to most system administrators. You may use and redistribute any open source program as you like. If you modify a program, though, you should be aware of redistribution requirements, particularly if you want to merge two or more programs or distribute a program under a modified license. You should also be aware that some Linux distributions (particularly those that ship in official boxes) may include software that doesn't qualify as open source. Some of this is commercial software, and some of it falls into some variant category.

Open source software is *not* the same as public domain software, although the two are similar in some ways. "Miscellaneous Software Licenses" touches upon public domain software later in this chapter.

Commercial Software Licenses

Commercial software isn't as common in Linux as it is in Windows or commercial Unix systems, but it still exists. Commercial programs for Linux are frequently large productivity applications, such as WordPerfect Office; or major servers, such as Roxen. You're less likely to find small utilities sold under commercial licenses.

Although the Linux kernel is distributed under the GPL, there are a few commercial kernel module packages. In particular, the Open Sound System (OSS; `http://www.4front-tech.com`) package is a commercial set of sound drivers for Linux. OSS can distribute commercial kernel modifications because they're separate modules that aren't compiled into the kernel proper.

Commercial software is not as clearly defined as open source software. Some people consider anything that's not open source to be commercial, but it's not as simple as that, as illustrated shortly, in the section "Miscellaneous Software Licenses." As a general rule, though, commercial software has several characteristics not shared with open source software:

Closed source The source code to commercial software tends to be unavailable, or is available only with serious restrictions, such as the recipient signing a non-disclosure agreement (NDA), which prevents its redistribution.

Distribution limitations Often, the user cannot legally redistribute commercial software. This limitation sometimes doesn't apply, though. For instance, many commercial software packages today may be downloaded from the Internet and redistributed, but they don't work fully unless you enter a license key. (In this case, you wouldn't be allowed to distribute your license key.)

Payment You must usually pay to use commercial software. To be sure, you may also pay to obtain open source software, but you can usually find open source software for little or no cost, particularly if you don't count media or Internet charges. The developer of a commercial product usually expects a monetary return, though. Sometimes a stripped-down version of commercial software is given away.

If all these restrictions apply, you can be pretty secure in applying the term "commercial software" to a package. If some of these conditions don't apply, it may be a gray area, but commercial packages often relax one or even all of these restrictions. For instance, Corel makes a stripped-down version of its WordPerfect 8 available for download from the Internet for free, but most people still consider it to be commercial software.

When you use commercial software in Linux, you should be sure that you comply with any license restrictions. These may include a limit on the number of total or simultaneous users of the software, an expiration date for a license, or other factors. If you use much commercial software, tracking these limitations can be a headache.

Some people use the term "closed source" instead of "commercial software." This usage has the merit of being more precise—it focuses on the unavailability of source code, de-emphasizing the redistribution rights and payment issues. Something that's not open source is not necessarily closed source, though, as described in a moment.

Miscellaneous Software Licenses

Some software falls in a limbo-land somewhere between open source and fully commercial software. There are also licenses (or a lack thereof) that simply aren't either open source or commercial. A few specific examples include the following:

Public domain The term *public domain* refers to a work for which the author has eliminated the copyright, or for which the copyright has lapsed. Public domain software is distributed with *no* licensing or copyright restrictions. As such, it's neither open source nor commercial, although somebody who makes modifications could copyright those changes in either way.

A lack of a copyright notice does not necessarily mean that a product is in the public domain. If you can't find a copyright notice or statement that a package is public domain, contact the author to learn its status. The author may have neglected to explicitly address the copyright issue, or the copyright notice may have been lost somewhere between the author and you.

Not-quite-open-source Some software is distributed under a license that's fairly free but not quite fully open source. For instance, the qmail mail server uses a modified version of the GPL that forbids redistribution of modified binary versions of the product. This clause violates the open source definition, but the software is far from commercial in nature.

Shareware The term *shareware* is applied to software that's freely redistributable but for which the author requests a payment. Sometimes this software is uncrippled in its freely available form, but other times it requires a license key to unlock some of its features. The line between shareware and commercial products has blurred as the Internet has allowed major software houses to distribute software electronically. There have also been cases of people distributing software under an open source license while simultaneously requesting (but not requiring) payment.

Demoware Commercial software houses sometimes distribute crippled versions of their products for free. This software is often called *demoware* or *crippleware*. It's generally considered fairly commercial in nature. Sometimes demoware can be converted into a full product by paying for it and entering a product key, making the difference between it and shareware foggy.

The distinctions between these licenses can become quite blurry. In all cases, though, what's important is not so much whether you call the software open source, commercial, or something else; it's what the license does and does not allow you to do. If you're in doubt, contact the author of the software.

One final term that deserves consideration is *freeware*. This word generally applies to any software that's distributed free of charge, whether it's open source or not. This is not the same as free software, the term favored by the FSF for GPLed and some other forms of open source software. The free version of WordPerfect 8 and StarOffice are two examples of freeware that are not open source.

Using Licensed Software in Linux

Some people believe that Linux's open source nature means that the OS may only run other open source software. This isn't true. Many commercial applications are available for Linux. Most libraries use the LGPL or other open source licenses that allow developers to write software that uses the library without becoming encumbered by the open source license themselves. If this weren't the case, programs like WordPerfect and Roxen would not be available in Linux.

As with any environment, though, you should check the licensing terms of the software you intend to run. Software licenses occasionally include peculiar terms, and in principle, a program could include a requirement that the software not be run from Linux, or something else that would prevent you from legally running the software. I've seen licenses that restrict the use of the software geographically or in certain professions, for instance (such practices spurred the open source requirements #5 and #6).

Linux Distributions' Licenses

One final concern when discussing software licenses is the license for Linux as a whole. When you download a CD-ROM image file or buy a Linux package, the software you obtain uses many different licenses—the GPL, the BSD license, the MIT license, and so on. Most of these licenses are open source, but some aren't. Many distributions ship with a few shareware or not-quite-open-source packages, such as the shareware XV. Retail packages sometimes include outright commercial software. For this reason, you shouldn't

copy a retail Linux package's CD-ROM. (An image file downloaded from the Internet is probably safe to copy, but check any accompanying file called `COPYING` or `COPYRIGHT` to be sure.)

Linux distributions include installation programs, configuration programs, and the like. These tools are usually all that a distribution packager can lay claim to, in terms of copyright. Most distribution maintainers have made their installation and configuration routines available under the GPL or some other open source license, but this isn't always the case. For instance, SuSE's GUI configuration tools, YaST and YaST2, are not open source. Such details can turn what might seem like an open source OS into something that's not quite fully open source. Debian maintains a policy of using only open source software in its main package set, although it lets freely redistributable but non-open source programs into its "non-free" package set.

Because a complete Linux distribution is composed of components using many different licenses, it's not very useful to speak of a single copyright or license applying to the entire OS. Instead, you should think of a Linux distribution as being a collection of different products that comes with a unifying installation utility. The vast majority of all the programs use one open source license or another, though.

Locating Linux Software

Modern Linux distributions are fairly complete entities as delivered. Most fill 1 to 6 650MB CD-ROMs—and most of this content is stored in a compressed form, so the result is a system that can easily fill 1–8GB when completely installed and uncompressed. In practice, though, a Linux installation can be much sparer than this; you're unlikely to install every package that comes with your distribution, after all. Some of this space is also consumed by source code, which you probably don't need.

On the other hand, it's entirely possible that your installed Linux system will be missing a few programs that you want. This is particularly likely if you use one of the smaller distributions or want to run some commercial software products. When this happens, it's necessary that you know where to go to find what you want.

Locating Open Source Software

There are several ways you can obtain open source software:

The Linux CD-ROM Linux installation media, as just noted, typically include a wide array of software. If you know the name of the package you want, you can search for it on the CD-ROM. For instance, to find the Nedit editor on a CD-ROM mounted at `/mnt/cdrom`, you might type `find /mnt/cdrom -name "nedit*"`. Such searches work best when the filename is something obvious, given the package name.

File archive collections Companies that distribute inexpensive Linux CD-ROMs also often distribute CD-ROMs that contain the archives of popular open source FTP sites. These can be a good way to obtain extra software if you don't have a fast Internet connection.

File archive sites Many Web and FTP sites host collections of open source software. Notable examples include `http://sourceforge.net`, `http://www.ibiblio.org/pub/Linux`, and `ftp://sunsite.unc.edu`.

Package maintainer's site Most open source projects have homepages, often named after the application itself (usually in the `.org` domain). If you can't find such a site by using the program's name, try doing a Web search.

If you don't know the name of a package but do know the type of software you want to find (such as a Pascal compiler or alternatives to the Apache Web server), you can try searching at `http://sourceforge.net` or `http://www.linux.org`. Both sites include categorized lists of open source or Linux programs. The GNU Project Web site, `http://www.gnu.org`, includes a listing of the FSF's software and some related packages. A Web search and a search on Google Groups (`http://groups.google.com`) are also well worth trying.

Locating Commercial Software

Commercial software for Linux can be obtained in ways that often parallel those for obtaining open source software. Specifically, consider the following:

Linux packages Retail Linux packages frequently include demos of commercial products. Many ship with one or two fully functional commercial programs, as well.

Linux retailers You can obtain many commercial Linux programs from Linux retailers, such as Linux Mall (`http://www.linuxmall.com`), Linux System Labs (`http://www.lsl.com`), and CheapBytes (`http://www.cheapbytes.com`). In addition to these Web-based retailers, Linux software is increasingly finding its way onto computer, office supply, and even department store shelves.

Software publishers The software publishers themselves often sell their packages directly. In fact, some offer downloadable versions; you can download the software and then buy a license key from the publisher's Web site.

If you don't know the name of the software you want, searching on Linux Web sites or doing a Web search, as with open source software, will often turn up useful information. Linux magazines, such as *Linux Journal* and *Linux Magazine*, often carry reviews of commercial software products. These publications frequently have monthly themes in which some topic is covered in depth, often including multiple or head-to-head product reviews.

Summary

Before installing Linux, you should take some time to plan the implementation. Although Linux works with a wide variety of hardware, you should consider this detail carefully, both to get a system with the features you need within your budget and to be sure that you don't have any components that are unsupported in Linux. Checking the hardware before you install Linux can also save you a great deal of aggravation, should some component be installed incorrectly or conflict with another device.

Planning your software configuration is also important. This begins with planning disk partitions to suit the needs of the system. There are several different Linux distributions available, and you may even want to consider non-Linux OSs for some purposes. Occasionally, selecting software is dependent upon software licenses. Most Linux packages use an open source license, but some programs use commercial and other types of licenses. Depending upon the role of the computer, you'll need to install different sets of software when it comes time to install Linux, as described in the next chapter.

Exam Essentials

Describe the difference between a workstation and a server. Individuals use workstations for productivity tasks; servers exchange data with other computers over a network.

Suggest ways to stretch a limited budget when buying or building a Linux computer. Upgrade an existing computer rather than buy a new one; incorporate existing components into an otherwise new computer; prioritize the system's hardware needs, and eliminate or use inexpensive hardware for low-priority functions.

Describe how CPU speed, available RAM, and hard disk characteristics influence performance. Faster CPUs result in faster computations, and thus faster speed in computationally intensive tasks, while plentiful RAM gives the computer room to perform computations on large data sets. Hard disks vary in capacity and speed, which affect your ability to store lots of data and your ability to rapidly access it.

Describe Linux's partitioning needs. Linux requires a single root partition, and may require a separate swap partition. Additional partitions, corresponding to directories such as `/boot`, `/home`, and `/var`, are desirable on some systems, but aren't usually required.

Summarize the concept of a Linux distribution. A distribution is a collection of software developed by diverse individuals and groups, bound by an installation routine. Linux distributions can differ in many details, but they all share the same heritage and the ability to run the same programs.

Determine how a computer will be used. Workstations serve as productivity tools for individuals, whereas servers respond to data transfer requests from many clients. Which role a computer will fill determines the types of programs you'll install on it.

Describe the most important characteristics of open source software licenses. Open source licenses ensure the availability of source code and your ability to change and redistribute that source code and the resulting binary programs.

Describe the most important characteristics of commercial software licenses. Commercial licenses usually don't allow users to distribute the software or see the source code. Commercial software copyright holders usually expect payment for their software.

Commands in This Chapter

In other chapters, you will find a list of commands that were used in the chapter here. You should make sure you are familiar with these commands and how to use them.

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

Basic Input/Output System (BIOS)
Berkeley Standard Distribution (BSD)
binary
bit
bus
byte
C library (libc)
cache memory
central processing unit (CPU)
chipset
Common Internet Filesystem (CIFS)
compiler
Complementary Metal Oxide Semiconductor (CMOS) setup utility
console
copyleft
crippleware
demoware
desktop computer
desktop environment
direct memory access (DMA)
distribution
dual inline memory module (DIMM)
dynamic RAM (DRAM)
Enhanced Integrated Device Electronics (EIDE)
Ethernet
extended partition
external transfer rate
failed dependency
file sharing protocol
filesystem
freeware
General Public License (GPL)
glibc
hot swapping

Hypertext Transfer Protocol (HTTP)

Industry Standard Architecture (ISA)

input/output (I/O)

internal transfer rate

interrupt request (IRQ)

journaling filesystem

library

logical partition

main memory

master

master boot record (MBR)

modem

motherboard

mount point

open source

partition

partition table

Peripheral Component Interconnect (PCI)

primary partition

proprietary

public domain

RAMbus dynamic RAM (RDRAM)

RDRAM inline memory module (RIMM)

ribbon cable

root partition

scripting language

second extended filesystem (ext2 or ext2fs)

Secure Shell (SSH)

server

Server Message Block (SMB)

server program

shareware

single inline memory module (SIMM)

slave

Small Computer System Interface (SCSI)

software modem

text editor

Universal Serial Bus (USB)

window manager

workstation

Review Questions

1. Which of the following are typical workstation tasks? (Choose all that apply.)
   - **A.** Word processing
   - **B.** Routing between networks
   - **C.** Running a Web site
   - **D.** Running scientific simulations

2. A computer is to be used to capture 640 × 480 images of a room every 10 minutes and then store them for a day on hard disk. Which of the following components might you research before building such a computer?
   - **A.** A 21-inch monitor for viewing the images
   - **B.** A high-end SCSI disk to store the images quickly
   - **C.** A 3D graphics card to render the image of the room
   - **D.** USB support for a USB-interfaced camera

3. You're designing a computer as a workstation to be used primarily for word processing. Which of the following cost-saving measures is *least* appropriate for this system?
   - **A.** Buying a keyboard that costs $10 rather than one that costs $50
   - **B.** Buying a 40GB hard disk rather than an 80GB model
   - **C.** Buying a CD-ROM drive rather than a DVD-ROM drive
   - **D.** Buying a 750MHz system rather than a 900MHz one

4. Linux runs on many different types of CPUs. Which of the following measures is most useful when comparing the speed of CPUs from different families?
   - **A.** The BogoMIPS measures reported by the kernel
   - **B.** The CPU speeds in MHz
   - **C.** The number of transistors in the CPUs
   - **D.** How quickly each CPU runs your programs

5. Which of the following is *not* an advantage of SCSI hard disks over EIDE hard disks?

   A. SCSI supports more devices per IRQ.

   B. SCSI hard disks are less expensive than their EIDE counterparts.

   C. SCSI allows multiple simultaneous transfers on a single chain.

   D. The highest-performance drives come in SCSI format.

6. As a general rule, which of the following is most important in order for a video card to be used in a Linux business workstation?

   A. The card should be supported by the commercial Accelerated-X and Metro-X servers.

   B. The card should have much more than 8MB of RAM for best speed.

   C. The card should be supported by XFree86.

   D. The card should be the most recent design to assure continued usefulness in the future.

7. When installing an EIDE hard disk, what feature might you have to set by changing a jumper setting on the disk?

   A. The drive's bus speed (33, 66, or 100 MBps)

   B. The drive's termination (on or off)

   C. The drive's master or slave status

   D. The drive's ID number (0–7 or 0–15)

8. Why might you want to check the motherboard BIOS settings on a computer before installing Linux?

   A. The BIOS lets you configure the partition to be booted by default.

   B. You can use the BIOS to disable built-in hardware you plan not to use in Linux.

   C. The motherboard BIOS lets you set the IDs of SCSI devices.

   D. You can set the screen resolution using the motherboard BIOS.

9. You want to attach an old 10MBps SCSI-2 scanner to a computer, but the only SCSI host adapter you have available is a 20MBps UltraSCSI device. The system has no other SCSI devices. Which of the following is true?

   A. You can attach the scanner to the UltraSCSI host adapter; the two are compatible, although you may need an adapter cable.

   B. You must set an appropriate jumper on the UltraSCSI host adapter before it will communicate with the SCSI-2 scanner.

   C. You must buy a new SCSI-2 host adapter; SCSI devices aren't compatible across versions, so the UltraSCSI adapter won't work.

   D. You can attach the scanner to the UltraSCSI host adapter, but performance will be very poor because of the incompatible protocols.

10. A new Linux administrator plans to create a system with separate `/home`, `/usr/local`, and `/etc` partitions. Which of the following best describes this configuration?

    A. The system won't boot because `/etc` contains configuration files necessary to mount non-root partitions.

    B. The system will boot, but `/usr/local` won't be available because mounted partitions must be mounted directly off of their parent partition, not in a subdirectory.

    C. The system will boot only if the `/home` partition is on a separate physical disk from the `/usr/local` partition.

    D. The system will boot and operate correctly, provided each partition is large enough for its intended use.

11. Which of the following best summarizes the differences between DOS's `FDISK` and Linux's `fdisk`?

    A. Linux's `fdisk` is a simple clone of DOS's `FDISK`, but written to work from Linux rather than from DOS or Windows.

    B. The two are completely independent programs that accomplish similar goals, although Linux's `fdisk` is more flexible.

    C. DOS's `FDISK` uses GUI controls, whereas Linux's `fdisk` uses a command-line interface, but they have similar functionality.

    D. Despite their similar names, they're completely different tools—DOS's `FDISK` handles disk partitioning, whereas Linux's `fdisk` formats floppy disks.

12. Which of the following characteristics differ between Linux and commercial Unix systems? (Choose all that apply.)

    A. Ability to run open source software

    B. Cost

    C. History of kernel source code base

    D. Underlying principles of OS design

13. In what ways do Linux distributions differ from one another? (Choose all that apply.)

    A. Package management systems

    B. Kernel development history

    C. Installation routines

    D. Ability to run popular Unix servers

14. Which of the following packages are *most* likely to be needed on a computer that functions as an office file server?

    A. Samba and Netatalk

    B. Apache and StarOffice

    C. Gnumeric and Postfix

    D. XV and BIND

15. What type of software is it most important to *remove* from a publicly accessible server?

    A. Unnecessary kernel modules

    B. Unused firewall software

    C. Uncompiled source code

    D. Software development tools

16. Which of the following is *not* required in order for software to be certified as open source?

A. The license must not discriminate against people or groups of people.

B. The license must not require that the software be distributed as part of a specific product.

C. The license must not require that changes be distributed under the same license.

D. The program must come with source code, or the author must make it readily available on the Internet.

17. Which of the following is true of commercial software licenses in Linux? (Choose all that apply.)

A. They must conform to the terms of the LGPL.

B. They may restrict distribution, require payments, or have other terms common to commercial licenses in commercial OSs.

C. They are uncommon compared to open source licenses.

D. They necessarily prevent distribution of the commercial package with a Linux distribution.

18. How do you set IRQs on PCI boards?

A. You don't; PCI boards don't use IRQs.

B. You don't; they're set automatically by the BIOS or kernel.

C. By adjusting jumpers on the board.

D. By editing the `/etc/isapnp.conf` file and running `isapnp`.

19. How can you expect your Linux distribution to arrive?

A. With enough software that some systems don't need additional packages.

B. It will invariably require additional software package installation.

C. Generally, it will consist of at least 50 percent commercial software.

D. It cannot be obtained from the same sources that make commercial software available.

20. Possible sources of both commercial and open source Linux software include which of the following? (Choose all that apply.)

A. The program author's Web site

B. Linux retail boxes

C. Internet retailers

D. Brick-and-mortar retailers

Answers to Review Questions

1. A, D. Workstations are used by individuals to perform productivity tasks, such as word processing, drafting, scientific simulations, and so on. Routing is a task that's performed by a router—typically a dedicated-appliance task. Web sites are run on servers.

2. D. Many digital cameras use USB interfaces, so Linux's support for USB, and for specific USB cameras, may be important for this application. (Some cameras use parallel-port or specialized PCI card interfaces, as well.) A 21-inch monitor is overkill for displaying 640× 480 images, and a 3D graphics card isn't required, either. Likewise, a 10-minute pause between captures is slow enough that a high-end hard disk (SCSI or EIDE) isn't necessary for speed reasons, although a large hard disk may be required if the images are to be retained for any length of time.

3. A. As a word processing workstation, this system's keyboard quality is important. A 40GB hard disk and a 750MHz CPU are almost certainly more than adequate for this application, as is a CD-ROM drive.

4. D. The ultimate measure of a CPU's speed is how quickly it runs *your* programs, so the best measure of CPU performance is the CPU's performance when running those programs. The BogoMIPS measure is almost meaningless; it's used to calibrate some internal kernel timing loops. CPU speed in MHz is also meaningless across CPU families, although it is useful *within* a family. Likewise, the number of transistors in a CPU is unimportant per se, although more sophisticated CPUs are often faster.

5. B. SCSI hard disks usually cost more than EIDE drives of the same size, although the SCSI disks often perform better.

6. C. XFree86 comes with all full Linux distributions, so having XFree86 support is important to getting Linux working in GUI mode. Support in Accelerated-X and Metro-X can work around a lack of support in XFree86 or provide a few features not present in XFree86, but in most cases, XFree86 support is more important. More than 8MB RAM is important if you want to use a card's 3D features, but few Linux programs use these in 2001. The most recent designs are often incompatible with XFree86 because drivers have yet to be written.

7. C. EIDE drives can be configured for one of two positions on an EIDE chain, master or slave. (Modern drives often support auto-configuration through a "cable select" or similar option, and sometimes a single-drive configuration, but these are just different ways of setting the same feature.) Termination and ID number are characteristics of SCSI devices, not EIDE devices. The drive's bus speed adjusts automatically depending upon the maximum of the drive and the EIDE controller.

8. B. Motherboards with built-in RS-232 serial, parallel, EIDE, and other devices generally allow you to disable these devices from the BIOS setup utility. The BIOS does *not* control the boot partition, although it *does* control the boot device (floppy, CD-ROM, hard disk, and so on). SCSI host adapters have their own BIOSes, with setup utilities that are separate from those of the motherboard BIOS. (They're usually accessed separately even when the SCSI adapter is built into the motherboard.) You set the screen resolution using X configuration tools, not the BIOS.

9. A. SCSI devices are compatible from one version of the SCSI protocols to another, with a few exceptions such as differential SCSI devices. There are several types of SCSI connectors, so a simple adapter may be required. No jumper settings should be needed to make the UltraSCSI adapter communicate with the SCSI-2 scanner. Performance will be at SCSI-2 levels, just as if you were using a SCSI-2 host adapter.

10. A. The `/etc/fstab` file contains the mapping of partitions to mount points, so `/etc` must be an ordinary directory on the root partition, not on a separate partition. Options B and C describe restrictions that don't exist. Option D would be correct if `/etc` were not a separate partition.

11. B. Although they have similar names and purposes, Linux's `fdisk` is not modeled after DOS's `FDISK`. DOS's `FDISK` does *not* have GUI controls. Linux's `fdisk` does *not* format floppy disks.

12. B, C. Linux generally costs less than commercial Unix systems, and its source code (particularly the kernel) is not derived from the same base as that of commercial Unixes. Both Linux and commercial Unix systems can run most of the same open source software, though. As a clone of Unix, Linux uses the same underlying OS design principles.

13. A, C. Different Linux distributions use different package management systems and installation routines. Although they may ship with slightly different kernel versions, they use fundamentally the same kernel. Likewise, they may ship with different server collections, but can run the same set of servers.

14. A. Samba is a file server for SMB/CIFS (Windows networking), while Netatalk is a file server for AppleShare (MacOS networking). Apache is a Web server, and StarOffice is a workstation package. Gnumeric is a spreadsheet, and Postfix is a mail server. XV is a graphics package, and BIND is a name server. Any of these last six *might* be found on a file server computer, but none fills the file serving or any other necessary role, and so each is superfluous on a system that's strictly a file server.

15. D. System crackers can use compilers and other development tools to compile their own damaging software on your computer. Unnecessary kernel modules don't pose a threat. You may want to begin using unused firewall software, but removing it is unlikely to be necessary or helpful. Uncompiled source code may consume disk space, but it isn't a threat unless a compiler is available and the source code is for network penetration tools.

16. C. The open source definition specifies that users be able to distribute changes, but it doesn't require that the license allow distribution under the terms of another license. Options A, B, and D all paraphrase actual open source license term requirements.

17. B, C. Linux's licensing terms don't restrict the rights of commercial software vendors to apply their own licensing terms. Most Linux software is open source in nature. The LGPL allows commercial software to link to LGPLed software, but the LGPL does not impose its terms on commercial packages. Commercial packages can be and sometimes are distributed with Linux packages.

18. B. PCI was designed so that the BIOS or OS could set IRQs for these devices automatically, so it's not normally necessary to explicitly adjust these features. In the event of a conflict, you can sometimes change the algorithm the BIOS uses to assign IRQs, though.

19. A. Linux distributions come on 1–6 CD-ROMs, which include wide assortments of open source and occasionally commercial software—filling all the needs of some systems. Sometimes—but not invariably—additional software is required. All Linux distributions sold today include far less than 50 percent commercial software. Many retailers sell both Linux distributions and commercial Linux software. Some companies (such as Corel) even produce both.

20. A, B, C, D. Open source software may be distributed in any way that's technologically possible. Distribution of commercial software is dependent upon license terms, but as a whole, commercial software developers have embraced a wide array of distribution methods, including all the options listed here.

Installing Linux

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **2.1 Determine appropriate method of installation based on the environment (e.g., boot disk, CD-ROM, Network (HTTP, FTP, NFS, SMB)).**
- ✓ **2.2 Describe the different types of Linux installation interaction and determine which to use for a given situation (e.g., GUI, text, network).**
- ✓ **2.3 Select appropriate parameters for Linux installation (e.g., language, time zones, keyboard, mouse).**
- ✓ **2.4 Select packages based on the machine's "role" (e.g., Workstation, Server, Custom).**
- ✓ **2.5 Select appropriate options for partitions based on preinstallation choices (e.g., `FDISK`, third party partitioning software).**
- ✓ **2.6 Partition according to your preinstallation plan using fdisk (e.g., `/boot`, `/`, `/usr`, `/var/home`, `SWAP`).**
- ✓ **2.7 Configure file systems (e.g., (ext2) or (ext3) or REISER).**
- ✓ **2.8 Select appropriate networking configuration and protocols (e.g., modems, Ethernet, Token-Ring).**
- ✓ **2.9 Select appropriate security settings (e.g., Shadow password, root password, umask value, password limitations and password rules).**
- ✓ **2.10 Create users and passwords during installation.**
- ✓ **2.11 Install and configure XFree86 server.**
- ✓ **2.12 Select Video card support (e.g., chipset, memory, support resolution(s)).**

- ✓ **2.13 Select appropriate monitor manufacturer and settings (e.g., custom, vertical, horizontal, refresh).**
- ✓ **2.14 Select the appropriate window managers or desktop environment (e.g., KDE, GNOME).**
- ✓ **2.18 Read the Logfiles created during installation to verify the success of the installation.**
- ✓ **3.1 Reconfigure the Xwindow System with automated utilities (e.g., Xconfigurator, XF86Setup).**

Planning a Linux installation, as described in Chapter 1, "Planning the Implementation," is an important first step toward getting a Linux system up and running. Once you've done this, you can proceed to actually installing Linux, as described here. Many of the details of Linux installation differ from one distribution to another, and covering them all would take an entire book. Therefore, this chapter presents just one distribution's installation procedures: Linux Mandrake 8.0. Other distributions are similar, but differ in many details, such as the order in which you perform certain actions.

One of the trickiest aspects of Linux installation is getting the *X Window System*, or *X* for short, up and running. Many installers can do this correctly from the start, but sometimes you may need to modify your X configuration after the fact. Most distributions include one or more X configuration tools to help in the matter, or you can modify the X configuration file manually. Even if X itself is working, various extra tools are required to make X a *practical* working environment, and you may want to select alternative tools to the ones provided as your distribution's defaults.

Selecting an Installation Method

After you've decided on a distribution, the first choice you must make when installing Linux is what installation method you intend to use. There are two classes of options: the installation media and the method of interaction during installation. In both cases, some distributions offer more or different options than do others, so in truth, your preferences in these matters

may influence your distribution choice. For instance, Debian GNU/Linux doesn't support GUI installations, so if you strongly desire this feature, you can't use Debian.

Media Options

Linux can be booted and installed from any of several different media—floppy disks, CD-ROMs, network connections, and so on. For both booting and installing files, different media offer different advantages and disadvantages.

Boot Method

Linux installer programs run within Linux itself. This means that in order to install Linux, you must be able to boot a small Linux system, which is provided by the distribution maintainer. This system is useful only for installing Linux and sometimes for doing emergency maintenance. It typically fits on one or two floppy disks, or can boot from a bootable CD-ROM.

As described in Chapter 1, modern BIOSes include options for the selection of a boot medium. Typical choices include the floppy disk, CD-ROM drive, EIDE hard disk, SCSI hard disk, and high-capacity removable-media drive (like a Zip or LS-120 disk). In addition, some network cards include BIOSes that allow a computer to boot from files stored on a server. In theory, any of these media can be used to boot a Linux installer. Additionally, some distributions provide a DOS or Windows program that can launch the installation from a working DOS or Windows system.

Although many boot methods are possible, the three most common are as follows:

Floppy Many boxed distributions come with one or more boot floppies. If you configure your BIOS to boot from floppy disks before any other working boot medium, you can insert the boot floppy and turn on the computer to start the installation process. Even if you download Linux or obtain it on a cut-rate CD-ROM without a boot floppy, you can create a boot floppy yourself from a file on the CD-ROM (often called `boot.img` or something similar), using a DOS program such as `RAWRITE`. Look for these files and instructions on how to use them on the installation CD-ROM. The floppy boot method may be necessary if you plan to install from a network server.

CD-ROM Modern Linux distributions almost always come on CD-ROMs or DVD-ROMs that are themselves bootable. On a computer that's configured to boot from CD-ROM before other bootable media, you can insert the CD-ROM in the drive, turn on the computer, and the boot program automatically starts up. If you download and burn a Linux CD-R image file, you don't need to take any special steps to make this CD-R bootable. Some older BIOSes don't support CD-ROM boots, in which case you should make boot floppies, as just described.

Existing OS bootstrap Some distributions come with a DOS, Windows, or MacOS program that shuts down that OS and boots up the Linux installer. These programs sometimes run automatically when you insert the Linux CD-ROM in the drive. Using them can be a good way to get started if you plan to install a dual-boot system, or if you plan to replace your current OS with Linux.

Ultimately, the boot method is unimportant, because the same installation programs run no matter what method you choose. Pick the boot method that's most convenient for your hardware and the form of installation medium you've chosen.

Installation Media

The installation medium is the physical form of the source of the Linux files. Linux is very flexible in its installation media. The most common choices include those listed here:

CD-ROM or DVD-ROM If you buy Linux in a store or from an online retailer, chances are you'll get a CD-ROM. In fact, many distributions come on multiple CD-ROMs. Some companies, like SuSE, have started shipping a DVD-ROM with some of their packages. (DVD-ROMs can store much more data than can CD-ROMs, so a single DVD-ROM is equivalent to multiple CD-ROMs.) CD-ROM installations tend to be quick. Most distribution maintainers offer CD-ROM image files that you can burn to CD-Rs yourself. To find CD-R image files, check `http://www.linuxiso.org`, `http://delaware.linux.tucows.com/distribution.html`, or `ftp://sunsite.unc.edu/pub/linux/distributions` or go to your chosen distribution's Web or FTP site.

Network If you have a fast network connection and don't want to be bothered with installation CD-ROMs, you can install many distributions via network connections. Download a boot floppy image, create a floppy disk from it, and boot the installer. Tell it you want to install via the network and point it to a public archive site for the distribution. This

approach can also be useful if you've got a CD-ROM and a network but your target system doesn't have a CD-ROM drive. You can insert your CD-ROM into one computer on your network, configure that system to share the disc, and use network installation tools to read the files from that disc over the network. The drawback to network installations is that they tend to be slower than installs from CD-ROMs. They require more information from the user, and so they can be more difficult for a new user to get working. They can also fail midway if a network connection goes down or a server stops responding. Network installations may use any of several protocols to transfer files, including FTP, HTTP (Web), SMB (Windows file sharing), and NFS (Unix/Linux file sharing). Precisely which protocols are supported varies from one distribution to another.

Hard disk It's possible to put the Linux files on a DOS or Windows partition and install Linux in another partition using those files. This approach used to be somewhat common among hobbyists who would download the files, but who didn't have a CD-R burner. It's less common today but still occasionally useful. You might use it if your CD-ROM drive doesn't seem to work in Linux, for instance; you could copy the files from the CD-ROM to the hard disk and then install from there. Because Linux treats high-capacity removable-media drives as if they were hard disks, you could also store installation files on something like a Jaz or Orb drive, which might be convenient for installing Linux on multiple systems in some environments.

Floppy disks Early Linux distributions came as floppy disk sets. With today's major distributions commonly exceeding 1GB compressed, floppy disks aren't a very appealing distribution medium. A few specialized distributions, however, are still quite small. The Linux Router Project (`http://www.linuxrouter.org`), for instance, produces a single-floppy Linux distribution intended to turn an old computer into a network router.

Monolithic files It's possible to distribute an entire Linux system as a single file. One example along these lines is an image file of a demo Linux CD-ROM, which can boot directly from the CD-ROM drive and run Linux without installing it on the computer. Another example is the ZipSlack distribution, which is a stripped-down version of Slackware (`http://www.slackware.com`). This distribution uses extensions to the DOS or Windows File Allocation Table (FAT) filesystem so that you can store the distribution on an ordinary FAT partition or high-capacity removable-media drive, like a Zip or LS-120 drive. Once this is done, you can boot ZipSlack using a floppy disk.

Not all distributions support all of these installation options. All mainstream distributions support installation from CD-ROM, and most support at least one form of network installation. Beyond this, you should check the documentation for the distribution.

Even if a system lacks a CD-ROM drive, you can temporarily install a drive from another computer in order to install Linux. This is usually not the most efficient course of action if the system has a network connection, but it can be handy for installing Linux in an isolated system.

Real World Scenario

How to Obtain Linux

There are several common methods of Linux distribution:

- In official packages from the distribution maintainer, which are typically shipped with multiple CD-ROMs, printed manuals, and some amount of customer support
- Packaged with third-party books that describe the distribution
- From cut-rate CD-ROM duplicators, who sell distributions for as little as $2
- Downloaded from the Internet, typically as CD-R image files for burning to CD-R discs or via network installations

As a general rule, you get less in the way of useful extras and support as you move down this list. Therefore, purchasing a commercial distribution is often the best choice for new Linux administrators. Also, by purchasing a commercial distribution you support companies that often help improve Linux—for instance, Red Hat contributes to kernel development, and SuSE contributes to audio and video driver development. The low cost of cut-rate CD-ROMs or (if you have fast Internet connections) downloading the distribution yourself can be appealing if you are on a budget. Ultimately, you'll have to settle on your own comfort level regarding support and your budget. Personally, I have used—and I continue to use—distributions obtained through all four of these methods.

Methods of Interaction during Installation

Most methods of Linux installation require you to make decisions during the process. You may need to tell the system how to partition your hard disk, what your network settings are, and so on. To handle such interactions, distribution maintainers have developed three methods of data entry: GUI-based, text-based, and scripted. The first two are most suitable for customized individual installations, while scripts are best used when you are configuring large numbers of nearly identical systems.

GUI Installations

As a general rule, Linux distributions are shifting towards GUI installer programs. These tools run the XFree86 GUI environment in a basic 640 × 480 (VGA) mode that works on most modern video cards. (Some installers can run at 800 × 600 or higher.) The system can then use familiar mouse-based point-and-click operations to obtain input from the user. Because the display is a bitmapped graphics image, it's also possible to display graphical representations of information such as partition sizes. These displays can be very useful because people often find it easier to interpret graphs than the numbers that are more often used by text-based utilities.

GUI installations are most popular on CD-based installations. XFree86 and its related libraries are fairly large, so implementing an X-based installation over a network or floppy-based connection is tedious at best. Also, GUI installers don't work on all systems because some have unusual video hardware that's incompatible with the GUI installer. This problem is particularly acute with laptop computers, whose LCD screens sometimes don't work with the video modes used by GUI installers. If you're faced with such a situation, you may need to use a text-based installer.

Text-Based Installations

A few distributions (most notably Debian and Slackware) don't provide GUI tools, so you *must* use a text-based installer if you want to install one of these distributions. In principle, a text-based installation works just like a GUI one. Specifically, you must enter the same types of information—Linux partition information, TCP/IP network configuration options, package selections, and so on. Text-based tools require you to select options using your keyboard, though, and they can't display graphics that are nearly as sophisticated as can a GUI installer. Some text-based programs can produce crude

progress bars and the like, though, and some use text-based menus in which you tab through options to select the one you want. A few even let you use the mouse to select options from textual menus.

Most Linux distributions offer a text-based installation option. Typically, an early screen gives you the choice of running a GUI or text-based install, or you can type a special command to switch into a text-based mode if the default is a GUI installer. Consult your distribution's documentation if you don't see an obvious way to start a text-based installer.

Scripted Installations

With an automatic scripted installation, you typically create a configuration file that includes the information you'd normally enter with the keyboard or mouse—partition sizes, networking options, packages to install, and so on. Early in the installation process, you tell the system to read the configuration file from a floppy disk or from the network. The system then proceeds with the installation without further intervention from you.

To create the configuration file, you must normally install Linux manually on one system. The installer gives you the option of saving the configuration file. When you install Linux on the next system, you use this file to create a system that's configured identically to the first.

Scripted installations work best when you need to install Linux on many identical or nearly identical computers. If the systems vary in important details like hard disk size or intended function (workstation, server, and so on), a scripted install won't give you the opportunity to change those details on the various systems, so you'll end up spending *more* time correcting matters after the installation than you'll save by using the scripting features. You can also save your configuration options so that you can quickly reinstall a distribution on a single computer, should the need arise.

If you have many nearly identical systems to install, invest time in getting the installation just right when you create a set of installation parameters. For instance, you might want to use the custom package selection option (described shortly) to fine-tune what packages are installed. You'll invest time in the initial installation, but you'll save time reconfiguring all the systems after they're installed.

Not all distributions include scripted installation options. Consult your distribution's documentation for details.

Starting the Installation

This chapter presents a sample installation of Mandrake 8.0 from a CD-ROM. This example shows how to set up a system using the Mandrake GUI installer. As noted earlier, the details of configuring other Linux systems will differ, but the general principles will be the same as those presented here.

If you're installing Linux on a system and you plan to dual-boot between Linux and some other OS, you should first run disk-partitioning software, such as FIPS or PartitionMagic, to create empty space on the hard disk for Linux. Most Linux distributions do not have the means to dynamically resize partitions during installation, so if the disk has no unpartitioned space or blank partitions, you'll find that you're unable to install Linux without deleting partitions that are in use.

To begin the installation, insert the CD-ROM in the drive and power on the computer. If it's set to boot from the CD-ROM drive, you'll hear it spin up and see a prompt that reads `boot:`. Press the Enter key at this prompt to start up the GUI installer.

After pressing the Enter key, you'll see a large number of messages scroll past on the screen. These are the kernel startup messages, and they reflect various kernel drivers starting up, checking for hardware, and reporting on their status. After these have finished, the screen will clear and you'll see the first of the option configuration screens, which lets you set your default language.

The order in which the installer collects information varies from one distribution to another. Therefore, you may find that your distribution asks for information in an order other than that presented here. More-or-less the same information must be entered for all distributions, though.

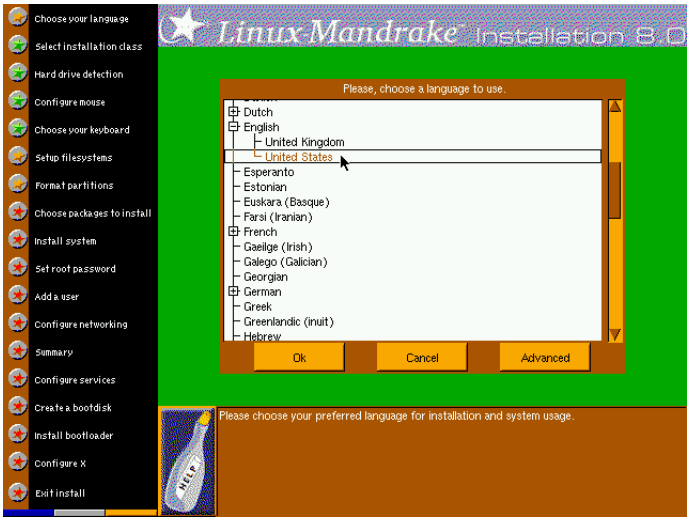
Selecting Basic Installation Parameters

The first stage of most Linux installations collects basic information on the system so that the installer can interact with you and locate the files to be installed. Many of these settings carry over into the installed system, as well, so be careful to enter correct information. You can change these settings after installing the OS, but it's simpler to get them right initially.

Language

The first question Mandrake's installer asks is what language it should use (see Figure 2.1). Mandrake's installer presents a summary of installation steps on the left, and the stars next to each option change color as you move through the list. The rest of the screen shows an option setting, with a help message below that. If you ever want to go back to a prior configuration step, click the star next to that step in the list on the left side of the screen.

FIGURE 2.1 Installation configuration screens typically present options in small groups.



In this example, I chose United States English (the default) as the language and clicked OK.

License Terms

A few distributions, including Mandrake, present a software license at some point during installation. Chapter 1 discusses software licenses, including licenses for Linux distributions as a whole. You should read the license terms to be sure there are no surprises, such as proprietary software included with the distribution that might limit what you can do with it.

Installation Class and Hardware Detection

After you accept the license terms, the system enters into the phase of the installation in which you prepare the computer to receive Linux. This involves choosing the general class of installation and detecting some of the hardware.

Installation Class

Linux Mandrake offers two major installation types: Install and Update. Each of these options has its own button, and for each, you can opt to do either a Recommended or an Expert installation. The latter gives you much greater control over what the system does during installation, but it assumes you know more about how Linux operates. As a general rule, the Expert installation offers more and better choices. For this reason, the rest of this chapter emphasizes the results of choosing this option, but you might want to choose Recommended for a streamlined installation if you're new to Linux.

Some distributions use terms like Workstation, Server, and Custom installations. The number and names of such options vary from one distribution to another.

Disk Detection

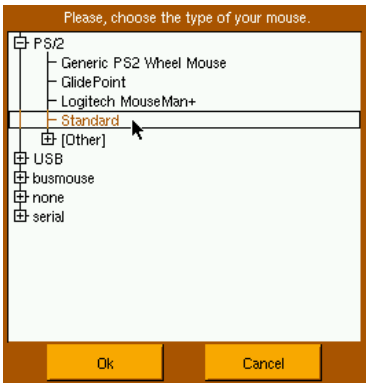
Mandrake attempts to auto-detect your hard disks. In some cases, it may miss SCSI devices, so if it doesn't detect a SCSI adapter, it asks if you have one. If you do, select Yes and click OK; otherwise select No and click OK.

Mandrake does a good job of auto-detecting EIDE disks, so you shouldn't be asked about your EIDE drives.

Mouse and Keyboard

Once the system has detected your hard disk, it asks for information on your mouse, as shown in Figure 2.2. The configuration groups mice into sections according to their hardware interfaces (PS/2, USB, and so on). Within each group, there are several options that relate to the protocols and mouse features, such as the number of buttons and whether or not the mouse has a wheel.

FIGURE 2.2 The installer shows the type of mouse it has auto-detected as the default.



Mandrake auto-detects the mouse, so it's probably set correctly to begin with. If you change the mouse selection, the installer gives you a chance to test it by showing a simple mouse outline with three buttons. Move the mouse around and click the buttons. You should see the mouse pointer move and the on-screen "buttons" should light up when you click the real mouse buttons. If this doesn't work, use the Tab key to highlight the Cancel button. This returns you to the mouse selection screen (Figure 2.2), where you can choose a different mouse. If you click OK, Mandrake will move on to keyboard configuration.

Linux supports many different keyboards, but Mandrake presents only two options by default: US Keyboard and US Keyboard (International). The first option works fine with most keyboards sold in the United States. If yours isn't one of these, click More to see an expanded list of keyboards for many nationalities. When you've selected your keyboard, click OK.

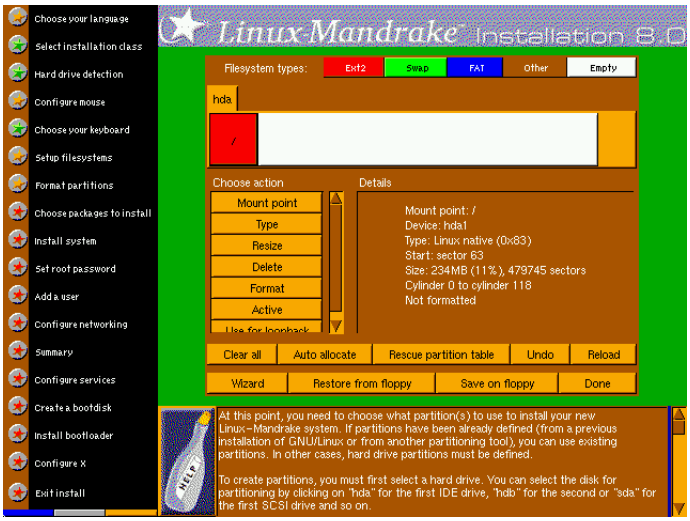
Configuring Disks

To install Linux, you must normally partition disks and create filesystems on those partitions. All Linux distributions include some way to accomplish both tasks before installing packages.

Disk Partitioning

Linux Mandrake uses a GUI disk-partitioning tool, shown in Figure 2.3. This tool presents a graphical representation of the partitions on the disk, and it allows you to adjust these partitions in various ways. If you want to completely destroy any existing partitions, click Clear All. You can click Auto Allocate or Wizard to have the system determine partition sizes automatically. These can be good options if you're unsure of how to proceed, but if you want more say in the matter, you can create partitions manually.

FIGURE 2.3 Linux Mandrake's GUI partitioning tool lets you point at free space or partitions and click options to create or modify partitions.

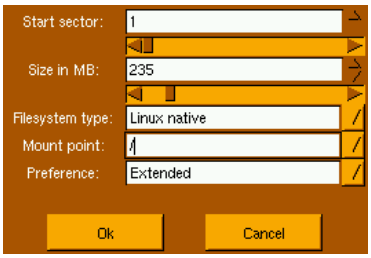


To create partitions for use by Linux, follow these steps:

1. Click in an area of free space (white in the partition listing). The Create option will become available in the Choose Action area of the screen.

2. Click Create. This produces a small dialog box, shown in Figure 2.4, in which you enter partition information.

FIGURE 2.4 You specify critical partition information in a single dialog box.



3. The Start Sector field defaults to the first available sector. You can increase this value if you want to create partitions out of order, but leave it alone if you create partitions from first to last in that order.
4. Type the size of the partition in megabytes in the Size in MB field. Alternatively, you can move the slider below this field to change it.
5. Select the type of filesystem in the Filesystem Type field. There are several options for Linux partitions: Linux Native (ext2fs), Linux Swap, ReiserFS, Linux RAID, and Linux Logical Volume Management (LVM). Linux Native is the safest choice, although ReiserFS might be desirable if you need a journaling filesystem, and the RAID and LVM options are useful if you understand what these options are and need their features. There are also options for various non-Linux partition types, such as DOS FAT-16 and BeOS. Linux won't actually create most of these non-Linux filesystem types, but the installer will mark the partition types appropriately if you choose one.
6. In the Mount Point field, select a mount point, as discussed in Chapter 1. You *must* create a root (/) filesystem, although it need not be the first one you define. You can skip this step for swap partitions; they don't have mount points.
7. In the Preference field, select either Primary or Extended. (Mandrake doesn't explicitly mention logical partitions; use the Extended choice for logical partitions, and Mandrake will create logical partitions inside an extended partition.)

8. Click OK to create the partition. You should see it appear in the partition list.
9. If you need to define additional partitions, repeat steps 1–8.

You can also edit existing partitions by clicking them and then clicking an option in the Choose Action area of the screen. Each action corresponds to one partition characteristic, like the mount point or filesystem type. Doing this is most helpful if you want to add Windows partitions or partitions for existing `/home` directories or the like to your system.

Some Linux distributions use `fdisk`, Linux's text-mode partitioning tool, for creating partitions. Some give the option of using `fdisk` or a GUI tool. Chapter 7, "Managing Partitions and Processes," describes the use of `fdisk`.

Most Linux distributions let you add partitions you've previously defined to your installation. This lets you create partitions for Linux in third-party utilities like PartitionMagic or reuse partitions you'd used in previous installations. A few, though, will only install to partitions they create, which means you must delete existing partitions and re-create them with the distribution's installation tools.

Creating Filesystems

When you're done defining partitions and setting their mount points, click Done to go on to the next screen. This produces the partition formatting screen. *Formatting* a partition, also known as *creating a filesystem*, is the process in which low-level data structures associated with a given filesystem (such as ext2fs or ReiserFS) are written to a partition. As such, creating a filesystem is an inherently destructive and dangerous process. If the partitions in question really *should* be formatted, there's nothing wrong with doing so, but if you select the wrong partition, you can wipe out a lot of data very quickly.

The Linux installation utilities offer a choice to create or not create filesystems for each of the partitions you defined earlier. If you're doing a fresh install of Linux on a Linux-only system, you'll almost certainly want to perform this format step, so be sure these options are selected. (You can omit this step if you created Linux filesystems using some other utility, such as PartitionMagic.) If you're upgrading a system or are installing Linux alongside another OS, you should be more cautious. Double- and triple-check that you format *only* the partitions you need to format before proceeding.

Low- and High-Level Formatting

The term "formatting" has dual meanings. It can refer to a low-level format, in which the physical sectors of a disk are redefined; or to a high-level format, in which the filesystem is re-created. Hard disks come from the factory with a suitable low-level format, and it's almost never necessary to redo this. Low-level hard disk formatting requires special utilities or, in the case of SCSI devices, BIOS options. Floppy disks, though, can be both low- and high-level formatted by users. In DOS and Windows, the `FORMAT` utility accomplishes both tasks on floppies, or a high-level format on hard disks. In Linux, the `fdformat` utility does a low-level floppy disk format, and `mkfs` (or subutilities associated with particular filesystems) does the high-level format on both floppies and hard disk partitions. Chapter 7 discusses creating filesystems, or you can read the man pages for these utilities after you've installed Linux by typing **`man fdformat`** or **`man mkfs`** at a command prompt.

Hard disk sectors occasionally fail. Such failed sectors are known as *bad blocks*, and Mandrake, like many other Linux distributions, gives you an option to check for them during the format process. You'll need to click the Advanced button to find this option, though. If you select this option, Linux will verify that every sector can hold data as it does the format. If any are found to be bad, the system will map them out of the list of available sectors, so as not to endanger your data. Performing a bad block check takes time, though—usually several minutes, and potentially over an hour on a large disk. Nonetheless, it's generally a good idea to perform a bad block check because it can save your data in the future.

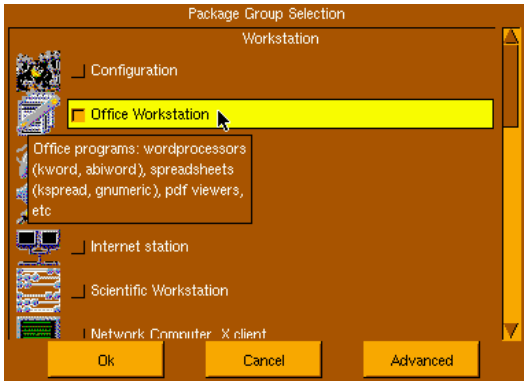
Bad blocks are usually mapped out by the hard disk itself. If a modern drive develops bad blocks that are detectable by Linux, chances are the disk won't last long. You'd do well to replace it.

Choosing Packages for Installation

After setting up the system's partitions, the Mandrake installer proceeds to a screen in which you enter package groups you want to install (see

Figure 2.5). Each option available on this screen represents a group of packages—several programs that together provide some set of functionality.

FIGURE 2.5 Each package group consists of multiple packages that provide related tools.



There are quite a few package groups available. As shown in Figure 2.5, most include explanation bubbles that appear when you move the mouse over the group. Select whichever of these package groups sounds useful, but be aware that the more you select, the more disk space they'll consume. Package groups tend to provide a broad range of functionality, so choosing just a few, particularly from large groups like KDE Workstation or GNOME Workstation, can consume hundreds of megabytes of disk space, if not more. Some distributions (but not Mandrake) provide an estimate of the required disk space somewhere on this screen. Mandrake does provide such an estimate after you move on, though.

If you think you might want *some* features from some of the groups you've selected, but not all of them, click Advanced and check the Individual Package Selection option. If you click this option and then click OK, Mandrake displays a screen in which you can select and deselect individual packages, as shown in Figure 2.6. You can browse through package categories using the list on the left of the screen. Selected packages appear with checks to the right of their names, and unselected packages have squares there. Click the name of the package to see a description of it, then click the box or check mark to select or deselect it, respectively. Some packages include dependencies that require installing additional packages. If you choose to install one of these with the dependency unmet, the system will inform you that it must install more packages.

FIGURE 2.6 The package selection screen lets you fine-tune your default package installation selections.

When you're done selecting packages, click Install to proceed with system installation. The installer displays a progress dialog box, showing the progress of installing both individual packages and the system as a whole. Depending upon your hardware, your installation medium, and the size of your installation, this process may take anywhere from a few minutes to a few hours.

Install-Time User Configuration

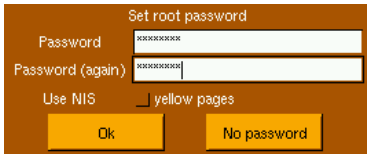
Most Linux distributions give you the opportunity of configuring at least one or two accounts during system installation, and often more. Mandrake is no exception to this rule. After you've installed the software for the system, you'll be able to configure the `root` password and create user accounts.

root Password Options

Every Linux system has a user with special privileges. This user is known as the superuser or the system administrator, and the associated account name is `root`. Because this account is so critical, you must have some way to log

into the account after you install Linux. Most distributions provide this functionality by allowing you to specify the `root` password during system installation. In Figure 2.7, you do this by typing the password twice, once in the Password field and a second time in the Password (Again) field. (Linux asks for a repeat password entry to protect against typos.) You won't see your password echoed to the screen; instead, you'll see a string of asterisks, as shown in Figure 2.7.

FIGURE 2.7 When you type a password during installation, you see asterisks echo to the screen as a security feature.



Most distributions, Mandrake included, enforce rules concerning password length and content. In Mandrake, the `root` password must be at least four characters long. Although not enforced during installation, Mandrake will object if you try to create a password after installation that's based on a word in its dictionary, such as common English words. Some distributions insist on passwords that have at least one or two nonalphabetic characters, such as numbers or punctuation marks. It's also a good idea to mix upper- and lowercase characters, although most distributions don't check for this. These measures are all designed to make passwords difficult to guess or deduce, even if a would-be intruder has a copy of Linux's password file, which stores the passwords in an encrypted form. Chapter 4, "Users and Security," covers password security in more detail.

At least one distribution, Corel Linux, does *not* set the `root` password at installation time. Instead, Corel Linux leaves the `root` account set with *no* password until the first `root` login. This approach leaves the computer vulnerable to intrusion until the `root` password is set. It's imperative that you do this *immediately* after installing the system, particularly if the computer is connected to a network. Clicking No Password in Mandrake also creates a `root` account with no password, which is an invitation to disaster.

If your network uses Network Information Services (NIS) to centralize password storage, check the Yellow Pages option. (NIS was formerly known as Yellow Pages, and Mandrake still uses that name during installation.)

User Account Creation

In addition to setting the `root` password, Mandrake's installer lets you configure one or more user accounts. You can do this after you click OK in the `root` password screen. Figure 2.8 illustrates this capability. To create an account during system installation, follow these steps:

1. Type the user's full name in the Real Name field.

FIGURE 2.8 Type information and click Accept User to add a user; click Done when you've defined all your users.

2. Type a username in the User Name field.
3. Type the password twice, once in the Password field and again in the Password (Again) field. As with the `root` password, you'll see asterisks rather than the actual password on the screen.
4. Click Accept User.
5. The Enter a User screen reappears, with an added note that it has already added the user you specified. You can repeat steps 1–4 for any additional users.

As with the root account, Corel Linux doesn't set the password of any user accounts you create during system installation. Instead, the system sets the password the first time somebody logs into the account. This design leaves the system vulnerable to intrusion until somebody logs into any of the user accounts that were created during system installation, so be sure to take care of this matter soon after installing the OS.

Pros and Cons of Defining Accounts during System Installation

As a general rule, it's helpful to create at least one user account during system installation. Even on a computer that's to be used by just one person, user accounts provide helpful security—when you are logged in as an ordinary user, you cannot do serious damage to a standard Linux installation. Therefore, creating a user account for the system's primary user during installation simplifies your post-installation work.

On the down side, account creation during system installation offers fewer options than are available after system installation. Chapter 4 covers account administration in more detail, so you should consult it for the details of what you can accomplish after installing a system. A few examples of things you can do after installing Linux but not typically before include setting up nonstandard home directory locations, adjusting account expiration policies, and defining group policies.

Miscellaneous Settings

After creating user accounts, Mandrake presents a series of miscellaneous configuration options. These options relate to network, time zone, and default account configuration.

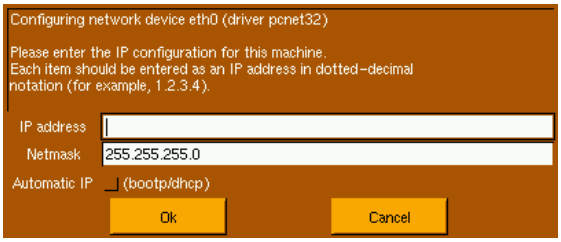
Install-Time Network Configuration

Mandrake lets you configure your network settings during installation. You can enable or disable networking by checking the Internet/Network Access

button. The next option lets you enable or disable auto-detection. Assuming you select this option, Mandrake presents a list of several possible connection methods, such as Normal Modem Connection and LAN Connection. This section describes the latter option.

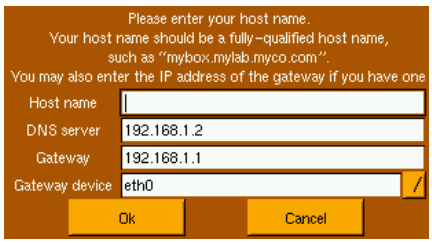
Mandrake informs you of the type of network card it detects. This detection is usually correct, so if you're unsure, click No when the installer asks if you have another card. You'll then be asked to enter critical IP address information, as shown in Figure 2.9. Enter your IP address and netmask in the appropriate fields. You'll have to obtain your address and other necessary information from your network administrator. If your network uses the Dynamic Host Configuration Protocol (DHCP), check the Automatic IP option and do *not* enter an IP address or netmask. When this option is set, your computer will obtain the necessary addresses and other information from your local network DHCP server. Not all networks include DHCP servers, though, so this option doesn't always work.

FIGURE 2.9 Linux lets you configure your network during system installation, even if you install from a CD-ROM drive.



After entering your IP address and netmask, Mandrake asks for additional information, as shown in Figure 2.10. The DNS Server and Gateway fields are particularly important. Again, you must obtain this information from your network administrator. You won't see this query if you selected the Automatic IP option, though.

FIGURE 2.10 The DNS server translates hostnames to IP addresses, and the gateway relays data between your network and the Internet.



Mandrake allows you to configure your system to use network *proxy servers*. These are servers that relay requests to the outside world, protecting a local network's systems in the process. If your network includes proxy servers, you can enter their addresses. If not, leave these fields blank.

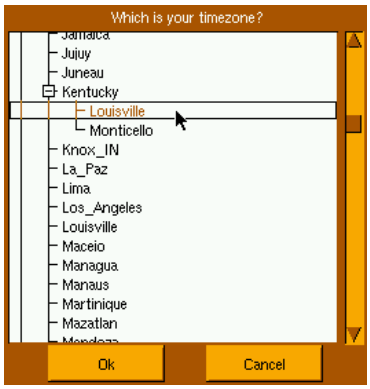
Mandrake Summary Configuration

After you have finished configuring networking, Mandrake presents a screen called Summary. Here, you can revisit your mouse configuration, set the system's time zone, or configure a printer.

Time Zone Configuration

To configure the time zone, click the time zone (which defaults to Europe/London) in the Summary screen. This produces an expandable list of cities, as shown in Figure 2.11. Locate a city in your time zone, click it, and click OK.

FIGURE 2.11 Time zone selection is important for maintaining appropriate time codes on files.



The list of cities used for setting the time zone is quite limited. For instance, New York is present, but few other major cities in the same time zone are there—so if you live in Boston, Philadelphia, Cincinnati, or elsewhere in the Eastern time zone of the United States, you'd indicate that fact by selecting New York.

Unix systems have traditionally stored their times in memory and in their hardware clocks in *Coordinated Universal Time (UTC)* format (aka *Greenwich Mean Time*)—the time in Greenwich, England, unadjusted for daylight savings. Unix systems traditionally do conversions to local time based on the time zone. Linux follows this tradition, but its task is complicated by the fact that *x*86 PCs have traditionally stored their times in hardware as local time and, when necessary, converted to UTC. Thus, the time stored in the computer's hardware clock (which Linux consults at boot time) is in local time. If Linux is to be the only OS on the computer, you can reduce the chance of problems with the time being skewed during daylight savings time conversions by responding Yes to the question, Is Your Clock Set to GMT. This obviates the need to change the hardware clock when daylight savings time rolls around. If the computer dual-boots between Linux and Windows, you should select No to this question.

Of course, no matter whether you store your time as UTC or local time, you or your software can convert to other formats, such as a 12-hour clock, or the peculiar Internet Time that breaks the day into 1000 "beats" rather than 24 hours. The PC BIOS requires a 24-hour time format, though.

Printer Configuration

If you want to configure a printer during installation, you may do so. Alternatively, you may put this task off until later. If you decide to configure a printer, click the No Printer item in the Summary screen. Mandrake will first ask if you want to use the Common Unix Printing System (CUPS) or the `lpr` printing system. CUPS is a more advanced system, but few applications in 2001 use its features, so in practice, both it and `lpr` work about equally well in most cases. The installer may then load some additional packages from CD-ROM. The installer will ask for several types of information, such as the following:

Connection type Options include a local printer, a remote Unix or Linux printer, or a remote Windows printer.

Connection details For local printers, you'll have to enter information on the port to which the printer is connected, such as the first printer port or the second RS-232 serial port. For remote printers, you'll need to specify the name of the remote server, the queue name, and possibly a username and password.

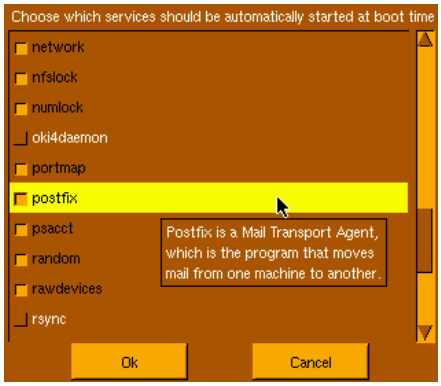
Printer type You must select the type of printer from a list. Mandrake provides a list of specific printer models, but some distributions provide shorter lists of printer types. (Most printers on the market today actually emulate just one of a dozen or two popular printer languages.)

You can test the printer configuration to see that it's working correctly. If it doesn't work, it's probably best to leave it until the system is fully installed rather than slow down your installation trying to fix it. After testing the printer configuration, Mandrake gives you the option to add more printer queue definitions or leave it at the one you've created.

Service Configuration

Linux systems invariably run several services. Many of these are network servers, like Web servers and mail servers. Others provide local services, like cron, which runs programs at scheduled times. Mandrake, like many distributions, gives you the chance to specify which services you want the system to start when you boot (Figure 2.12). Many of the services on this list have descriptions that pop up when you pause the mouse over the option. Select the services you want to run and click OK. If you're uncertain about these options, leave the defaults.

FIGURE 2.12 Choose which services you would like to have start automatically.



Boot Options

After you configure services, the Mandrake installer gives you the opportunity to create a boot disk. This floppy disk, if you create it, should boot the

computer even if something goes wrong with the hard disk boot process. Problems do sometimes arise with hard disk booting, particularly in dual-boot environments, so it's a good idea to create this boot floppy.

In most cases, you'll be able to boot Linux directly from your hard disk most of the time. To do this, Linux uses one of two programs: the Linux Loader (LILO) or the GRand Unified Bootloader (GRUB). Most Linux distributions use LILO, and a few (including Mandrake) offer GRUB as an option. Figure 2.13 shows Mandrake's boot loader install-time options.

FIGURE 2.13 You must provide a few key pieces of information about how you intend to boot the computer.

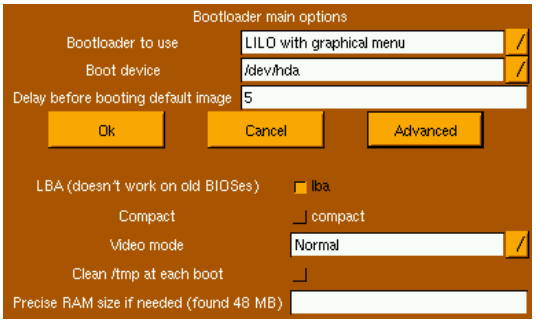


Figure 2.13 shows additional options available by clicking the Advanced button. You may or may not need to adjust these features. The complete set of options lets you adjust several factors:

- You can pick a boot loader (LILO or GRUB) from the Bootloader to Use option list. Mandrake lets you select between a graphical or text-based menu if you pick LILO. Both work, but the graphical menu is more visually appealing.
- The Boot Device is where LILO stores its boot-time code. On most systems, this will be `/dev/hda` (for EIDE hard disks) or `/dev/sda` (for SCSI hard drives). If you're configuring a dual-boot system and want to use something other than LILO or GRUB as the primary boot loader, you should put LILO or GRUB on the Linux boot partition (`/dev/hda1`, `/dev/sda5`, or some other numbered device—review your partition layout if necessary).
- LILO pauses for a period before booting a default OS. The Delay Before Booting Default Image entry sets this delay.

- Systems with hard drives larger than 8GB may need to have the LBA option set, as is the default. If you've got an old BIOS, though, this option should be disabled.
- The Compact option can speed up boot time slightly, especially when booting from floppy disks.
- A Linux boot loader can be programmed to configure the video card to run in any of several resolutions. The Normal setting for the Video Mode option leaves the default resolution set, but you can set the system to boot in various graphical or text modes.
- The `/tmp` directory holds temporary files. You can tell the system to clean this out when it boots by checking this option, or you can tell it to leave `/tmp` files intact on reboot.
- Linux can usually detect the amount of installed RAM correctly, but on some BIOSes this detection fails. The result is that Linux thinks the system has just 64MB (or sometimes just 16MB), when in fact it's got much more. If the amount of RAM shown in parentheses (48MB in Figure 2.13) is inaccurate, enter the correct value in the Precise RAM Size if Needed field.

When you click OK, the system presents a summary of the boot loader options. In Mandrake, this includes two Linux options (called `linux` and `failsafe`) and a floppy option (which redirects the boot process to the floppy disk). There may also be an option for DOS or Windows if your system dual-boots. Double-click Add to add more entries, or Done if you're finished.

Initial X Configuration

Linux uses the X Window System (or X for short) as its GUI environment. Prior to installing the system, Mandrake gives you the opportunity to configure X by using a series of X configuration screens. You're likely to be asked to enter the following information in these screens:

Video card model Your video card's model is important for determining the X drivers that Linux will use. Mandrake tries to auto-detect this value, so you may be able to use whatever the default is. If not, you should try to locate your video card from the list. Sometimes you'll need to select the video card's chipset (such as GeForce or S3 Trio), rather than the video card itself, so it helps to know what chipset the video card uses.

XFree86 version Mandrake 8.0 gives you the choice of using XFree86 version 3.3.6 or 4.0.2. The former is compatible with slightly more hardware, but the latter has additional features. Chances are that either will work, so pick one, and if you have problems, you can change it later (as described later in this chapter). Mandrake will install a package or two after you make this selection.

Your monitor model Different monitors have different maximum resolutions and refresh rates. If you're lucky, you'll be able to specify your monitor's exact model from a list. If not, you may be able to tell the system to use the Data Display Channel (DDC), which obtains information directly from the monitor, but this option works only on monitors and video cards built after the mid-1990s. If all else fails, you can enter your monitor's maximum horizontal and vertical refresh rates manually. These values should be documented in your monitor's manual.

Most distributions, including Mandrake, provide you with a way to test your X specifications. Typically, the system tries to launch an X server using the information you've entered. The computer displays a message asking if you can read the display. If you can, click Yes and the system will retain the settings you entered. If not, the system will return to the previous display after a brief period (usually 10 seconds or so), and you can try again.

On rare occasions, an attempt to test a video configuration can fail so badly that you must abort the installation. If Mandrake crashes at this point, it will probably boot, but it's conceivable the system will be damaged from the crash. Some distributions arrange installation tasks such that the system won't boot after a badly failed X video test. Therefore, these tests present the risk that you may need to go through the entire installation again. If you think you might have problems, it can make sense to configure the system to start in text mode and test and fine-tune X after installing the OS.

Some distributions provide you with the choice of having Linux start up in X mode or in text mode. X startups are often desirable for workstations, but most servers don't need to have X running at all times; in fact, doing so consumes RAM that might be better applied to the computer's server tasks. You can change this configuration detail after installing the system, as described in Chapter 6, "Managing Files and Services."

Checking Post-Installation Log Files

After you finish with X configuration, the Mandrake installer informs you that you may reboot the computer. If you're performing a template installation for subsequent scripted installs, though, you should click the Advanced button. This allows you to generate an automatic install floppy or save your package selections for a semi-automated installation. Once you've done that, or if you don't plan to create a scripted install floppy, click OK to reboot the computer.

Once the system has installed and rebooted, you should be greeted by a graphical or text-based login screen, depending upon the X options you selected. Try logging into the computer both as `root` and as any users you defined during the installation process. If you have problems, you may need to reconfigure accounts or passwords or perhaps change account-specific configuration files.

Most Linux distributions leave a log of installation options somewhere on the system, such as `/tmp/install.log`. It's generally a good idea to check this log after a system installation. You can do this by typing **`less /tmp/install.log`** at a command prompt. Most of the entries are routine—just the name of an installed package. Some, though, report configuration problems or errors, or even outright failures to install a package.

One common installation problem occurs when inadequate space is allocated for installation of all packages. If you can log onto the computer, type **`df`** to see a report of used and available disk space. If the `Available` column reads 0, or even if the `Use%` column shows a value above about 90% for any Linux partition, you may have allocated inadequate space for that partition. Immediately after installation, the easiest solution is often to go back and redo the installation with a better mix of partition sizes. Alternatively, you can use a package management tool to remove packages you don't need, as described in Chapter 3, "Software Management"; or you can use `resize2fs` or PartitionMagic to resize your partitions to more reasonable values.

Additional Possible Configuration Options

The preceding discussion outlines installation of a Linux Mandrake system on typical *x*86 hardware. There are, however, a few installation options that may be available on specific hardware or with other distributions:

Installation medium In the past, many distributions allowed you to select the installation medium (CD-ROM drive, network, hard disk, and so on) from a screen early in the installation process. The increasing size of the Linux kernel, however, has forced many distributions to provide separate boot disks for different installation media. If you boot from a CD-ROM drive, you'll install from CD-ROM. If you want to install from some other medium, you'll need to boot using an appropriate boot disk. Mandrake, for instance, provides a `network.img` file that, when written to floppy disk with the DOS `RAWRITE` program or Linux's `dd`, lets you install from the network.

Modem configuration Some distributions let you configure dial-up Point-to-Point Protocol (PPP) networking instead of or in addition to the networking configuration described earlier. Such configuration lets you specify a modem device (typically `/dev/ttyS0` or `/dev/ttyS1` for external RS-232 serial modems), a telephone number for your ISP, and a username and password.

Umask value As described in Chapter 4, the umask value determines the default permissions set on files created by users. This value is an octal (base 8) number that, when converted to binary, represents the bits that are *not* set in a permission string. Common umask values include `002`, `022`, and `027`. Consult Chapter 4 for more details.

Password options Some distributions give options for how to handle passwords. Specifically, two password encoding methods (MD5 and DES) have been used, and some distributions give the option of using either. Others use MD5 automatically. MD5 is more recent and more secure than DES. Also, some distributions give the option of whether or not to use *shadow passwords*. Traditionally, Unix and Linux systems have stored passwords and other account information in an encrypted form in a file called `/etc/passwd`. Various tools need access to this file for non-password information, so it needs to be readable to all users. Shadow

passwords allow the passwords to be separated into a file that's not readable by most users; this improves security.

Video Card RAM Although most distributions can auto-detect the amount of RAM you have installed in your video card, you might need this information in a few cases. It's usually printed early during the system boot process, but you may need a quick eye to notice it.

Post-Installation X Configuration

Once you've installed Linux, you may need to perform additional configurations to get it working at even a minimally acceptable level. The item that's most likely to cause problems is X configuration. You may find that you've installed Linux, but that X doesn't work correctly. You might also want to modify your X configuration to work in a way that's more to your liking, such as running in a different resolution. You'll also need to change your X configuration if you replace your video card with an incompatible model. For all of these cases, Linux provides X configuration tools, or you can manually edit the X configuration file. The first task you may need to undertake is selecting an X server.

Selecting an X Server

X is a network-enabled GUI system. It consists of an *X server*, which displays information on its local monitor and sends back user input from a keyboard and mouse; and an *X client*, which is a program that relies upon the X server for user interaction. Although these two programs frequently run on the same computer, they don't need to. Chapter 5, "Networking," includes additional information on using X over a network. The rest of this chapter assumes you'll be running X programs on the same system that runs the server, but you don't install X differently if you'll be running X programs remotely.

The X server includes the driver for your video card, as well as support for your mouse and keyboard. Therefore, it's important that you know something about your video card when you install and configure your X server.

Determining Your Video Card Chipset

In order to properly configure X for your system, you must know what video card chipset your system uses. Unfortunately, this information isn't always

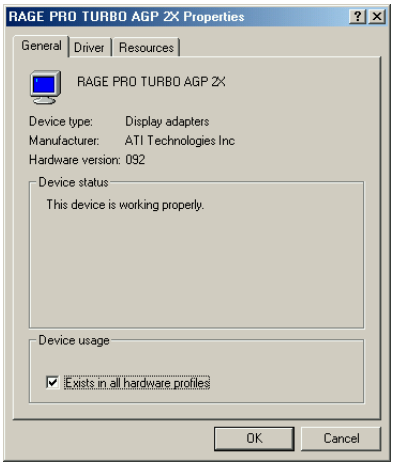
obvious from the video card's box or manual because many manufacturers use other companies' chipsets, and they don't always make the chipset manufacturer obvious. There are several ways to approach this problem, including the following:

Auto-detection Linux can often auto-detect the chipset, either during system installation, as described earlier; or by running an X configuration tool after installation.

Video card documentation Although some manufacturers attempt to hide the true identity of their products' chipsets, many do not. Because of this, it's quite worthwhile to check the product's documentation. This documentation might not use the word "chipset," though; it could use a phrase such as "powered by" or "based on."

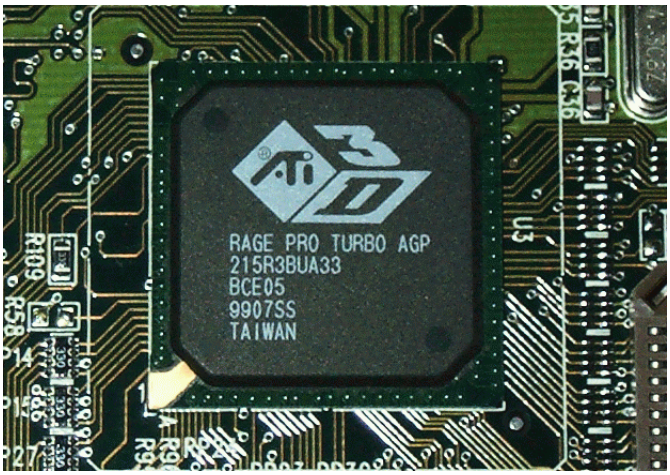
Windows driver report If the computer dual-boots to Windows, or if you've just bought a Windows system and intend to convert it to Linux, you can use the System tool in Windows to find out what driver (and thus, perhaps, what chipset) is installed. Double-click the System icon in the Windows Control Panel, then click the Device Manager tab. Click the plus sign next to the Display Adapters item. This will produce a list of the video cards installed in the computer. (Normally, there'll be just one.) Double-click the entry for more information; this produces the Properties dialog box for the video card, as shown in Figure 2.14. The driver and manufacturer name may be that of the video card or of the chipset.

FIGURE 2.14 The Windows Properties dialog box for the video card may provide information on the video chipset manufacturer.



Visual inspection You can examine your video card for signs of the chipset manufacturer. Most video cards are dominated by just one large chip. This chip probably has markings identifying the manufacturer and model number, as shown in Figure 2.15. Normally, the first line or two of text contain the relevant information; the remaining lines specify the revision number, place of manufacture, and so on.

FIGURE 2.15 Markings on chips can help identify the chipset for XFree86.



Some high-performance video card chipsets generate a great deal of heat, and for reliability, that heat must be dissipated by means of a heat sink—a finned metallic device that draws heat away from the chip so that it can be radiated into the surrounding air. Some boards also place a fan atop the heat sink. *Do not* attempt to remove a heat sink that's glued to a chip; doing so can damage the chip. Some manufacturers cover their chips with paper labels. These can be safely removed.

If you examine Figures 2.14 and 2.15, you'll see that they identify the chipset in the same way—as that of an ATI Rage Pro Turbo AGP. You won't always find consistency, however; sometimes a chipset may go by more than one name, or one identification method or another may reveal the board manufacturer's name rather than the chipset name. These situations need not

be too troublesome, though; they just mean that you'll have to look for a driver under more than one name.

One point to keep in mind when identifying the video card chipset is that some manufacturers produce both video cards and the chipsets that go on them (ATI and Matrox both fall into this category). Other companies produce just one or the other; for instance, Trident produces chipsets, and ELSA produces video cards. Thus, if you find that the name you uncover matches your card manufacturer's name, that's not necessarily a sign that you've failed to turn up the correct chipset manufacturer.

X Server Options for Linux

All major Linux distributions ship with a free X server known as *XFree86*. You can learn more about XFree86 at `http://www.xfree86.org`. One particularly important subpage on this site is `http://www.xfree86.org/current/Status.html`. This page hosts information about XFree86 compatibility with various chipsets, so it's a good place to go once you've discovered what chipset your board uses. You may find notes here on how to work around problems such as using an older or newer version of XFree86 than was shipped with your distribution.

Most Linux distributions shipped in 2001 use a 4.0.*x* version of XFree86. This version of the X server includes quite a few changes from the earlier 3.3.*x* version, but as of XFree86 4.0.3, there are still a few video chipsets that aren't supported in the new release. If you happen to have one of these, you'll have to use an older XFree86 3.3.*x* server, although you can continue to use the XFree86 4.0.*x* support programs and libraries.

Some video card and chipset manufacturers have made XFree86-compatible drivers available for their products. Thus, it's worth checking the Web sites maintained by your board and chipset manufacturers to see if there are drivers available. This is definitely true if the main XFree86 release doesn't include appropriate drivers, and it may be true even if there are drivers—a few standard XFree86 drivers are not accelerated, meaning that they don't support some of the video card's features for improving the speed of drawing or moving images. If the video card manufacturer has accelerated drivers but the main XFree86 distribution ships with unaccelerated drivers, you'll see a substantial improvement in video performance by installing the accelerated drivers.

Real World Scenario

XFree86 3.3.6 or 4.0.*x*?

In mid-2001, the transition to XFree86 4.0.*x* is still incomplete. Part of the reason for this is that, although most video card chipsets supported by 3.3.6 are also supported by 4.0.*x*, this isn't universally true. Some cards that are nominally supported by 4.0.*x* aren't supported as well as they were under 3.3.6 because the drivers don't implement all the features handled by the 3.3.6 drivers. There are also a few programs that don't work correctly under XFree86 4.0.*x*, although these are rare. On the other hand, XFree86 4.0.*x* supports some features not found in 3.3.6, such as multiscreen displays (creating an extra-large desktop displayed across two or more monitors).

Which should you use, then? The answer depends on your video card. Check the XFree86 Web site for information on support under both versions of the X server and make a preliminary judgment. If you run into problems with one, you might want to try the other. It's possible to install both XFree86 3.3.6 and 4.0.*x* servers and switch quickly between them by changing a symbolic link, as described in "Choosing the Server or Driver," later in this chapter. XFree86 4.0.*x* support is likely to improve as 2001 and 2002 wear on and the version number rises past 4.0.3, so be sure to check back periodically to see what improvements make their way into the 4.0.*x* series.

XFree86 occasionally doesn't support a device at all. You have three choices in this case:

Use the frame buffer device. The Linux kernel has some video drivers of its own. These can be accessed via the *frame buffer* XFree86 driver. For this to work, your kernel must include frame buffer support for your video chipset. This support has been greatly expanded in the 2.4.*x* kernel series.

Use a commercial X server. Two companies, Xi Graphics (`http://www.xig.com`) and Metro Link (`http://www.metrolink.com`), produce commercial X servers for Linux, known as Accelerated-X and Metro-X, respectively. These servers occasionally work on hardware that's not supported by XFree86, or produce better speed.

Replace the hardware. If you have a recalcitrant video card, the final option is to replace it. You may be able to swap with a Windows system that uses a different card, or you may need to buy a new one. Unfortunately, this isn't always an option; you can't replace the video card on a notebook computer, for instance.

Installing an X Server

Actually installing an X server is usually not very difficult; it's a matter of using your distribution's package management tools to install the software—much as you would any other software (described in Chapter 3). In most cases, this will be done during system installation, as described earlier in this chapter. You'll only have to manually install a server if you failed to install XFree86 during system installation or if you need to install a new server.

X normally comes in several packages. Only one package contains the X server proper; others provide support libraries, fonts, utilities, and so on.

In XFree86 4.0.*x*, one server package supports all video chipsets. The name of this package varies from one distribution to another, but it's likely to be called `XFree86-server`, `xserver-xfree86`, or something similar. You might install it using a command similar to the following in a distribution that uses RPMs:

```
# rpm -Uvh XFree86-server-4.0.2-11.i386.rpm
```

If your distribution uses Debian packages, something akin to the following will do the trick:

```
# dpkg -i xserver-xfree86_4.0.2-7_i386.deb
```

The result is the installation of a program called `XFree86`, which is usually stored in `/usr/X11R6/bin`. This program is a generic X server. It relies on separate driver modules, which are installed along with the main package in most cases. These driver modules probably reside in `/usr/X11R6/lib/modules/drivers`.

If you're using an XFree86 4.0.*x* driver provided by a video card manufacturer, follow the manufacturer's directions for installing the driver. Chances are you'll be required to copy a driver file to the XFree86 4.0 drivers directory, although the driver may come as an RPM or Debian package that will do this automatically.

If your card isn't supported by XFree86 4.0, but it is supported by XFree86 3.3.6, you'll need to install an old XFree86 3.3.6 X server. These come in files that typically include the name of the chipset, such as `XFree86-S3-3.3.6-19.i386.rpm`. This file provides an X server for various chipsets made by S3, some of which aren't supported in XFree86 4.0.3. If you had one of these chipsets, you could install the 3.3.6 server file, which would install an X server called `XF86_S3`. Running this server program rather than the 4.0.*x* `XFree86` executable would let you use your video card. (The upcoming section "Choosing the Server or Driver" specifies how to have the system launch a particular X server program.)

Configuring X

XFree86 is configured through the `XF86Config` file, which is usually located in `/etc` or `/etc/X11`. For XFree86 4.0.*x*, this file is sometimes called `XF86Config-4`. Using two filenames allows you to easily maintain separate configurations for XFree86 3.3.6 and 4.0.*x*. Keeping separate configurations can be useful if you're making the transition from the older X server to the new one. The commercial X servers have their own configuration files, but these files' formats differ from that described here for XFree86. Consult the package's documentation for configuration details.

When you configure X, you provide information on the input devices (the keyboard and mouse), the video card, and the monitor. Particularly important is information on the monitor's maximum horizontal and vertical refresh rates; if this information is wrong or missing, you might not get a display. This information can be obtained from the monitor's manual.

Methods of Configuring X

There are two methods available for configuring XFree86: by using configuration tools and by configuring manually. Configuration tools are programs that prompt you for information or obtain it directly from the hardware and then write the `XF86Config` file, which is a standard plain-text file like other Linux configuration files. Because this file is relatively complex, it's usually wise to begin with an automatic configuration, even if it's a flawed one. Manual configuration involves opening `XF86Config` in a text editor and changing its settings using your own know-how. You can use this method to tweak a working configuration for better performance or to correct one that's not working at all. Either way, you may need to configure X, test it, reconfigure X, test it, and so on for several iterations until you find a configuration that works correctly.

The X Configure-and-Test Cycle

If your X configuration isn't working correctly, you need to be able to modify that configuration and then test it. Many Linux distributions configure the system to start X automatically, however. Starting X automatically can make it difficult to test the X configuration; to a new Linux administrator, the only obvious way to test a new configuration is to reboot the computer.

A better solution is to kick the system into a mode in which X is *not* started automatically. On most distributions, this goal can be achieved by typing **`telinit 3`**. (In SuSE Linux, use **`telinit 2`** instead.) This action sets the computer to use runlevel 3 (or 2 for SuSE), in which X normally doesn't run. Chapter 6 covers the runlevel in more detail, but for now, know that setting the system to a runlevel of 3 or 2 normally shuts down the X session that launched automatically at system startup.

Once the X session is shut down, you can log in using a text-mode login prompt and tweak your X settings manually, or you can use text-based X configuration programs, as described shortly. You can then type **`startx`** to start the X server again. If the results are as you desire, quit from X and type **`telinit 5`** (**`telinit 3`** in SuSE) to restore the system to its normal X login screen. If after typing **`startx`** you don't get the results you want, you can try modifying the system some more.

If X is working minimally but you want to modify it using X-based configuration tools, you can do so after typing **`startx`** to get a normal X session running. Alternatively, you can reconfigure the system before taking it out of the X-enabled runlevel.

Another approach to restarting X is to leave the system in its X-enabled runlevel and then kill the X server. The Ctrl+Alt+Backspace keystroke does this on many systems, or you can do it manually with the `kill` command, after finding the appropriate process ID with the `ps` command, as shown below:

```
# ps ax | grep X
1375 ?   S  6:32 /etc/X11/X -auth /etc/X11/xdm/authdir/
# kill 1375
```

This approach works better on systems that don't map the running of X to specific runlevels, such as Debian and its derivatives.

Configuration Tools for XFree86 3.3.*x*

There are many configuration tools available for XFree86 3.3.*x*. Most distributions ship with at least one of these, often more:

`xf86config` This is the crudest of the XFree86 3.3.*x* configuration programs. It runs entirely in text mode, and it asks a series of questions regarding your hardware and configuration preferences. Because of its simple user interface, it can be used to create an initial X configuration. `xf86config` includes no provision for correcting errors, though, so if you mistype an entry, you must stop it and start again from scratch.

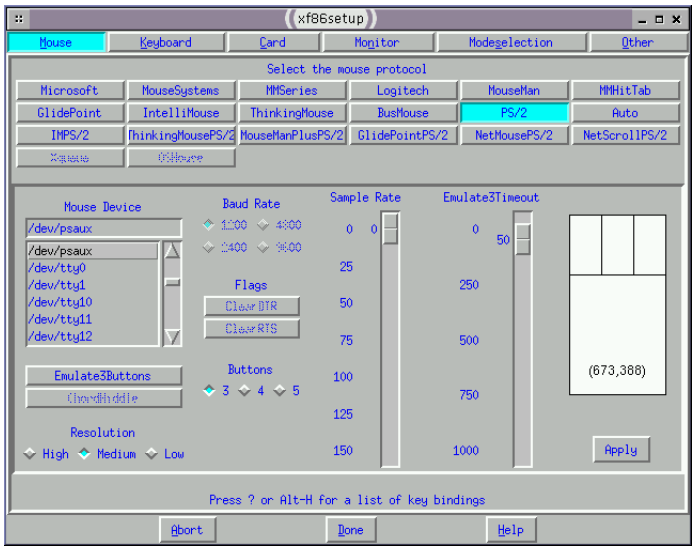
`Xconfigurator` This program runs in text mode when launched from a text-based login. In some distributions, it runs in GUI mode when launched from X. In either case, it uses menus to help you pick appropriate settings, but it still steps you through the configuration in a fixed order. As a general rule, it's easier to use than `xf86config`. Recent versions support both XFree86 3.3.*x* and 4.0.*x*.

`XF86Setup` This program runs only in GUI mode, so it works only once X is running at least minimally. Figure 2.16 shows it in operation. This program can be used to reconfigure X from a working but non-optimal mode. It allows you to move around the configuration as you see fit by selecting any of the major systems from the buttons along the top of the window (Mouse, Keyboard, Card, and so on).

As a general rule, `Xconfigurator` can be a good tool to use for configuring a new system for the first time, and `XF86Setup` is good for adjusting an existing system without digging into the `XF86Config` file.

Sometimes one configuration tool will produce a working X system but another won't. This is particularly likely when using finicky hardware, such as the LCD monitors used on notebook computers. If you can't seem to get a working X configuration out of one utility program, try another.

FIGURE 2.16 XF86Setup lets you jump around the configuration options to set details in any order you like.



Configuration Tools for XFree86 4.0.*x*

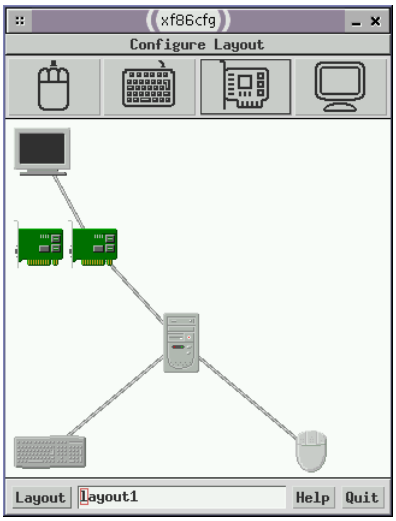
At its first release, configuration utilities for XFree86 4.0.*x* were scarce; however, matters have improved since then. There are now at least three utilities to help in XFree86 4.0 configuration, although not all distributions ship with all of them:

`XFree86` The `XFree86` server itself includes the capacity to query the hardware and produce an `XF86Config` file. To do so, type **`XFree86 -configure`** when no X server is running. The result should be a file called `/root/XF86Config.new`. This file might not produce optimal results, but it is at least a starting point for manual modifications.

`Xconfigurator` Recent versions of `Xconfigurator` can produce and modify the XFree86 4.0.*x* `XF86Config` file format, as well as the 3.3.*x* format of the file.

`xf86cfg` This is a utility that works only on XFree86 4.0.*x*. This program is another that works only once X is already running. Its user interface (shown in Figure 2.17), like that of `XF86Setup`, lets you jump around to configure different elements in whatever order you like. In `xf86cfg`, you right-click an icon and select the resultant Configure option to configure an element, or you can select other options (Remove, Disable, and so on) to perform other actions.

FIGURE 2.17 `xf86cfg` lets you configure XFree86 4.0.*x* using point-and-click operations.



Some of these utilities try to edit the `XF86Config` file, but others may be smart enough to look for `XF86Config-4` if that's where your distribution places the XFree86 4.0.*x* configuration file. If the utility finds the wrong file, it will probably crash, so you may need to temporarily juggle files to get the system to work.

Manually Editing the *XF86Config* File

Although some options have changed between XFree86 3.3.6 and 4.0.*x*, the basics of the `XF86Config` file remain unchanged. This file consists of a number of labeled sections, each of which begins with the keyword `Section` followed by the section name in quotes and ends with the keyword `EndSection`. Between these two lines lie lines that define features relevant to the configuration of that feature. There may also be comments, which are lines that begin with pound signs (#). For instance, here's a section that defines where the computer can find certain critical files:

```
Section "Files"
    RgbPath   "/usr/X11R6/lib/X11/rgb"
    # Multiple FontPath entries are allowed
    FontPath  "/usr/X11R6/lib/X11/fonts/75dpi"
    FontPath  "/usr/X11R6/lib/X11/fonts/Type1"
EndSection
```

The pages that follow tell you what sections and critical options within these sections exist to modify XFree86's operation. You should then be able to edit the `XF86Config` file directly or use a configuration utility to do the job. (The configuration utilities tend to use terminology that's similar to that used in the configuration file, so figuring out what to change with a utility isn't difficult if you know for what option you're looking.)

If you have a working configuration, be sure to back up `XF86Config` before modifying it. If you mistakenly delete or modify some critical line, you can easily end up with a system that won't start X at all, and without a backup or a perfect memory of what you changed, it can be difficult to restore even a partially functioning system.

Setting Miscellaneous Options

Some sections of the `XF86Config` file relate to miscellaneous options or those that require just a handful of lines to set properly. (The big video sections often boast dozens of lines of configuration options.) Nonetheless, getting these settings right is important to a functioning XFree86 system.

This section emphasizes the configuration of XFree86 4.0.*x*. Some 4.0.*x* configuration lines begin with the keyword `Option` and enclose the option name in quote marks. When configuring XFree86 3.3.*x*, you should omit the `Option` keyword and the quote marks around the option name.

Configuring Paths

The `Files` section hosts information on the locations of important files. The entries you're most likely to change relate to the locations of X's fonts. These are handled through the `FontPath` option line. Examples of the use of this line include the following:

```
FontPath   "/usr/X11R6/lib/X11/fonts/Type1"
FontPath   "unix/:-1"
FontPath   "tcp/fontserver.example.com:7101"
```

The first of these lines indicates a directory in which fonts may be found. The second refers to a *font server* that runs locally, and is not accessible to

other systems. The final line points to a font server that runs on another computer (*fontserver.example.com*) on port 7101. A font server is a program that delivers fonts to local or remote computers. Some Linux distributions use font servers for local font handling, and networks sometimes use them to reduce the effort of administering fonts. You don't need to use a font server if you don't want to, but if your distribution uses a local font server by default, you should leave its reference intact in `XF86Config`. A single `XF86Config` file can have multiple `FontPath` lines; X searches for fonts in each of the specified locations in order.

Configuring the Keyboard

The `Keyboard` section defines the operation of the keyboard in XFree86. In most cases, there's little need to modify most `XF86Config` keyboard settings. One that you might want to change, however, is the `AutoRepeat` line. When you press and hold a key, the system begins repeating it, as if you were repeatedly pressing the key. This line controls the rate at which keys repeat when running X. This line takes the form:

```
Option "AutoRepeat"  delay  rate
```

The *`delay`* is the time in milliseconds (ms), thousandths of a second, before the system begins repeating a key, and *`rate`* is the interval between repeats once they begin. For instance, if *`delay`* is 500 and *`rate`* is 200, the system waits 500ms after the key is first depressed, and thereafter produces another character every 200ms (five per second) until you release the key.

Users can override the default keyboard repeat rate by setting this option using a desktop environment's control utilities or various other programs.

In some cases, you might also want to adjust the following lines:

```
Option "XkbModel" "pc105"
Option "XkbLayout" "us"
```

These set the keyboard model and layout. The model relates to the number of keys and their placement, and the layout determines what character each key produces. The layout is used to specify keyboards for different nationalities or languages, which often contain slightly different key selections.

Configuring the Mouse

The `InputDevice` section defines the mouse. This section is typically quite short, as shown here:

```
Section "InputDevice"
    Identifier "Mouse1"
    Driver      "mouse"
    Option "Protocol"    "PS/2"
    Option "Device"      "/dev/psaux"
    Option "Emulate3Buttons"
    Option "Emulate3Timeout"    "50"
EndSection
```

In XFree86 3.3.*x*, this section is called `Pointer`, and it lacks the `Identifier` and `Driver` options.

Chances are you won't need to modify the `Identifier` or `Driver` options. The `Protocol` is the software protocol used by mice. It's most often `PS/2`, but it may be something else (such as `Microsoft` or `Logitech`), particularly for older serial mice. The `Device` option points to the Linux device file with which the mouse is associated. This is sometimes `/dev/mouse`, which is a symbolic link to the real device file, such as `/dev/psaux` (for PS/2 mice), `/dev/usb/usbmouse` (for USB mice), or `/dev/ttyS0` or `/dev/ttyS1` (for serial mice). (Chapter 7 covers the `/dev` directory and its contents in more detail.) `Emulate3Buttons` tells XFree86 to treat simultaneous presses of the two buttons of a 2-button mouse as if they were the third button, and `Emulate3Timeout` tells the system how close (in milliseconds) those two presses must be. If the system has a 3-button mouse to begin with, these options should be commented out or deleted.

X programs frequently use the middle button; for instance, text editors use it for pasting text. Therefore, any Linux workstation should be equipped with a genuine 3-button mouse rather than a 2-button device. Although the `Emulate3Buttons` option allows you to use a 2-button mouse in Linux, doing so is awkward.

Setting Monitor Options

Some of the trickiest aspects of X configuration relate to the monitor options. You set these in the `Monitor` section, which has a tendency to be quite large, particularly in XFree86 3.3.6. A shortened `Monitor` section looks like this:

```
Section "Monitor"
    Identifier "Iiyama"
    ModelName  "VisionMaster Pro 450"
    HorizSync  27.0-115.0
    VertRefresh 50.0-160.0
    # My custom 1360x1024 mode
    Modeline "1360x1024" 197.8 \
              1360 1370 1480 1752 \
              1024 1031 1046 1072 -HSync -VSync
EndSection
```

The `Identifier` option is a free-form string that contains information that's used to identify a monitor in a later section. This later section links together various components of the configuration. The `Identifier` can be just about anything you like. Likewise, the `ModelName` option also can be anything you like; it's used mainly for your own edification when reviewing the configuration file.

As you continue down the section, you'll see the `HorizSync` and `VertRefresh` lines, which are extremely critical; they define the range of horizontal and vertical refresh rates that the monitor can accept, in kilohertz (kHz) and hertz (Hz), respectively. Together, these values determine the maximum resolution and refresh rate of the monitor. Despite the name, the `HorizSync` item alone doesn't determine the maximum horizontal refresh rate. Rather, this value, the `VertRefresh` value, and the resolution determine the monitor's maximum refresh rate. XFree86 selects the maximum refresh rate that the monitor will support, given the resolution you specify in other sections. Some X configuration utilities show a list of monitor models or resolution and refresh rate combinations (such as "800 × 600 at 72 Hz") to obtain this information. This approach is often simpler to handle, but it's less precise than entering the exact horizontal and vertical sync values.

Don't set random horizontal and vertical refresh rates; particularly on older hardware, setting these values too high can actually damage a monitor. (Most modern monitors ignore signals presented at too high a refresh rate.)

To settle on a resolution, X looks through a series of *mode lines*, which are specified via the `Modeline` option. Computing mode lines is tricky, so I don't recommend you try it unless you're skilled with such matters. The mode lines define combinations of horizontal and vertical timing that can produce a given resolution and refresh rate. For instance, a particular mode line might define a 1024 × 768 display at a 90Hz refresh rate, and another might represent 1024 × 768 at 72Hz.

Some mode lines represent video modes that are outside the horizontal or vertical sync ranges of a monitor. X can compute these cases and discard the video modes that a monitor can't support. If asked to produce a given resolution, then X searches all the mode lines that accomplish the job, discards those that the monitor can't handle, and uses the remaining mode line that creates the highest refresh rate at that resolution. (If no mode line supports the requested resolution, X drops down to another specified resolution, as described shortly, and tries again.)

As a result of this arrangement, you'll see a large number of `Modeline` entries in the `XF86Config` file for XFree86 3.3.*x*. Most end up going unused because they're for resolutions you don't use or because your monitor can't support them. You can delete these unused mode lines, but it's usually not worth the bother.

XFree86 4.0.*x* supports a feature known as *Data Display Channel (DDC)*. This is a protocol that allows monitors to communicate their maximum horizontal and vertical refresh rates and appropriate mode lines to the computer. The `XFree86 -configure` command uses this information to generate mode lines, and on every start, the system can obtain horizontal and vertical refresh rates. The end result is that an XFree86 4.0.*x* system can have a substantially shorter `Monitor` section than is typical with XFree86 3.3.*x*.

Setting Video Card Options

Your monitor is usually the most important factor in determining your maximum refresh rate at any given resolution, but X sends data to the monitor only indirectly, through the video card. Because of this, it's important that

you be able to configure this component correctly. An incorrect configuration of the video card is likely to result in an inability to start X.

Choosing the Server or Driver

One of the most fundamental aspects of video card configuration is specifying the correct server or driver. The server is most often selected via a symbolic link—typically `/usr/X11R6/bin/X` or `/etc/X11/X`. This link points to the actual X server, such as `/usr/X11R6/bin/XFree86` for XFree86 4.0.*x*, or a server whose name takes the form `XF86_Driver` for XFree86 3.3.*x*.

A symbolic link is a special type of file that contains no data itself but that points to a file that contains the real data. These links are similar to shortcuts in Windows, and can be created with the `ln` command. Type **`man ln`** at a Linux command prompt to learn more.

Sometimes, Linux configures itself to use a server such as `XF86_VGA16`, which runs the system in a lowest-common-denominator 640 × 480 display mode. The solution to this problem is to change the server link. Sometimes you must do this manually, but sometimes installing a new server package runs a post-installation script that will do the job automatically.

In XFree86 4.0.*x*, the `XFree86` server program uses driver modules that are stored in separate files. The server can't determine what module is required automatically, however. Instead, you must give it that information in the `XF86Config` file. In particular, the driver module is set by a line in the `Device` section, which resembles the following:

```
Driver "ati"
```

This line sets the name of the driver. The drivers reside in the `/usr/X11R6/lib/modules/drivers/` directory. Most of the drivers' filenames end in `_drv.o`, and if you remove this portion, you're left with the driver name. For instance, `ati_drv.o` corresponds to the `ati` driver.

The `xf86cfg` utility provides a large list of chipsets and specific video card models, so you can select the chipset or board from this list to have the utility configure this detail.

Setting Card-Specific Options

The `Device` section of the `XF86Config` file sets various options related to specific X servers. A typical `Device` section resembles the following:

```
Section "Device"
    Identifier  "ATI Mach64"
    VendorName  "ATI"
    BoardName   "Xpert 98"
    Driver      "ati"
    VideoRam    8192
EndSection
```

The `Identifier` line provides a name that's used in the subsequent `Screen` section to identify this particular `Device` section. (`XF86Config` files frequently host multiple `Device` sections—for instance, one for a bare-bones VGA driver and one for an accelerated driver.) The `VendorName` and `BoardName` lines provide information that's useful mainly to people reading the file.

The `Driver` line, as described earlier, is used by XFree86 4.0.*x* to locate the driver to be used. An XFree86 3.3.*x* `XF86Config` file won't contain this line.

The `VideoRam` line is unnecessary with many servers and drivers because the driver can detect the amount of RAM installed in the card. With some devices, however, you may need to specify the amount of RAM installed in the card, in kilobytes. For instance, the preceding example indicates a card with 8MB of RAM installed.

Many drivers and servers support additional server-specific options. These may enable support for features such as hardware cursors (special hardware that allows the card to handle mouse pointers more easily) or caches (using spare memory to speed up various operations). Consult the `XF86Config` man page or other server-specific documentation for details.

Setting Screen Options

The `Screen` section ties together the other sections. A short example is:

```
Section "Screen"
    Identifier "screen1"
    Device     "ATI Mach64"
    Monitor    "Iiyama"
    DefaultColorDepth 16
    Subsection "Display"
```

```
        Depth       8
        Modes       "1280x1024" "1024x768" "640x400"
    EndSubsection
    Subsection "Display"
        Depth       16
        Modes       "1024x768" "800x600" "640x480"
        Virtual     1280 1024
        ViewPort    0 0
    EndSubsection
EndSection
```

There are several key points in this section that should be emphasized:

- The `Identifier` specifies an overall configuration in XFree86 4.0.*x*. In this version of XFree86, a configuration file can hold multiple `Screen` sections, as described shortly. In XFree86 3.3.*x*, there's no `Identifier` line; instead, there's a `Driver` line, which identifies the X server (such as `svga`, `vga16`, or `accel`; this last one stands in for all the specialized accelerated servers, such as `XF86_S3`). When the 3.3.*x* server reads the `XF86Config` file, it locates the appropriate `Screen` section based on the `Driver` line.
- The `Device` and `Monitor` lines point to specific `Device` and `Monitor` sections, respectively.
- The `DefaultColorDepth` line specifies the number of bits per pixel to be used by default. For instance, the preceding example sets this value to `16`, so a 16-bit color depth is used, resulting in 2^{16}, or 65,536, possible colors.
- Each `Subsection` defines a particular display type. These have associated color depths (specified by the `Depth` line) and a series of resolutions (specified by the `Modes` line). The system tries each resolution specified by the `Modes` line in turn, until it finds one that works. There are also various optional parameters, such as `Virtual` (which defines a virtual screen that can be larger than the one that's actually displayed) and `ViewPort` (a point within that virtual display at which the initial display is started).

In XFree86 4.0.*x*, one final section is required: the `ServerLayout` section. This section consists of lines that identify the default `Screen` section and link

it to mouse and keyboard definitions. (In 3.3.*x*, only one mouse and keyboard may be defined; but 4.0.*x* is more flexible in this respect.) For instance, a typical configuration will include a `ServerLayout` section resembling the following:

```
Section "ServerLayout"
    Identifier  "layout1"
    Screen      "screen1"
    InputDevice "Mouse1" "CorePointer"
    InputDevice "Keyboard1" "CoreKeyboard"
EndSection
```

Normally, an `XF86Config` file for 4.0.*x* will have just one `ServerLayout` section, but by passing the `-layout` parameter to XFree86, you can tell the server to use a different `ServerLayout` section, if one is present. You might use this to start X using a different mouse, for instance—say, a USB mouse on a notebook rather than the built-in PS/2 touch pad.

Configuring Window Managers

As noted earlier in this chapter, a working X configuration consists of many separate components. These components are selected by various people: the programmer who wrote the software being run, the system administrator, and even the user. You must understand how these components fit together in order to effectively administer and use a Linux computer.

The X server lies at the foundation of the X hierarchy. It provides drivers for the video hardware, mouse, and other peripherals, as well as basic low-level graphics features including the ability to draw lines, circles, text, and so on. It can also display windows, although a "raw" X window would barely be recognizable as such to a user because it would lack the borders and drag bars that are so integral to windows. This is where a second component comes into play: the window manager.

Understanding the Role of the Window Manager

The *window manager* provides a means for users to control individual windows. This tool surrounds an X window with a border, which is both decorative and functional. The border may allow users to resize the window, and

it typically includes a portion at the top or to the side that allows users to drag the window around (the *drag bar*). Window managers also control which window has *focus*—that is, which window accepts input from the keyboard or mouse. There are several different focus models available, such as focus-follows-mouse (you need only move a mouse over a window to give it focus), click-to-focus (click in a window to give it focus), and so on. Most window managers support at least two of these, although the most common today is click-to-focus, which is also used by Microsoft's Windows and Apple's MacOS. Likewise, window managers control when a window is moved to the front of an overlapping stack of windows. This may occur when a window is given focus, or it might require some other action.

In addition to handling individual windows, the window manager controls the screen as a whole. Most window managers provide some form of menu through which users can run programs and control aspects of the window manager's operation. This menu might be a pop-up menu that appears when the user right-clicks the desktop (any part of the screen that's not covered by a window or other object); or it could be accessible through an icon that's permanently visible, typically in the lower-left corner of the screen. Either way, users can customize this feature by editing configuration files in their home directories—to add programs to a program launch menu, for instance.

Many Linux window managers include a *pager* function. This feature allows you to maintain several workspaces and switch between them by clicking a button or selecting an option from a menu. You might have one workspace in which you run a word processor, another in which your Web browser runs, and a third with programming tools. A pager can be a great way to reduce clutter while still running many programs.

Many window manager features, such as pagers and program-launch tools, are also provided by other programs, such as desktop environments (described shortly).

Window managers differ from one another in how elaborately they support various roles. Some window managers are bare bones and provide only very simple controls. Others provide elaborate menus, pagers, and so on. As

a result, window managers can differ substantially in overall appearance and functionality. Figures 2.18 and 2.19 illustrate how different two window managers can be. Figure 2.18 shows IceWM using its Helix theme (this is the window manager used for most screen shots in this book), while Figure 2.19 shows the WMX window manager. IceWM sports a "task bar" at the bottom of the screen that includes buttons for launching programs and switching to windows even if they're completely obscured. WMX features an unusual side-mounted drag bar and very minimalistic window controls. Note that the contents of the windows (including the menus in the Opera Web browser window) are identical, except for the fact that the clock shows a different time in the two screen shots. The window manager does *not* manage the contents of windows—that detail is left to the program and its widget set, as described shortly.

FIGURE 2.18 IceWM is in many ways a typical window manager that provides popular window controls.

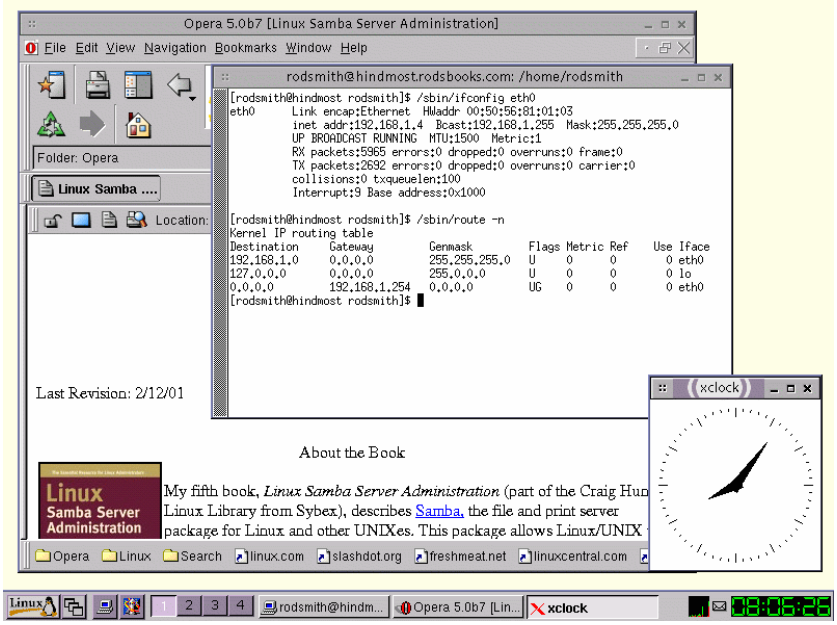
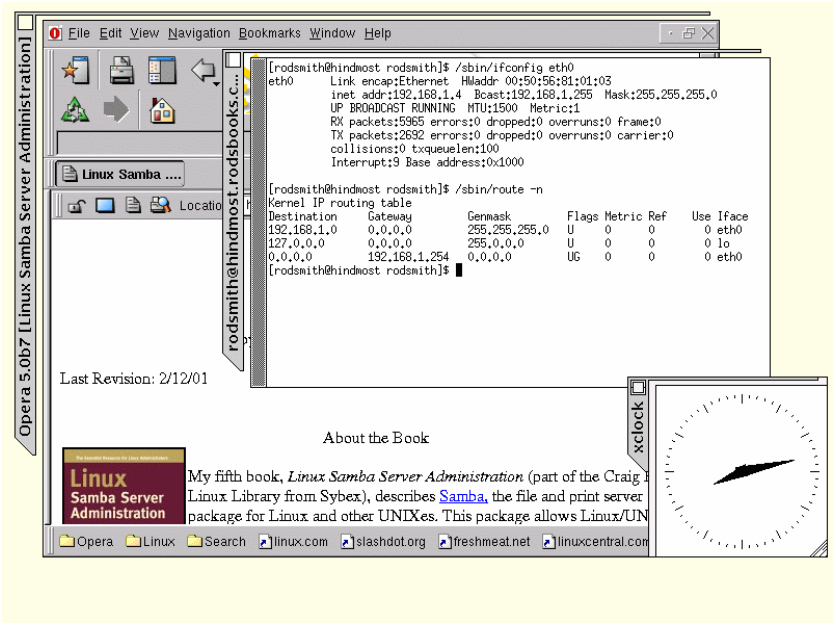


FIGURE 2.19 WMX provides few controls and uses unusual side-mounted drag bars.



Common Window Managers for Linux

Linux supports a wide array of window managers. An excellent site for learning about the available choices is the Window Managers for X Web page (`http://www.plig.org/xwinman`). Some of the more popular choices today include the following:

KWM The K Window Manager (KWM) is part of the *K Desktop Environment (KDE)*, described shortly. Because of KDE's popularity, KWM is a very common window manager in Linux, although it's seldom used except as part of KDE. The KDE Web site, `http://www.kde.org`, includes information on KWM.

Sawfish Sawfish (formerly known as Sawmill) is the default window manager in the *GNU Network Object Model Environment (GNOME)* versions 1.2 and later, but it's available and is often used separately. You can learn more at `http://sawmill.sourceforge.net`.

Enlightenment This is a window manager that's designed with maximum graphical configuration in mind—it's possible to make Enlightenment look like just about anything. It was the default window manager in GNOME 1.0 and 1.1. The Enlightenment Web site is `http://www.enlightenment.org`.

IceWM This window manager aims to be fairly small but provide the most commonly desired features, such as a pager and some customizability of appearance. Although not the default, it integrates well with GNOME. You can read more at `http://icewm.sourceforge.net`.

Window Maker Window Maker takes its inspiration from the old NeXT computer's interface, although it doesn't provide anything remotely resembling NeXT compatibility to a Linux system. Like IceWM, it aims to be small but functional, and integrates well with GNOME. It can also work as a KDE window manager. The official Window Maker Web page is `http://www.windowmaker.org`.

This list is far from comprehensive. Some window managers, such as the venerable FVWM and TWM, were popular once but are no longer the default on most systems. Others, such as WMX, have never been popular but may offer something desirable to at least some users.

Which window manager is best? That's like asking which fruit is best. Although you can certainly apply objective measures to window manager evaluation, in the end it's subjective matters that are important. For instance, do you find the configuration file format intuitive? Do you like the window manager's focus policies? Does it integrate with your desktop environment (if you use one)? As a user, your best bet is to try several window managers until you find one you like. As a system administrator, you should probably provide several options to your users, although doing so can increase your workload—if a user comes to you with a window manager problem, you'll need to know how to solve that problem, which could be unique to just one window manager.

Running a Window Manager

The first step in running a window manager is installing it. If you installed X on your system, you almost certainly also installed at least one window manager, whether you knew it or not. In fact, if the computer boots into X mode and you can log in and see window borders, or if you can boot in text mode and type **`startx`** and see windows with borders, then a window manager is automatically running. This is normal on most systems; default X login scripts include calls to a standard window manager.

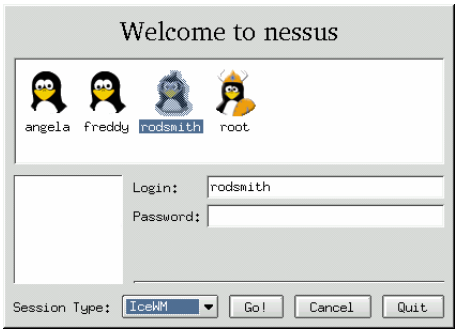
Assuming you want to explore window manager options or provide more choices to your users, though, you should check to see what's currently

installed, or try installing or updating window managers. The Window Managers for X Web site, mentioned earlier, is a good source of leads to new window managers. Most window manager Web sites include binary files you can install using `rpm`, `dpkg`, or `tar`, as described in Chapter 3.

Window managers are run by users, or at least by users' startup scripts. Therefore, as a system administrator, you don't normally need to modify any settings to make window managers available. You can, however, change the default window manager. How you do this varies from one system to another. Files that might set this include `/etc/X11/Xconfig`, `/etc/X11/xinit/xinitrc`, `/etc/X11/xdm/Xsession`, and `/etc/sysconfig/desktop`.

When a system uses the K Desktop Manager (KDM) or GNOME Desktop Manager (GDM) program as a GUI login program, as is common for systems that start directly into X, the KDM or GDM program provides a way for users to select the default login environment. For instance, Figure 2.20 shows a KDM login dialog box. Users can select a desktop environment or window manager from the Session Type menu.

FIGURE 2.20 The GDM and KDM programs both provide users with a way to select the login environment.



If the login screen doesn't give the window manager options you want, you can usually modify a user's login files to get the job done. Common X login scripts include `.xsession` and `.Xlogin` for GUI logins, and `.xinitrc` when starting X from a text-mode login. You may need to modify different files depending upon the login option chosen. These scripts include a series of commands to be run when logging in. As scripts, they must normally be executable, as described in "File Permissions" in Chapter 4. They can be modified to launch the desired window manager as the last item run. For

instance, the following script launches an `xterm` window and the WMX window manager:

```
xterm &
wmx
```

It's important to note that most commands *before* the window manager command must be followed by an ampersand (`&`). This tells Linux to launch the program and move on to the next command before the first one finishes. The window manager command should normally *not* include an ampersand, though. This way, Linux logs the user out when the user quits from the window manager. (All window managers include some sort of quit command. Today, many of these appear on a menu as "log out.")

Understanding Widget Sets

Figures 2.18 and 2.19 illustrate the effects of two window managers on the appearance of Linux programs. Another important determinant of how a program appears on screen and behaves is the *widget set* it uses. This is a set of programming tools that builds upon what's provided by X. Widget sets provide programmers with easy ways to display menus, dialog boxes, and the like—in short, most controls inside a window. As such, they greatly influence the overall feel of programs written for X.

Most OSs provide the functionality that's part of a Linux widget set in the OS itself, but in Linux these features are separate. Therefore, two different Linux programs can use different widget sets, and as a consequence, they will have very different appearances and behaviors. One might have menu bars that use a large font and require you to hold the mouse button to select menu items, whereas another might use small fonts and require you to click, but not hold, the mouse button to select items. Differences such as these can be confusing to Linux newcomers.

Fortunately, Linux programmers are increasingly shifting to just two widget sets: Qt (`http://www.trolltech.com`) and GTK+ (`http://www.gtk.org`). Although these widget sets differ from each other, they're more similar than are some other X widget sets, so as programmers increasingly use these two, the overall consistency of Linux programs improves. Other widget sets that enjoy some degree of popularity include Motif (`http://www.opengroup.org/motif`), which has long been popular among commercial

Unix and Linux programs; LessTif (`http://www.lesstif.org`), a fully open source clone of Motif; and Athena, which is part of the standard collection of programs associated with X.

As a system administrator or user, you have no control over the widget set a program uses, unless you want to rewrite the program to use a different widget set. The program's author decides on a widget set, and changing it can be a tedious proposition. As a system administrator, though, you may need to install the widget set itself. Many widget sets ship as *libraries*, which are files that provide code in support of other programs. Several programs can use a single library, thus saving disk space and memory. If a program uses a given widget set and is *dynamically linked* to that widget set, that means you must have the library installed on your computer. Programs can also be *statically linked* to a library, meaning that the relevant library calls are included as part of the program's main file. This latter approach is most common with Motif programs.

GNOME uses GTK+, and KDE is built on Qt. As a result, people often refer to programs as "GNOME programs" or "KDE programs," when in fact the programs merely use GTK+ or Qt. It's usually possible to use a Qt program within GNOME, or a GTK+ program in KDE; you just need the appropriate libraries installed. A few programs, however, rely upon other elements of the desktop environment. These may or may not work in another environment. If they do, they'll require that additional parts of the competing environment be installed.

Configuring Desktop Environments

A window manager is a useful tool, but by no means does it create a complete GUI experience. Users of computers running MacOS, Windows, OS/2, or most other GUI environments are accustomed to having a variety of tools at hand. These include a file manager (for copying files, launching programs, and so on), small utility programs (a text editor, calculator, and the like), system configuration tools (to set keyboard repeat rate, default fonts, and so on), and assorted other miscellaneous tools. Although programs like these are available in Linux, they don't always work together well. For instance, default fonts set in one program might not be used in another.

The *desktop environment* takes care of these needs. A desktop environment is an integrated suite of programs designed to provide all the creature comforts users of other GUI OSs are accustomed to having at their fingertips. A desktop environment builds from a window manager, adding utility and configuration programs to make a comfortable working environment.

Common Desktop Environments for Linux

Several desktop environments are available in Linux, the most common of which are listed below:

KDE KDE (`http://www.kde.org`) is probably the most popular desktop environment in Linux. It's built upon its own window manager, KWM, and the Qt widget set. KDE is in the 2.1 series in mid-2001, and it is fairly advanced and complete. Most Linux distributions favor KDE.

GNOME GNOME (`http://www.gnome.org`) is KDE's principal competitor. It's the default desktop environment for a few distributions, including Red Hat and Debian. GNOME is built upon GTK+, and uses the Sawfish window manager in its 1.2 incarnation. GNOME is not quite as polished as is KDE (GNOME is newer), but the two are very close.

CDE The Common Desktop Environment (CDE) is a commercial desktop environment common on commercial Unix systems. It's built on the Motif widget set. A Linux incarnation is available as DeXtop from Xi Graphics (`http://www.xig.com`).

XFce This is an open source desktop environment modeled after CDE, but it uses the GTK+ widget set. It's not as complete as KDE or GNOME, but it's also less disk- and RAM-intensive. You can find out more at `http://www.xfce.org`.

Most distributions ship with both the KDE and GNOME desktop environments, although as just noted, most distributions make one or the other the default, and put more effort into getting the configuration of the default environment right. Except for CDE, all of these environments are open source software, so obtaining and distributing them isn't a problem.

As a system administrator, your biggest choice with respect to desktop environments is which ones to support. You can install more than one desktop environment on a computer, and you can allow users to select between them when they are logging in, as described earlier with respect to window managers. Installing multiple desktop environments, however, can take a heavy toll on disk space. Both GNOME and KDE, for instance, can easily consume well over 100MB of disk space. On the other hand, an increasing

number of applications require substantial subsets of one or the other environment installed, even if the application isn't officially part of either KDE or GNOME.

Running a Desktop Environment

From both the system administrator's and the user's point of view, running a desktop environment is done in much the same way as one runs a window manager. KDM and GDM login programs may be configured to start desktop environments. If a user has a customized login script, you can include the `startkde` or `startgnome` commands in it to start KDE or GNOME, in place of a call to a simple window manager, as described earlier.

Both KDE and GNOME include the ability to run associated programs from menus located in the lower-left corner of the screen. The KDE menu resembles a K, and the GNOME menu looks like a G-shaped footprint. Not all the programs listed on these menus are associated with their respective desktop environments, but many of them are. Browse through the options to learn what they are. Some options allow you to configure features of the desktop environment itself. In particular, the GNOME Control Center and the KDE Control Center both let you obtain information on your system's hardware and adjust your environment (features like your keyboard repeat rate, screen saver, and so on).

Although desktop environments aim to provide a friendly interface for new Linux users, they rely upon textual configuration files similar to those used by other Linux programs. KDE's settings are stored in the `.kde` subdirectory of the user's home directory, and GNOME uses `.gnome` for this purpose. In principle, you can change your desktop environment's operation by editing these files directly. In most cases, though, it's easier to use the GUI configuration tools provided for this purpose. If an environment behaves strangely or won't start, deleting the entire set of configuration files might help—but this action will result in the loss of any customizations the user may have created. You might try renaming the backup directory; that way you can restore individual configuration files, if you know of specific files that might otherwise be tedious to recover.

Mixing and Matching Desktop Environment Components

Although KDE, GNOME, CDE, and XFce are all designed to be integrated environments, it's possible to mix and match components from all these systems. For instance, you might want to use KDE's audio CD player but GNOME's mixer (to control the volume of CDs you play). This is certainly possible. The main drawback is that you're likely to lose some functionality because the programs may not interact with each other as well as they might. For instance, a personal scheduler program and e-mail utility might share a contact database if designed to do so; but if you mix and match components, chances are you'll lose the ability to share this file.

In fact, it's possible to build your own environment from bits and pieces of others, as well as from unaffiliated programs. This approach can actually save memory because you're not likely to expend RAM on tools and features you never use. On the other hand, mixing and matching components in this way will sacrifice functionality, as just described.

Whether it's done with individual components or to generate an entirely unique desktop environment, mixing and matching components is something that you need not be concerned with as a system administrator. It's individual users who will do this.

Summary

The Linux installation process involves implementing many of the decisions you made when planning the installation (Chapter 1). You must run the Linux installer, choose the broad outlines of how you want to install (text, GUI, etc.), set assorted options, choose packages, and set the installer running. If all goes well, when the installer has done its job, you can reboot the computer into Linux.

After doing a basic Linux installation, the main task that may remain is to configure X. Although the installation usually tries to do this, it doesn't always succeed, or it may create a suboptimal configuration. You can use utilities or edit the `XF86Config` file by hand to optimize your X configuration.

In addition to doing basic X configuration, you may want to change the default choices for window managers and desktop environments. These tools sit atop X in the GUI hierarchy, and they provide user interface and control elements that X doesn't provide. These elements can be configured

on a user-by-user basis, so a single Linux system can accommodate users with different personal preferences.

Exam Essentials

Describe when it's most appropriate to use CD-ROM and network installations. CD-ROM installations are most convenient when installing to systems with poor network connectivity or when you have a CD-ROM and want to install quickly. Network installations are convenient when you are installing several systems simultaneously or when you don't have a Linux CD-ROM or a CD-ROM drive on the target system.

Ascertain what type of interaction is most appropriate during installation. GUI- and text-based installations are good for when you are installing a single system or when you are preparing a template for scripted installations with some distributions. Automatic scripted installations are convenient when you are installing nearly identical systems on multiple computers.

Explain why you might create user accounts at installation. Creating one or more user accounts when installing Linux allows you to immediately use the computer after installation, thus making it useable more quickly. Install-time user creation tools, however, don't usually offer the range of options available in normal user management tools.

Describe the reasons for choosing packages in groups or individually. You can pick packages in groups, which saves you considerable time during installation but will likely leave you to add and remove packages later. Alternatively, you can pick individual packages at install time, which lengthens the time required to configure the installation, but saves you time in subsequent configuration.

Determine what video chipset your system uses. Many manufacturers document the video card chipset in their manuals or on the product boxes. You can also check the Microsoft Windows System control panel or visually inspect the board, if the manufacturer did not make the information readily available.

Summarize how X determines the monitor's refresh rate. X uses the monitor's maximum horizontal and vertical refresh rates and a series of fixed mode lines, which define particular timings for various video resolutions. X picks the mode line that produces the highest refresh rate allowed by the monitor at the specified resolution.

Describe the role of the window manager in X. The window manager provides decorative and functional borders around windows. It also controls the screen as a whole, providing some means of launching useful programs from menus.

Summarize the relationship between window managers and desktop environments. A window manager is one component of a desktop environment. Desktop environments also include file managers, configuration tools, and various small utilities.

Commands in This Chapter

Command	Description
`xf86config`	Text-based XFree86 3.3.*x* configuration program
`Xconfigurator`	Text- or GUI-based XFree86 3.3.*x* and 4.0.*x* configuration program
`XF86Setup`	GUI-based XFree86 3.3.*x* configuration program
`XFree86`	XFree86 4.0.*x* server that can automatically produce its own configuration file
`xf86cfg`	GUI-based XFree86 4.0.*x* configuration program

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

- bad block
- Coordinated Universal Time (UTC)
- creating a filesystem
- Data Display Channel (DDC)
- drag bar
- dynamically linked
- focus
- font server
- formatting
- frame buffer
- GNU Network Object Model Environment (GNOME)
- Greenwich Mean Time
- K Desktop Environment (KDE)
- mode line
- pager
- proxy server
- statically linked
- widget set
- X
- X client
- X server
- X Window System
- XFree86

Review Questions

1. Which of the following best describes a typical Linux distribution's method of installation?

   A. The installation program is a small Linux system that boots from floppy, CD-ROM, or hard disk to install a larger system on the hard disk.

   B. The installation program is a set of DOS scripts that copies files to the hard disk, followed by a conversion program that turns the target partition into a Linux partition.

   C. The installation program boots only from a network boot server to allow installation from CD-ROM or network connections.

   D. The installation program runs under the Minix OS, which is small enough to fit on a floppy disk but can copy data to a Linux partition.

2. Which of the following is an advantage of a GUI installation over a text-based installation?

   A. GUI installers support more hardware than do their text-based counterparts.

   B. GUI installers can provide graphical representations of partition sizes, package browsers, and so on.

   C. GUI installers can work even on video cards that support only VGA graphics.

   D. GUI installers better test the system's hardware during the installation.

3. When specifying a pointer type during installation, what information must you provide? (Choose all that apply.)

   A. Whether it's a mouse or trackball

   B. Whether it uses an optical or a mechanical mechanism

   C. The type of interface it uses (PS/2, serial, or USB)

   D. The number of buttons the pointer has

4. Which of the following tools may you use when creating partitions for Linux? (Choose all that apply.)

   A. Linux's `fdisk` from an emergency disk, run prior to the system installation

   B. PowerQuest's PartitionMagic, or similar third-party utilities

   C. Disk Druid or another distribution-specific install-time utility

   D. The DOS `FORMAT` utility, run prior to the system installation

5. What mount point should you associate with swap partitions?

   A. `/`

   B. `/swap`

   C. `/boot`

   D. None

6. What does a bad block check do?

   A. It verifies that the data written to disk during installation is correct.

   B. It checks for hard disk sectors that are unreliable and ensures they aren't used.

   C. It checks dependency information in packages for internal consistency.

   D. It looks for runs of disk sectors holding lost data.

7. Why is it important to set the `root` password during system installation or immediately thereafter?

   A. A system without a `root` password is comparatively easy to break into.

   B. The `root` password must be set for the system to mount the root filesystem.

   C. The `root` password must be set before regular user accounts can be defined.

   D. The `root` password must be set before regular user passwords can be set.

8. If your installation allows for the entry of dialup PPP account information, what type of information are you likely to be required to enter? (Choose all that apply.)

   A. One or more dialup telephone numbers

   B. The name of the ISP you use

   C. An account name and password

   D. Your IP address as assigned by the ISP

9. Why might you delay creating most user accounts until after installing the system?

   A. User accounts created during system installation are inherently insecure.

   B. Accounts created during installation are good for only one login before they're deleted.

   C. There are more options available for account creation using regular utilities than install-time utilities.

   D. You can use MD5 password storage only on accounts created after system installation.

10. What is a disadvantage of selecting packages by groups during installation?

    A. To later uninstall a package, you must uninstall the entire group.

    B. The packages included in package groups are often incomplete.

    C. Installing package groups often installs packages you don't want or need.

    D. Package groups cannot easily be upgraded at a later date.

11. What types of information are typically revealed by installation log files? (Choose all that apply.)

    A. The name of the computer's manufacturer

    B. The names of packages installed

    C. Comments you enter regarding specific packages

    D. Errors related to the installation of specific packages

12. Which of the following is the most useful information in locating an X driver for a video card?

    A. The interrupt used by the video card under Microsoft Windows

    B. Markings on the video card's main chip

    C. Whether the card uses the ISA, VLB, PCI, or AGP bus

    D. The name of the video card's manufacturer

13. Which of the following is true of XFree86 installation?

    A. You must rebuild XFree86 from source if you don't install it with the rest of the system.

    B. Most distributions can install XFree86 along with the rest of the system software.

    C. If you change your video card, you must reinstall all of XFree86.

    D. XFree86 3.3.6 and 4.0.*x* installations cannot coexist on one computer.

14. When you configure an X server, you need to make changes to configuration files and then start or restart the X server. Which of the following can help streamline this process?

    A. Shut down X by switching to a runlevel in which X doesn't run automatically, then reconfigure it and use `startx` to test X startup.

    B. Shut down X by booting into single-user mode, then reconfigure it and use `telinit` to start X running again.

    C. Reconfigure X, then unplug the computer to avoid the lengthy shutdown process before restarting the system, and X along with it.

    D. Use the `startx` utility to check the X configuration file for errors before restarting the X server.

15. Which of the following summarizes the organization of the `XF86Config` file?

A. The file contains multiple sections, one for each screen. Each section includes subsections for individual components (keyboard, video card, and so on).

B. Configuration options are entered in any order desired. Options relating to specific components (keyboard, video card, and so on) may be interspersed.

C. The file begins with a summary of individual screens. Configuration options are preceded by a code word indicating the screen to which they apply.

D. The file is broken into sections, one or more for each component (keyboard, video card, and so on). At the end of the file, there are one or more sections that define how to combine the main sections.

16. What is an advantage of a font server?

A. It provides faster font displays than is otherwise possible.

B. It can simplify font maintenance on a network with many X servers.

C. It's the only means of providing TrueType font support for XFree86 4.0.*x*.

D. It allows the computer to turn a bitmapped display into an ASCII text file.

17. A monitor's manual lists its range of acceptable synchronization values as 27–96kHz horizontal and 50–160Hz vertical. What implications does this have for the resolutions and refresh rates of which the monitor is capable?

A. The monitor can run at up to 160Hz vertical refresh rate in all resolutions.

B. The monitor can handle up to 160Hz vertical refresh rates depending upon the color depth.

C. The monitor can handle up to 160Hz vertical refresh rates depending upon the resolution.

D. The monitor can handle vertical resolutions of up to 600 lines (96,000 ÷ 160), but no more.

18. In what section of `XF86Config` do you specify the resolution that you want to run?

    A. In the `Screen` section, subsection `Display`, using the `Modes` option

    B. In the `Monitor` section, using the `Modeline` option

    C. In the `Device` section, using the `Modeline` option

    D. In the `DefaultResolution` section, using the `Define` option

19. Which major Linux desktop environment is a commercial package?

    A. KDE

    B. GNOME

    C. CDE

    D. XFce

20. Which of the following should be true when running programs from one desktop environment in another?

    A. You must install necessary support libraries (such as GTK+ libraries for GNOME programs) on the computer.

    B. You must configure the two environments with the same defaults for keyboard repeat rate, mouse tracking speed, and so on.

    C. You must configure the target program in its parent environment; thereafter, it can be run from other environments without problems.

    D. You must launch the program from the parent environment's file manager.

Answers to Review Questions

1. A. Most Linux distributions use installation programs written in Linux, not in DOS or Minix. The system usually boots from floppy or CD-ROM, although other boot media (such as hard disk or even network) are possible.

2. B. A bitmapped display, as used by a GUI installer, can be used to show graphical representations of the system's state that can't be done in a text-mode display. Text-based installers actually have an edge in hardware support because they can run on video cards that aren't supported by X.

3. C, D. The interface and number of buttons are important to mouse configuration. The form (mouse, trackball, touch pad, etc.) and underlying technology (optical, mechanical, etc.) are unimportant to Linux, and so they do not need to be specified.

4. A, B, C. You can usually define partitions using just about any tool that can create them, although with some tools (such as DOS's `FDISK`), you may need to change the partition type code using Linux tools. The DOS `FORMAT` utility is used to create a FAT filesystem, not define a partition.

5. D. Swap partitions aren't mounted in the way filesystems are, so they have no associated mount points.

6. B. As a hard disk ages, its magnetic properties degrade, resulting in sectors becoming unreliable. A bad block check looks for this condition to improve the reliability of data storage by avoiding afflicted sectors. When a disk begins to accumulate bad blocks, though, chances are other blocks will soon go bad, so the disk is better off being replaced.

7. A. If the `root` password isn't set, anybody who has physical access to the computer or who can log into a regular account can do anything they like to the computer's configuration. The `root` password is *not* required to mount the root filesystem, nor is it required to configure regular user accounts or passwords.

8. A, C. PPP dialup configuration requires the telephone number the computer will call, an account name, and a password. PPP does not require the ISP's name. Although PPP does permit static IP address assignment, this information is usually provided automatically by the ISP.

9. C. Regular account creation tools provide numerous options to customize and fine-tune the accounts they create. Most of these options are missing from the install-time account creation tools.

10. C. Package groups are groups of packages that fill some role on the system. You may need only some of the individual packages, so installing the entire group can waste disk space that might be better devoted to other purposes.

11. B, D. The installation log file typically summarizes the packages installed, including their names and any errors that were encountered when the packages were installed. Linux doesn't know or care about the name of the computer's manufacturer, and you don't enter comments on specific packages.

12. B. Markings on the video card's main chip typically include a name or number for the chipset; this is what you need in order to locate an X driver for the card. The video card's manufacturer name might or might not be useful information. If it proves to be useful, you'd also need a model number. The interrupt used by the video card in Windows is irrelevant. The card's bus can narrow the range of possibilities, but it isn't extremely helpful.

13. B. Linux installation routines normally include the ability to install and configure XFree86. If you fail to do this, you can usually install a pre-built binary version of the software; you don't need to rebuild from source code. The main XFree86 4.0.*x* package includes all drivers, so there's no need to reinstall it if you change a video card. Even 3.3.6 is broken into several packages, and some drivers work on multiple cards. 3.3.6 and 4.0.*x* servers can both exist on one computer, although normally only one runs at a time.

14. A. On most Linux systems, some runlevels don't run X by default, so using one of them along with the `startx` program (which starts X running) can be an effective way to quickly test changes to an X configuration. The `telinit` program changes runlevels, which is a lengthy process compared to using `startx`. Unplugging the computer to avoid the shutdown process is self-defeating since you'll have to suffer through a long startup (unless you use a new journaling filesystem), and it can also result in data loss. The `startx` utility doesn't check the veracity of an X configuration file; it starts X running from a text-mode login.

15. D. The `XF86Config` file design allows you to define variants or multiple components and easily combine or recombine them as necessary.

16. B. By maintaining fonts on one font server and pointing other X servers to that font server, you can reduce the administrative cost of maintaining the fonts on all the systems. Font servers do not produce faster font displays than X's local font handling; if anything, the opposite is true. XFree86 4.0 added native TrueType font support, but XFree86 3.3.6 and earlier didn't include it by default. Converting a bitmapped display into ASCII text is a function of optical character recognition (OCR) software, not a font server.

17. C. The vertical refresh rate range includes a maximum value, but that value may be reduced when the resolution and vertical refresh rate would demand a higher horizontal refresh rate than the monitor can handle. In practice, it's usually the horizontal limit that's most important, at least when running at typical resolutions. The color depth is irrelevant to this computation.

18. A. The `Modeline` option in the Monitor section defines *one* possible resolution, but there are usually several `Modeline` entries defining many resolutions. The `Modeline` option doesn't exist in the `Device` section, however, nor is that section where the resolution is set. There is no `DefaultResolution` section.

19. C. KDE, GNOME, and XFce are all covered under open source licenses. CDE is a commercial package that's common on commercial Unix systems, and which is also available for Linux.

20. A. Programs usually rely upon separate library files to provide necessary functions. KDE and GNOME are built upon different libraries (Qt and GTK+, respectively), so when running a program from one environment in another, you must ensure that the correct libraries are installed.

Chapter 3

Software Management

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.7 Identify strengths and weaknesses of different distributions and their packaging solutions (e.g., tar ball vs. RPM/DEB).**
- ✓ **1.9 Identify how the Linux kernel version numbering works.**
- ✓ **2.15 Explain when and why the kernel will need to be recompiled.**
- ✓ **2.16 Install boot loader (e.g., LILO, MBR vs. first sector of boot partition).**
- ✓ **2.17 Install and uninstall applications after installing the operating system (e.g., RPM, `tar`, `gzip`).**
- ✓ **2.19 Validate that an installed application is performing correctly in both a test and a production environment.**
- ✓ **3.10 Reconfigure boot loader (e.g., LILO).**
- ✓ **4.10 Use common shell commands and expressions.**
- ✓ **4.12 Create, extract and edit file and tape archives using `tar`.**
- ✓ **5.5 Download and install patches and updates (e.g., packages, `tgz`).**

Managing installed software involves a wide variety of tasks, many of which are specific to particular types of software or even specific packages. Other chapters cover some specific examples, such as X Window System configuration (covered in Chapter 2, "Installing Linux") or network client and server configuration (Chapter 5, "Networking"). This chapter covers the mechanics of package installation in general, using any of three common packaging schemes. Because of the importance of shell commands in package management and other day-to-day Linux administration, this chapter begins with a look at the use of shells.

One particular program deserves special attention in package installation, and that's the Linux kernel. Although it's sometimes possible to upgrade the kernel just as you would any other program, it's often beneficial to do it by recompiling the kernel from scratch. Even when you don't do this, you may need to modify the way the kernel is loaded, through a boot loader program. This chapter covers these issues as well.

Basic Command Shell Use

A *shell* is a program that allows you to interact with the computer by launching programs, manipulating files, and issuing commands. If you've read Chapter 2, this may sound a lot like the definition presented there of a desktop environment or file manager, and in some sense, these programs are graphical shells. The term *shell* is most often applied to text-based programs, though, and that's what's described in this section—text-based programs used to control a Linux system. Even if you prefer to use a GUI environment,

it's important that you understand basic shell use because the shell provides the user interface that's most consistent across distributions and other environments. You can also use text-based shells through text-mode network connections. Once you've started a shell, you can view and manipulate files and launch programs.

Starting a Shell

Linux supports many different shells, although precisely which ones might be installed varies from one distribution to another. The vast majority of Linux systems include bash, which is usually the default shell for new users. Another common shell is known as `tcsh`, and many others, such as `zsh`, `csh`, and `ash`, are also available. Most shells are similar in broad strokes, but some details differ.

There are many different ways to start a shell, most of which are at least partially automatic. The most common methods include the following:

Logging in at the text-mode console If you log into the computer using a text-mode console, you'll be greeted by your default shell, as it is set in your user account information (see Chapter 4, "Users and Security").

Logging in remotely Logging in remotely via Telnet, the Secure Shell (SSH; despite the name, SSH is not a shell in the sense discussed here, but it will start one automatically), or some other remote text-mode login tool will start a shell.

Starting an xterm An xterm is a GUI program in which text-based programs can run. By default, xterms usually start your default shell unless told to do otherwise. The section "Launching an Xterm" in Chapter 6, "Managing Files and Services," discusses xterms in more detail.

Explicitly launching a shell You can start one shell from within another. This can be helpful if you find you need features of one shell but are running another. Type the new shell's name to start it.

When you start a shell, you'll see a *command prompt*. This is one or more characters that indicate the shell is waiting for input. Although not universal, command prompts often include your username, the computer's hostname, or the directory in which the shell is operating. For instance, a command prompt might resemble the following:

```
[rodsmith@nessus /mnt]$
```

Although not a universal convention (it can be set in a user's shell configuration files), the final character is often a dollar sign ($) for ordinary users or a pound sign (#) for `root`. This serves as a visual indication of superuser status; you should be cautious when entering commands in a `root` shell, because it's easy to damage the system from such a shell. (Chapter 4 discusses `root` and its capabilities in more detail.)

This book includes command examples on separate lines. When the command is one that an ordinary user might issue, it's preceded by a $ prompt; when only root should be issuing the command, it's preceded by a # prompt. Because the username, computer name, and directory are usually unimportant, this information is omitted from the prompts printed in this book. The prompts are also omitted from command examples within a paragraph of text.

Viewing Files and Directories

When using a shell, it's often necessary to see the files in a given directory. This task is accomplished with the `ls` command, which displays the contents of either the current directory or the directory you name after the command. (If you list a file, it shows only that filename.) It's used like this:

```
$ ls /var
arpwatch  db   local  logcheck  opt       spool    www
cache     ftp  lock   mail      preserve  tmp      yp
catman    lib  log    nis       run       win4lin
```

By default, `ls` creates a listing that's sorted by filename. On some distributions, `ls` displays color-coded names so that you can easily identify directories and other important file types. The command accepts many parameters that can modify its default behavior. One of the most common of these is `-l`, which creates a long listing, like this:

```
$ ls -l t*
-rwxr-xr-x  1 rodsmith users      111 Apr 13 13:48 test
-rw-r--r--  1 rodsmith users   176322 Dec 16 09:34 thttpd-
↳2.20b-1.i686.rpm
-rw-r--r--  1 rodsmith users  1838045 Apr 24 18:52
↳tomsrtbt-1.7.269.tar.gz
-rw-r--r--  1 rodsmith users  3265021 Apr 22 23:46
↳tripwire-2.3.0-2mdk.i586.rpm
```

This output includes the permission strings, ownership, file sizes, and file creation dates in addition to the filenames. This example also illustrates the use of the * wildcard, which matches any string—thus, `t*` matches any file that begins with `t`.

Chapter 7, "Managing Partitions and Processes," discusses the use of `ls` in more detail.

To change to another directory, you should use the `cd` command. Type **cd** followed by the name of the directory to which you want to change, thus:

```
$ cd /tmp
```

You can specify the target directory name either in absolute form (starting with a / character), in which case the directory path is interpreted as being relative to the root directory; or in relative form (without the leading /), in which case it's relative to the current directory. (These rules also apply to specifying other filenames and directory names.)

If you want to view the contents of a file, you can do so in many different ways. You can load it into a text editor, for instance. Another possibility is to use either `more` or `less`. Both of these commands display a text file a page at a time, but `less` is the more sophisticated program.

Manipulating Files

If you need to move or rename a file, you should use the `mv` command. This command associates a new name (possibly in a different directory) with a file. To use it, issue the command followed by the current filename followed by a new filename or directory. (If you specify an existing directory without a filename, `mv` uses the current filename but moves the file to the specified directory.) For instance, take a look at the following:

```
$ mv /tmp/somefile.txt tempfile.txt
```

This command moves `somefile.txt` from the `/tmp` directory to the current directory and simultaneously renames it to `tempfile.txt`.

If you need to delete a file, you can do so with the `rm` command, which removes the file. List one or more files you want deleted after the command, thus:

```
$ rm tempfile.txt anotherfile.*
```

Unlike the trash can icons that are common in GUI environments, `rm` doesn't allow you to undo a deletion. Therefore, you should be very careful when using this command, particularly as `root`.

Chapter 7 includes expanded discussion of both the `mv` and `rm` commands.

Launching Programs

You can launch a program from a shell by typing its name. In fact, many shell "commands," including `ls` and `mv`, are actually programs that the shell runs. Most of these standard commands reside in the `/bin` directory, but shells search all directories specified by the `PATH` environment variable (discussed in more detail in Chapter 6) for commands to run. If you type the name of a program that resides in any directory on the path, the shell runs that program. You can also pass *parameters* to a program—optional information that the program can use in a program-specific way. For instance, the names of the files to be manipulated are parameters to commands like `mv` and `rm`. Many programs accept parameters that are preceded by one or two dashes and a code, as in `-r` or `-t` *`time`*. Most parameters are case-sensitive; in fact, many programs use upper- and lowercase versions of a parameter in different ways.

Most text-based programs take over the display (the text-mode login, Telnet session, xterm, or what have you). Many show little or no information before returning control to the shell, so you don't really notice this fact. Some programs, such as text-mode editors, truly control the display; they may clear all the information that has previously appeared and fill the display with their own information. Other programs may not clear the screen entirely, or even display their own information, but they may take a long time to operate. In some cases, you may want to retain control of your shell while the program does its thing in the background. To do this, follow the command with an ampersand (`&`). When you do this, the program you launch will still be attached to the display from which it was launched, but it shares that display with the shell. This works well for noninteractive programs but very poorly for interactive tools. For instance, suppose you have a program called `supercrunch` that performs some lengthy computation but requires no interaction from the user. You could launch it like this:

```
$ supercrunch &
```

If `supercrunch` produces text-based output, it will appear on the screen, but you'll still be able to use the shell for other purposes. If you've already launched a program and want to move it into the background, press Ctrl+Z. This suspends the currently running program and returns you to the shell. At this point, the program you've suspended will *not* be doing any work. This may be fine for a text editor you wanted to momentarily suspend or the like, but if the program was performing computations that must continue, you must take additional steps to see that this happens. You can type **`fg`** to return to the suspended program, or **`bg`** to start it running again in the background. The latter is much like appending an ampersand to the command name when you launched it.

If you try to launch an X-based program, you must be running the shell in an xterm, or possibly in some other way that allows X programs to run, such as from another computer with its own X server and all appropriate environment variables set to permit remote X program operation, as discussed in Chapter 5. If you try to launch an X program from a text-only login, you'll receive an error message along the lines of `Can't open display`.

Although X-based programs don't normally produce text output, they do take over the terminal from which they were launched. If you want to continue to use an xterm after launching an X-based program, follow its name with an ampersand (&), as just described.

Shell Shortcuts

Linux shells permit some important operational shortcuts. One of the most useful of these is the use of the Tab key for *filename completion*. Suppose you want to move a file that's called `shareholder-report-for-2001.txt`. You could type the entire filename, but that can become quite tedious. Most Linux shells, including the popular bash shell, support a feature in which hitting the Tab key completes an incomplete command or filename, as long as you've typed enough characters to uniquely define the file or command. For instance, suppose that `ls` reveals two files in a directory, thus:

```
$ ls
share-price-in-2001.txt  shareholder-report-for-2001.txt
```

If you want to move the second file to another directory, you could type **mv shareh**, then the Tab key. The shell will complete the filename, so your command line will include the entire name. You can then type the destination directory for the file (possibly using filename completion to enter it more efficiently, as well).

What happens when the characters you enter are *not* unique? In this case, the shell completes as much of the job as it can. For instance, if you type **mv sh** and then hit the Tab key, bash fills out the next three characters, so that the command line reads **mv share**. Some configurations also display the possible completions at this point. (For those that don't, pressing Tab again usually displays these completions.) If you then type either **h** or **-** and press Tab again, bash completes the filename.

Another shortcut is the use of the up and down arrow keys to scroll through previous commands. If you need to type two similar commands in a row, you can type one, then hit the up arrow key to retrieve the previous command. You can then use the left arrow or Backspace keys to move back in the line to edit it (Backspace deletes characters, but the left arrow key doesn't). You can go back through several commands in this way, and if you overshoot, you can use the down arrow key to retrieve more recent commands.

These shortcuts, and other basic shell commands for that matter, are extremely helpful in working with packages and other files. You can perform many tasks with a file manager, of course, but text-based utilities like the `rpm` and `dpkg` utilities discussed in this chapter were designed to be used from shells. Because package filenames are frequently very long, using filename completion can be particularly helpful with them.

Although not a shortcut in the same sense as using the Tab key, one particularly important tool is the Linux man page system. The `man` program (short for "manual") contains usage information on many Linux commands and files. Type **man** followed by the command name to learn more about the command, as in **man mv**. Linux man pages are usually written in a very succinct style; they aren't intended as complete documentation, but rather as a reference aid.

Package Concepts

Any OS is defined largely by the files it installs on the computer. In the case of Linux, these files include the Linux kernel; critical utilities stored in directories like `/bin`, `/sbin`, `/usr/bin`, and `/usr/sbin`; and configuration files stored in `/etc`. How those files came to reside in their locations is irrelevant to the identity of the computer as a Linux box, but this detail is critically important to the day-to-day duties of a system administrator. When an updated version of a program is released, it's extremely helpful to be able to track down the installed version of the program, determine just what version the installed program is, and update all the necessary files. A failure to do all of this can leave a system with two copies of a program or its support files, which can result in confusion. It's also important that when you install a new program, you not accidentally overwrite files that belong to another program.

In order to keep track of installed programs, documentation, and so on, various package maintenance utilities have emerged. Some of these, such as RPM and Debian package tools, are tightly woven into various Linux distributions, thus providing a centralized mechanism for program updates.

File Collections

Most programs today consist of several files. Many programs come with one or more documentation files, configuration files, and support programs. For this reason, it's long been common practice, on all platforms, to package related files together in one carrier file. This carrier file typically uses compression to save disk space and download time, and it may include information on the placement of specific files once they're extracted and installed on the computer.

Linux package file formats all provide these useful features. A single package file may contain a single program file or dozens (even hundreds or thousands) of files. A complete Linux distribution, in turn, consists of hundreds of package files, all designed to coexist and even work together to provide the features associated with Linux.

In addition to providing a common carrier mechanism for package transport, the RPM and Debian package systems provide a means of recording additional information about the package. This information includes a version number, a build number, the name of the package maintainer, the date and time of the package's last compilation, the hostname of the computer

that built the package, one or more descriptions of the package, and a few other miscellaneous pieces of information. Typically, you can access all of this information either before or after installing a package on the computer, which can be quite helpful—you can read the package description to determine whether it's really what you want to install, before you do so.

The Installed File Database

One of the problems with a simple file collection mechanism is that there's no way to track what files you've installed, what files are associated with other files, and so on. It's easy for a system using such a simple package mechanism to fall into chaos or collect detritus. A partial solution to these problems is to maintain a centralized database of installed files. Both the RPM and Debian systems provide this feature. With RPM, the database is stored in the `/var/lib/rpm` directory; for Debian packages, the database is in `/var/lib/dpkg`. These directories actually contain several files, each of which tracks a different type of information. Tarballs, however, don't support a package management database.

Tarballs are file collections created by the `tar` utility program. Although they lack some of the features of RPM and Debian packages, they're more universally compatible, and they're easier to create than are RPM or Debian packages. Chapter 5 covers tarballs, RPMs, and Debian packages in more detail.

Most people don't need to understand the details of how the installed file database works; this information is most useful to those who write the tools or need to recover a seriously corrupted system. What is important are the features that the database provides to a Linux system, which include those listed here:

Package information The supplementary information associated with a package—build date, description, version number, and so on—is copied from the package file to the installed file database when you install the package. This allows you to retrieve this information even if you delete the original package file.

File information The database includes information on all of the files installed on the computer via the package system. This information includes the name of the package to which the file belongs so that you can

track a file back to its owner. There's also a checksum value and information on file ownership and permissions, which make it possible to detect when a file has been altered—assuming the database hasn't been altered as well. This file information does *not* extend to any files users create or even to non-standard configuration files for some packages. Standard configuration files are typically tracked, however.

Dependencies A *dependency* is a reliance of one package upon another. For instance, many programs rely upon libc. Packages include information on the files or packages upon which they depend. This feature allows the package management system to detect these dependencies and block installation of a package if its dependencies are unmet. The system can also block the removal of a package if others depend upon it.

Provision information Some packages provide features that are used by other packages. For instance, a mail client may rely upon a mail server, and various different mail servers exist for Linux. In this case, a simple file or package dependency can't be used because more than one mail server can be used to fulfill the client's requirements. Nonetheless, this feature is essentially a type of dependency.

Whenever you install, remove, or modify a package through a package management system, that system updates its database to reflect the changes you've made. You can then query the database about your installed packages, and the system can use the database when you subsequently modify your installation. In this way, the system can head off trouble—for instance, it can warn you and abort installation of a package if that package contains files that would overwrite files belonging to another package.

The package database does not include information on files or packages installed in any way but through the package management system. For this reason, it's best not to mix different types of packages. Although it's possible to install both RPM and Debian package management systems on one computer, their databases remain separate, thus cutting the benefits of conflict tracking, dependencies, and so on. For instance, you might install an important library in Debian format, but RPM packages that rely on that library won't know the library is installed, and so they will not install unless you provide an override switch. Further, you may not be warned that other programs require the library when you remove or upgrade it, so you might inadvertently break the RPM packages.

Some programs are distributed only in tarball form. In such cases, you can attempt to build an RPM or Debian package from the tarball or install from the tarball without the benefit of a package management system. Although the latter option has the drawbacks just outlined, it's often simpler than trying to create an RPM or Debian package. If you only install a few such programs, chances are you won't have too much trouble, especially if you keep good records on what you're installing from tarballs. Typically, programs you compile from source code go in the `/usr/local` directory tree, which isn't used by most RPM or Debian packages. This fact helps keep the two program types isolated, further reducing the chance of trouble.

If you have a binary tarball package, you may be able to convert it to RPM or Debian format using the `alien` utility, described in the upcoming "Converting between Package Formats" section.

Rebuilding Packages

One of the features of package systems is that they allow you to either install a *binary package* (sometimes referred to as a precompiled package) or recompile a *source package* on your own system. The former approach is usually simpler and less time-consuming, but the latter approach has its advantages, too. Specifically, it's possible to customize a program when you recompile it from source code. This can include both changes to the program source code and compile-time customizations (such as compiling a package on an unusual architecture). Recompilation is possible both with the sophisticated RPM and Debian systems and with simpler tarballs—in fact, the primary means of source code distribution is usually as a tarball.

If you find a tarball for a package that is not available in other forms, you have two basic choices: You can compile or install the software as per the instructions in the tarball, which bypasses your RPM or Debian database if your distribution uses one; or you can create an RPM or Debian package from the original source code, and install the resulting binary package. The former approach is usually simpler when you want to install the package on just one system, despite the drawback of losing package database information. The latter approach is superior if you need to install the package on many similar systems, but it takes more effort—you must create special files to control the creation of a final RPM or Debian package, and then use special commands to create that package. (Using the `alien` utility to convert a binary tarball is an exception, though.)

These actions are beyond the scope of this book. Consult the documentation for the package system for more information. In particular, the RPM HOWTO (`http://www.linuxdoc.org/HOWTO/RPM-HOWTO`) contains this information for RPM. The book *Maximum RPM* by Ed Bailey (Red Hat Press, 1997) may also be useful for those who need to delve deeply into the RPM system.

Source code is available in formats other than tarballs. Today, many program authors take the time to create *source RPMs*, which are source code packages meant to be processed by the RPM tools. It's also possible to create equivalent Debian packages, but these are most commonly found on sites catering specifically to Debian-based systems. A source RPM is easy to compile into a binary RPM for any given computer; all you need to do is call the `rpm` program with the `--rebuild` argument and the name of the source package. (Sometimes additional arguments are needed, as when you are cross-compiling for one platform on another.) This recompilation may take anywhere from a few seconds to several minutes, or conceivably hours for large packages on slow computers. The result is one or more binary RPMs in the `/usr/src/redhat/RPMS/i386` directory or someplace similar (`redhat` may be something else on non-Red Hat distributions, and `i386` may be something else on non-*x*86 platforms or on distributions that optimize for Pentium or later CPUs).

However you do it, recompiling programs from source code has several advantages and disadvantages compared to using a ready-made binary package. One of the primary advantages is that you can control various compilation options, and you can even modify the source code to fix bugs or customize the program for your particular needs. Making such changes is much easier when you start with a tarball than when you start with an RPM or Debian source package, however. Another advantage is that you can compile a program for an unusual distribution. You might not be able to find a package of a particular program for Alpha or PowerPC architectures, for instance, but if a source package is available, you can compile it yourself. Similarly, if you compile a package yourself, you can work around some library incompatibilities you might encounter with pre-built binaries, particularly if the binaries were created on a distribution other than the one you use.

The primary drawback to compiling your own packages is that it takes time. This problem is exacerbated if you need to install additional development libraries, compilers, or other tools in order to make a package compile.

(Many programs need particular utilities to compile but not to run.) Sometimes a source package needs particular versions of other programs to compile, but you may have an incompatible version, making compilation impossible until you change the version you've got. New Linux users also often have troubles with recompiling because of unfamiliarity with the procedures.

Installing and Removing Packages

The three most common package formats in Linux are *Red Hat Package Manager (RPM)* packages, *Debian packages*, and *tarballs* (files collected together using the `tar` program). Of these three, tarballs are the most primitive, but they are also the most widely supported. Most distributions use RPMs or Debian packages as the basis for most installed files. Therefore, it's important to understand how to use at least one of these two formats for most distributions, as well as tarballs.

RPM Packages

The most popular package manager in the Linux world is RPM. In fact, RPM is available on non-Linux platforms, although it sees less use outside of the Linux world. The RPM system provides all the basic tools described in the earlier section, "Package Concepts," such as a package database that allows for checking conflicts and ownership of particular files.

Distributions That Use RPM

As RPM's full name (Red Hat Package Manager) implies, RPM was developed by Red Hat for its own distribution. Red Hat released the software under the General Public License (GPL), however, so others have been free to use it in their own distributions, and in fact, this is precisely what has happened. Some distributions, such as Mandrake, TurboLinux, LinuxPPC, and Yellow Dog, are based on Red Hat, and so they use RPMs as well as many other parts of the Red Hat distribution. Others, such as SuSE and Caldera, borrow less from the Red Hat template, but they do use RPMs. Of course, all Linux distributions share many common components, so even those that weren't originally based on Red Hat are very similar to it in many ways other

than just their use of RPM packages. On the other hand, distributions that were originally based on Red Hat have diverged from it over time. As a result, the group of RPM-using distributions shows substantial variability, but all of them are still Linux distributions that provide the same basic tools, such as the Linux kernel, common shells, XFree86, and so on.

RPM is a cross-platform tool. As noted earlier, some non-Linux Unix systems can use RPM, although most don't use it as their primary package distribution system. RPM supports any CPU architecture. In fact, Red Hat Linux is or has been available for at least three CPUs: *x*86, Alpha, and SPARC. Among the distributions mentioned earlier, LinuxPPC and Yellow Dog are PowerPC distributions (they run on Apple PowerMacs and some non-Apple systems), and SuSE is available on *x*86, PowerPC, and Alpha systems. For the most part, source RPMs are transportable across architectures—you can use the same source RPM to build packages for *x*86, PowerPC, Alpha, SPARC, or any other platform you like. Some programs are actually composed of architecture-independent scripts, and so they need no recompilation. There are also documentation and configuration packages that work on any CPU.

The convention for naming RPM packages is as follows:

packagename`-`*a.b.c*`-`*x.arch*`.rpm`

Each of the filename components has a specific meaning:

packagename This is the name of the package, such as `samba` for the Samba file and print server.

a.b.c This is the package version number, such as 2.0.7. The version number doesn't have to be three period-separated numbers, but that's the most common form. The program author assigns the version number.

x The number following the version number is the *build number* (aka the *release number*). This number represents minor changes made by the package maintainer, not by the program author. These changes may represent altered startup scripts or configuration files, changed file locations, added documentation, or patches appended to the original program to fix bugs or to make the program more compatible with the target Linux distribution. Some distribution maintainers add a letter code to the build number to distinguish their packages from those of others. Note that these numbers are *not* comparable across package maintainers—George's build number 5 of a package is *not* necessarily an improvement on Susan's build number 4 of the same package.

arch The final component preceding the `.rpm` extension is a code for the package's architecture. `i386` is the most common architecture code; it represents a file compiled for any *x*86 CPU from the 80386 onward. Some packages include optimizations for Pentiums or above (`i586` or `i686`), and non-*x*86 binary packages use codes for their CPUs, such as `ppc` for PowerPC CPUs. Scripts, documentation, and other CPU-independent packages generally use the `noarch` architecture code. The main exception to this rule is source RPMs, which use the `src` architecture code.

For instance, the Linux Mandrake 7.2 distribution ships with a Samba package called `samba-2.0.7-18mdk.i586.rpm`, indicating that this is build 18 from Mandrake of Samba 2.0.7, compiled with Pentium optimizations. These naming conventions are just that, though—conventions. It's possible to rename a package however you like, and it will still install and work. The information in the filename is retained within the package. This fact can be useful if you're ever forced to transfer RPMs using a medium that doesn't allow for long filenames. In fact, the SuSE distribution eschews long filenames, preferring short filenames such as `samba.rpm`.

In an ideal world, any RPM package will install and run on any RPM-based distribution that uses an appropriate CPU type. Unfortunately, there are compatibility issues that can crop up from time to time. These include the following:

- Distributions may use different versions of the RPM utilities, as discussed shortly. This problem can completely prevent an RPM from one distribution being used on another.
- An RPM package designed for one distribution may have dependencies that are unmet in another distribution. A package may require a newer version of a library than is present on the distribution you're using, for instance. This problem can usually be overcome by installing or upgrading the depended-upon package, but sometimes this causes problems because the upgrade may break other packages. By rebuilding the package you want to install from a source RPM, you can often work around these problems, but sometimes the underlying source code also needs the upgraded libraries.
- An RPM package may be built to depend upon a package of a particular name, such as `samba-client` depending upon `samba-common`, but if the distribution you're using has named the package differently, the `rpm` utility will object. You can override this objection by using the

`--nodeps` switch, but sometimes the package won't work once installed. Rebuilding from a source RPM may or may not fix this problem.

- Even when a dependency appears to be met, different distributions may include slightly different files in their packages. For this reason, a package meant for one distribution may not run correctly when installed on another distribution. Sometimes installing an additional package will fix this problem.
- Some programs include distribution-specific scripts or configuration files. This problem is particularly acute for servers, which may include startup scripts that go in `/etc/rc.d/init.d` or elsewhere. Overcoming this problem usually requires that you remove the offending script after installing the RPM and either start the server in some other way or write a new startup script, perhaps modeled after one that came with some other server for your distribution.

Despite this list of caveats, mixing and matching RPMs from different distributions usually works reasonably well for most programs, particularly if the distributions are closely related or you rebuild from a source RPM. If you have trouble with an RPM, though, you may do well to try to find an equivalent package that was built with your distribution in mind.

Upgrades to RPM

The earliest versions of RPM were quite primitive by today's standards; for instance, they did not support dependencies. Over time, though, improvements have been made. This fact occasionally causes problems when Red Hat releases a new version of RPM. For instance, Red Hat 7.0 uses version 4 of the RPM utilities, but version 4 RPM files cannot be installed with most earlier versions of RPM. This led to frustration on the part of many people who used RPM-based distributions in late 2000 because they couldn't use Red Hat 7.0 RPMs on their systems.

It's usually possible to overcome such problems by installing a newer version of RPM and upgrading the RPM database. Unfortunately, there's a chicken-and-egg problem, because without the new version of RPM, it's impossible to install the updated version of RPM. Red Hat and many other RPM-based distribution providers frequently do make a version of the next-generation version of RPM available for older systems. In the case of the

switch to RPM 4.0 with Red Hat 7.0, Red Hat has made this upgrade available in their Red Hat 6.2 updates area, for instance. After installing such an upgrade, be sure to type **`rpm --rebuilddb`** to have the system rebuild your RPM database to conform to the new program's expectations. If you fail to do this, you may be unable to install new programs or access information on old ones.

The *rpm* Command Set

The main RPM utility program is known as `rpm`. Use this program to install or upgrade a package at the shell prompt. `rpm` has the following syntax:

`rpm [`*`operation`*`][`*`options`*`] [`*`package-files`*`|`*`package-names`*`]`

Table 3.1 summarizes the most common `rpm` operations, and Table 3.2 summarizes the most important options. Be aware, however, that `rpm` is a very complex tool, so this listing is necessarily incomplete. Tables 3.1 and 3.2 do include information on the most common `rpm` features, however. For information on operations and options more obscure than those listed in Tables 3.1 and 3.2, see the `rpm` man pages. Many of `rpm`'s less-used features are devoted to the creation of RPM packages by software developers.

TABLE 3.1 Common `rpm` Operations

`rpm` Operation	Description
`-i`	Installs a package; system must *not* contain a package of the same name
`-U`	Installs a new package or upgrades an existing one
`-F` or `--freshen`	Upgrades a package only if an earlier version already exists
`-q`	Queries a package—finds if a package is installed, what files it contains, and so on
`-V` or `-y` or `--verify`	Verifies a package—checks that its files are present and unchanged since installation
`-e`	Uninstalls a package

TABLE 3.1 Common `rpm` Operations *(continued)*

`rpm` Operation	Description
`-b`	Builds a binary package, given source code and configuration files
`--rebuild`	Builds a binary package, given a source RPM file
`--rebuilddb`	Rebuilds the RPM database to fix errors

TABLE 3.2 Common `rpm` Options

`rpm` Option	Used with Operations	Description
`--root` *`dir`*	Any	Modifies the Linux system having a root directory located at *`dir`*. This option can be used to maintain one Linux installation discrete from another one (say, during OS installation or emergency maintenance).
`--force`	`-i`, `-U`, `-F`	Forces installation of a package even when it means overwriting existing files or packages.
`-h` or `--hash`	`-i`, `-U`, `-F`	Displays a series of pound signs (#) to indicate the progress of the operation.
`-v`	`-i`, `-U`, `-F`	Used in conjunction with the `-h` option to produce a uniform number of hash marks for each package.

TABLE 3.2 Common `rpm` Options *(continued)*

`rpm` Option	Used with Operations	Description
`--nodeps`	`-i`, `-U`, `-F`, `-e`	Performs no dependency checks. Installs or removes the package even if it relies on a package or file that's not present or is required by a package that's not being uninstalled.
`--test`	`-i`, `-U`, `-F`	Checks for dependencies, conflicts, and other problems without actually installing the package.
`--prefix` *`path`*	`-i`, `-U`, `-F`	Sets the installation directory to *`path`* (works only for some packages).
`-a` or `--all`	`-q`, `-V`	Queries or verifies all packages.
`-f` *`file`* or `--file` *`file`*	`-q`, `-V`	Queries or verifies the package that owns *`file`*.
`-p` *`package-file`*	`-q`	Queries the uninstalled RPM *`package-file`*.
`-i`	`-q`	Displays package information, including the package maintainer, a short description, and so on.
`-R` or `--requires`	`-q`	Displays the packages and files upon which this one depends.
`-l` or `--list`	`-q`	Displays the files contained in the package.

To use `rpm`, you combine one operation with one or more options. In most cases, you include one or more package names or package filenames, as well. (A package filename is a complete filename, but a package name is a shortened version. For instance, a package filename might be `samba-2.0.7-18mdk.i586.rpm`, while the matching package name is `samba`.) You can either issue the `rpm` command once for each package, or you can list multiple packages, separated by spaces, on the command line. The latter is often preferable when you're installing or removing several packages, some of which depend upon others in the group. Issuing separate commands in this situation requires that you install the depended-upon package first or remove it last, whereas issuing a single command allows you to list the packages on the command line in any order.

Some operations require that you give a package filename, and others require a package name. In particular, `-i`, `-U`, `-F`, and the rebuild operations require package filenames. `-q`, `-V`, and `-e` normally take a package name, although the `-p` option can modify a query (`-q`) operation to work on a package filename.

When installing or upgrading a package, the `-U` operation is generally the most useful because it allows you to install the package without manually uninstalling the old one. This one-step operation is particularly helpful when packages contain many dependencies because `rpm` detects these and can perform the operation should the new package fulfill the dependencies provided by the old one.

To use `rpm` to install or upgrade a package, issue a command similar to the following:

```
# rpm -Uvh samba-2.0.7-18mdk.i586.rpm
```

You could also use **`rpm -ivh`** in place of **`rpm -Uvh`** if you don't already have a `samba` package installed.

It's possible to distribute the same program under different names. In this situation, upgrading may fail, or it may produce a duplicate installation, which can yield bizarre program-specific malfunctions. Red Hat has described a formal system for package naming to avoid such problems, but they still occur occasionally. Therefore, it's best to upgrade a package using a subsequent release provided by the same individual or organization that provided the original.

Verify that the package is installed with the **`rpm -qi`** command, which displays information such as when and on what computer the binary package was built. Listing 3.1 demonstrates this command. (`rpm -qi` also displays an extended plain-English summary of what the package is, which has been omitted from Listing 3.1.)

Listing 3.1: RPM Query Output

```
$ rpm -qi samba
Name        : samba                         Relocations: (not
↳relocateable)
Version     : 2.0.7                              Vendor:
↳MandrakeSoft
Release     : 18mdk                          Build Date: Mon
↳16 Oct 2000 09:54:0 6 AM EDT
Install date: Sat 04 Nov 2000 11:36:09 AM EST      Build
↳Host: no.mandrakesoft.com
Group       : System/Servers                 Source RPM:
↳samba-2.0.7-18mdk.src.rpm
Size        : 10429317                          License: GPL
Packager    : Till Kamppeter <till@mandrakesoft.com>
Summary     : Samba SMB server.
```

RPM Compared to Other Package Formats

RPM is a very flexible package management system. In most respects, it's comparable to Debian's package manager, and it offers many more features than tarballs do. When compared to Debian packages, the greatest strength of RPMs is probably their ubiquity. Many software packages are available in RPM form from their developers and/or from distribution maintainers.

Distribution packagers frequently modify the original programs in order to make them integrate more smoothly into the distribution as a whole. For instance, distribution-specific startup scripts may be added, program binaries may be relocated from default `/usr/local` subdirectories, and program source code may be patched to fix bugs or add features. Although these changes can be useful, you may not want them, particularly if you're using a program on another distribution.

The fact that there are so many RPM-based distributions can also be a boon. You may be able to use an RPM intended for one distribution on another, although as noted earlier, this isn't certain. In fact, this advantage can turn into a drawback if you try to mix and match too much—you can wind up with a mishmash of conflicting packages that can be very difficult to disentangle.

The RPMFind Web site, `http://rpmfind.net/linux/RPM`, is an extremely useful resource when you want to find an RPM of a specific program. This site includes links to RPMs built by programs' authors, specific distributions' RPMs, and those built by third parties.

Compared to tarballs, RPMs offer much more sophisticated package management tools. This can be important when upgrading or removing packages and also for verifying the integrity of installed packages. On the other hand, although RPMs are very common in the Linux world, they're less common on other platforms. Therefore, you're more likely to find tarballs of generic Unix source code, and tarballs are preferred if you've written a program that you intend to distribute for other platforms.

Debian Packages

In their overall features, Debian packages are similar to RPMs, but the details of operation for each differ, and Debian packages are used on different distributions than are RPMs. Because each system uses its own database format, RPMs and Debian packages aren't interchangeable, although conversion between the formats is possible. (This process is described in one of the upcoming sections, "Converting between Package Formats.")

Distributions That Use Debian Packages

As the name implies, Debian packages originated with the Debian distribution. Since that time, the format has been adopted by several other distributions, including Storm, Corel, and Libranet. All three of these are derived from the original Debian, which means that packages from the original Debian are likely to work well on other Debian-based systems. Although Debian doesn't emphasize flashy GUI installation or configuration tools, its derivatives—particularly Corel and Storm—add GUI configuration tools to the

base Debian system, which makes these distributions more appealing to Linux novices. The original Debian favors a system that's as bug-free as possible, and it tries to adhere strictly to open source software principles, rather than invest effort in GUI configuration tools. The original Debian is unusual in that it's maintained by volunteers who are motivated by the desire to build a product they want to use, rather than by a company that is motivated by profits.

Like RPM, the Debian package format is neutral with respect to both OS and CPU type. Debian packages are extremely rare outside of Linux, although there are efforts underway to create a Debian distribution that uses the GNU Hurd kernel rather than the Linux kernel. Such a distribution would not be Linux but would closely resemble Debian GNU/Linux in operation and configuration.

The original Debian distribution has been ported to many different CPUs, including *x*86, PowerPC, Alpha, 680*x*0, MIPS, and SPARC. *x*86 was the original architecture, and subsequent ports exist at varying levels of maturity. Derivative distributions generally work only on *x*86 systems, but this could change in the future.

Debian packages follow a naming convention similar to those for RPMs, but Debian packages generally don't include codes in the filename to specify a package's architecture—there is no `i386`, for instance, to specify an *x*86 binary. Some Debian-based distributions, however, do use these codes. In Corel Linux 1.1 and later, for instance, a filename ending in `i386.deb` indicates an *x*86 binary, and `all.deb` indicates a CPU-independent package, such as documentation or scripts. As with RPM files, this file-naming convention is only that—a convention. You can rename a file as you see fit, either to include or omit the processor code. There is no code for Debian source packages because, as described shortly, these actually consist of several separate files.

The *dpkg* Command Set

Debian packages are incompatible with RPM packages, but the basic principles of operation are the same across both package types. Like RPMs, Debian packages include dependency information, and the Debian package utilities maintain a database of installed packages, files, and so on. You use the `dpkg` command to install a Debian package. This command's syntax is similar to that of `rpm`:

```
dpkg [options][action] [package-files|package-name]
```

The *action* is the action to be taken; common actions are summarized in Table 3.3. The options (Table 3.4) modify the behavior of the action, much like the options to `rpm`.

TABLE 3.3 dpkg Primary Actions

dpkg Action	Description
`-i` or `--install`	Installs a package
`--configure`	Reconfigures an installed package: runs the postinstallation script to set site-specific options
`-r` or `--remove`	Removes a package, but leaves configuration files intact
`-P` or `--purge`	Removes a package, including configuration files
`-p` or `--print-avail`	Displays information about an installed package
`-I` or `--info`	Displays information about an uninstalled package file
`-l` *pattern* or `--list` *pattern*	Lists all installed packages whose names match *pattern*
`-L` or `--listfiles`	Lists the installed files associated with a package
`-S` *pattern* or `--search` *pattern*	Locates the package(s) that own the file(s) specified by *pattern*
`-C or --audit`	Searches for partially installed packages and suggests what to do with them

TABLE 3.4 Options to Fine-Tune dpkg Actions

dpkg Option	Used with Actions	Description
`--root=`*`dir`*	All	Modifies the Linux system using a root directory located at *dir*. Can be used to maintain one Linux installation discrete from another one, say during OS installation or emergency maintenance.
`-B` or `--auto-deconfigure`	`-r`	Disables packages that rely upon one that is being removed.
`--force-`*`things`*	Assorted	Forces specific actions to be taken. Consult the `dpkg` man page for details of *`things`* this option does.
`--ignore-depends=`*`package`*	`-i`, `-r`	Ignores dependency information for the specified package.
`--no-act`	`-i`, `-r`	Checks for dependencies, conflicts, and other problems without actually installing or removing the package.
`--recursive`	`-i`	Installs all packages that match the package name wildcard in the specified directory and all subdirectories.
`-G`	`-i`	Doesn't install the package if a newer version of the same package is already installed.

TABLE 3.4 Options to Fine-Tune dpkg Actions *(continued)*

dpkg Option	Used with Actions	Description
-E or --skip-same-version	-i	Doesn't install the package if the same version of the package is already installed.

As with rpm, dpkg expects a package name in some cases and a package filename in others. Specifically, --install (-i) and --info (-I) both require the package filename, but the other commands take the shorter package name.

As an example, consider the following command, which installs the samba-common_2.0.7-3.deb package:

```
# dpkg -i samba-common_2.0.7-3.deb
```

If you're upgrading a package, you may need to remove an old package before installing the new one. To do this, use the -r option to dpkg, as in

```
# dpkg -r samba
```

To find information on an installed package, use the -p parameter to dpkg, as shown in Listing 3.2. This listing omits an extended English description of what the package does.

Listing 3.2: dpkg Package Information Query Output

```
$ dpkg -p samba-common_2.0.7-3.deb
Package: samba-common
Priority: optional
Section: net
Installed-Size: 3206
Maintainer: Eloy A. Paris <peloy@debian.org>
Architecture: i386
Source: samba
Version: 2.0.7-3
Replaces: samba (<= 2.0.5a-2)
Depends: libpam-modules, libc6 (>= 2.1.2), libncurses5,
↳libpam0g, libreadline4 (>= 4.1)
Filename: dists/potato/main/binary-i386/net/
↳samba-common_2.0.7-3.deb
Size: 569658
MD5sum: 12212667de0fd35aeb656624af2ac7d2
```

Debian-based systems often use a somewhat higher-level utility called `dselect` to handle package installation and removal. `dselect` provides a text-mode list of installed packages and packages available from a specified source (such as a CD-ROM drive or an FTP site), and it allows you to select which packages you want to install and remove. This interface can be very useful when you want to install several packages, but `dpkg` is often more convenient when manipulating just one or two packages. Because `dpkg` can take package filenames as input, it's also the preferred method of installing a package that you download from an unusual source or create yourself.

Using *apt-get*

Another option for Debian package management is the `apt` utilities, and particularly `apt-get`. This tool allows you to perform easy upgrades of packages, especially if you have a fast Internet connection. Debian-based systems include a file, typically `/etc/apt/sources.list`, that specifies locations from which important packages can be obtained. If you installed the OS from a CD-ROM drive, this file will initially list directories on the installation CD-ROM in which packages can be found. There are also likely to be a few lines near the top, commented out with pound signs (#), indicating directories on an FTP or Web site from which you can obtain updated packages. (These lines may be uncommented if you did a network install initially.)

Don't add a site to `/etc/apt/sources.list` unless you're sure it can be trusted. `apt-get` does automatic and semi-automatic upgrades, so if you add a network source to `sources.list` and that source contains unreliable programs or programs with security holes, your system will become vulnerable after upgrading via `apt-get`.

`apt-get` works by obtaining information on available packages from the sources listed in `/etc/apt/sources.list` and then using that information to upgrade or install packages. The syntax is similar to that of `dpkg`:

`apt-get [`*`options`*`][`*`command`*`] [`*`package-names`*`]`

Table 3.5 lists the `apt-get` commands, and Table 3.6 lists the most commonly used options. In most cases, you won't actually use *any* options with `apt-get`, just a single command and possibly one or more package names. One particularly common use of this utility is to keep your system up-to-date with any new packages. The following two commands will accomplish this

goal, if /etc/apt/sources.list includes pointers to up-to-date file archive FTP sites:

```
# apt-get update
# apt-get dist-upgrade
```

TABLE 3.5 apt-get Commands

apt-get Command	Description
update	Obtains updated information on packages available from the installation sources listed in /etc/apt/sources.list.
upgrade	Upgrades all installed packages to the newest versions available, based on locally stored information on available packages.
dselect-upgrade	Performs any changes in package status (installation, removal, etc.) left undone after running dselect.
dist-upgrade	Similar to upgrade, but performs "smart" conflict resolution to avoid upgrading a package if that would break a dependency.
install	Installs a package by package name (not by package filename), obtaining the package from the source that contains the most up-to-date version.
remove	Removes a specified package by package name.
source	Retrieves the newest available source package file by package filename, using information on available packages and installation archives listed in /etc/apt/sources.list.
check	Checks the package database for consistency and broken package installations.
clean	Performs housekeeping to help clear out information on retrieved files from the Debian package database. If you don't use dselect for package management, run this from time to time in order to save disk space.

TABLE 3.5 `apt-get` Commands *(continued)*

apt-get Command	Description
`autoclean`	Similar to `clean`, but only removes information on packages that can no longer be downloaded.

TABLE 3.6 Most Useful `apt-get` Options

apt-get Option	Used with Commands	Description
`-d` or `--download-only`	`upgrade`, `dselect-upgrade`, `install`, `source`	Downloads package files but does not install them.
`-f` or `--fix-broken`	`install`, `remove`	Attempts to fix a system on which dependencies are unsatisfied.
`-m`, `--ignore-missing`, or `--fix-missing`	`upgrade`, `dselect-upgrade`, `install`, `remove`, `source`	Ignores all package files that can't be retrieved (because of network errors, missing files, or the like).
`-q` or `--quiet`	All	Omits some progress indicator information. May be doubled (for instance, `-qq`) to produce still less progress information.
`-s`, `--simulate`, `--just-print`, `--dry-run`, `--recon`, or `--no-act`	All	Performs a simulation of the action without actually modifying, installing, or removing files.
`-y`, `--yes`, or `--assume-yes`	All	Produces a "yes" response to any yes/no prompt in installation scripts.

TABLE 3.6 Most Useful `apt-get` Options *(continued)*

`apt-get` Option	Used with Commands	Description
`-b`, `--compile`, or `--build`	`source`	Compiles a source package after retrieving it.
`--no-upgrade`	`install`	Causes `apt-get` to *not* upgrade a package if an older version is already installed.

If you use `apt-get` to automatically upgrade all packages on your system, you are effectively giving control of your system to the distribution maintainer. Although Debian or other distribution maintainers are unlikely to try to break into your computer in this way, an automatic update with minimal supervision on your part could easily break something on your system, particularly if you've obtained packages from unusual sources in the past.

Debian Packages Compared to Other Package Formats

The overall functionality of Debian packages is similar to that of RPMs, although there are differences. Debian source packages are not actually single files; they're groups of files—the original source tarball, a patch file that's used to modify the source code (including a file to control the building of a Debian package), and a `.dsc` file that contains a digital "signature" to help verify the authenticity of the collection. The Debian package tools can combine these and compile the package to create a Debian binary package. This structure makes Debian source packages slightly less convenient to transport because you must move at least two files (the tarball and patch file; the `.dsc` file is optional) rather than just one. Debian source packages also support just one patch file, whereas RPM source packages may contain multiple patch files. Although you can certainly combine multiple patch files into one, doing so makes it less clear where a patch comes from, thus making it harder to back out of any given change.

These source package differences are mostly of interest to software developers, however. As a system administrator or end user, you need not normally be concerned with them, unless you must recompile a package from a

source form—and even then, the differences between the formats need not be overwhelming. The exact commands and features used by each system differ, but they accomplish similar overall goals.

Because all distributions that use Debian packages in 2001 are derived from Debian, these distributions tend to be more compatible with one another (in terms of their packages) than are RPM-based distributions. In particular, Debian has defined details of its system startup scripts and many other features to help Debian packages install and run on any Debian-based system. This helps Debian-based systems avoid the sorts of incompatibilities in startup scripts that can creep into RPM systems. Of course, some future distribution could violate Debian's guidelines for these matters, so this advantage isn't guaranteed to hold over time.

As a practical matter, it can be harder to locate Debian packages than RPM packages for some more exotic programs. Nonetheless, Debian maintains a good collection at `http://www.debian.org/distrib/packages`, and some program authors make Debian packages available, as well. If you can find an RPM but not a Debian package, you may be able to convert the RPM to Debian format using the `alien` tool, described shortly. If all else fails, you can use a tarball, but you'll lose the advantages of the Debian package database.

Tarballs

All distributions can use tarballs—files collected together with the `tar` utility and typically compressed with `compress`, `gzip`, or `bzip2`. Like RPM and Debian packages, tarballs may contain source code, binary files, or architecture-independent files such as documentation or fonts. These files lack dependency information, however, and `tar` maintains no database of installed files, so it's harder to remove programs installed via tarballs than it is to remove RPM or Debian packages.

The Ubiquity of Tarballs

`tar` is a multipurpose tool. The program was originally created for archiving files to tape—the name stands for "tape archiver." Because Unix (and hence Linux) treats hardware devices as files, a tape archiving program like `tar` can be used to create archives as files on disk. These files can then be compressed, copied to floppy disk or other removable media, sent over a network, and so on.

In the Linux world, tarballs fill a role that's similar to that of zip files in the DOS and Windows worlds. There are differences, however. Zip utilities (including the `zip` and `unzip` commands in Linux) compress files and then add them to the archive. `tar`, by contrast, does not directly support compression, so to compress files, the resulting archive is compressed with a second utility, such as `gzip`, `bzip2`, or `compress`. `gzip` is the most popular on Linux systems, although `bzip2` is becoming more common and `compress` is still used on some older Unix systems. (`gzip` can uncompress old `compress` archives, so many Linux systems omit `compress`.) The resulting file may have two extensions (such as `.tar.gz` or `.tar.bz2`), or that dual extension may be combined into a single, three-character extension (`.tgz`) for easy storage on filesystems (like DOS's FAT) that don't support longer or multiple extensions. (The older `compress` archives used an uppercase `Z` extension, so these tarballs have `.tar.Z` extensions.)

Both RPM and Debian packages are similar to tarballs internally. RPM uses a compressed `cpio` archive (similar to a compressed `tar` archive) to store its files, and custom file components aside from the `cpio` archive to store RPM-specific information. Debian packages use tarballs for file storage and a control file, merged together into one file using the `ar` utility. (`ar` is an archiving utility similar to `tar` in overall principles.)

Considered as a package distribution mechanism, tarballs are used primarily by the Slackware distribution, which is the oldest of the major Linux distributions still in common use. Slackware eschews flashy configuration tools in favor of a bare-bones approach. In this respect, Slackware resembles Debian, but Slackware also foregoes the package management tools upon which Debian relies. As noted earlier, Debian also uses source tarballs as part of its source package management system, but most administrators don't need to be concerned with this detail.

Although most other distributions don't rely upon tarballs, they can be used with any distribution. Tarballs are particularly likely to be useful when you're faced with the task of compiling a program from source code, and particularly if you must modify that source code for your system. If you like, you can go to the effort of creating appropriate control files and turn a source tarball into an RPM or Debian package, but if you only need to use a program on a single computer, it's usually not worth the effort to do this.

Source code in tarball form usually comes with installation instructions. These will probably tell you to edit one or two configuration files, run a configuration command, and run two or three commands to build binary files from the source code and install them on your system. Details vary substantially from one package to another, though, so check the instructions.

Binary tarballs contain precompiled programs. Sometimes the tarball contains the program files in a form that allows you to expand the tarball directly into a target directory. For instance, you could change to the `/usr/local` directory and uncompress the tarball to have the program files dropped directly into `/usr/local/bin`, `/usr/local/man`, and so on. Other times you may need to uncompress the tarball in a temporary directory and then run an installation utility to install the software.

If you're unsure of how to proceed with a tarball installation, extract it into a temporary directory and look for instructions. Sometimes, you'll find separate installation instructions on the program's Web site or on the FTP site from which you obtained the software.

The *tar* Command Set

`tar` is a complex package with many options. Most of what you'll do with `tar`, however, can be covered with a few common commands. Table 3.7 lists the primary `tar` commands, and Table 3.8 lists the qualifiers for these commands that modify what the command does. Whenever you run `tar`, you use exactly one command and you usually use at least one qualifier.

TABLE 3.7 `tar` Commands

Command	Abbreviation	Description
`--create`	`c`	Creates an archive
`--concatenate`	`A`	Appends `tar` files to an archive
`--append`	`r`	Appends non-`tar` files to an archive
`--update`	`u`	Appends files that are newer than those in an archive

TABLE 3.7 `tar` Commands *(continued)*

Command	Abbreviation	Description
`--diff` or `--compare`	`d`	Compares an archive to files on disk
`--list`	`t`	Lists archive contents
`--extract` or `--get`	`x`	Extracts files from an archive

TABLE 3.8 `tar` Qualifiers

Command	Abbreviation	Description
`--directory` *dir*	`C`	Changes to directory *dir* before performing operations
`--file` [*host*:]*file*	`f`	Uses file called *file* on computer called *host* as the archive file
`--listed-incremental` *file*	`g`	Performs incremental backup or restore, using *file* as a list of previously archived files
`--one-file-system`	`l`	Backs up or restores only one filesystem (partition)
`--multi-volume`	`M`	Creates or extracts a multitape archive
`--tape-length` *N*	`L`	Changes tapes after *N* kilobytes
`--same-permissions`	`p`	Preserves all protection information
`--absolute-paths`	`P`	Retains the leading / on filenames
`--verbose`	`v`	Lists all files read or extracted; when used with `--list`, displays file sizes, ownership, and time stamps
`--verify`	`W`	Verifies the archive after writing it

TABLE 3.8 `tar` Qualifiers *(continued)*

Command	Abbreviation	Description
`--exclude` *`file`*	(none)	Excludes *`file`* from the archive
`--exclude-from` *`file`*	X	Excludes files listed in *`file`* from the archive
`--gzip` or `--ungzip`	z	Processes archive through `gzip`
`--bzip2`	I or y	Processes archive through `bzip2`

Of the commands listed in Table 3.7, the most commonly used are `--create`, `--extract`, and `--list`. The most useful qualifiers from Table 3.8 are `--file`, `--listed-incremental`, `--one-file-system`, `--same-permissions`, `--gzip`, `--bzip2`, and `--verbose`. (`--bzip2` is a fairly recent addition, so it may not work if you're using an older version of `tar`.) If you fail to specify a filename with the `--file` qualifier, `tar` will try to use a default device, which is often (but not always) a tape device file.

A typical `tar` command to extract files from a tarball looks like this:

```
# tar --extract --verbose --gunzip --file
↳samba-2.0.7.tar.gz
```

This command can be expressed somewhat more succinctly using command abbreviations:

```
# tar xvzf samba-2.0.7.tar.gz
```

In either form, this `tar` command extracts files from `samba-2.0.7.tar.gz` to the current directory. Most tarballs include entire directory trees, so this command results in one or more directories being created, if they don't already exist, as well as files within the directories.

Before extracting a tarball, use the `--list` command to find out what files and directories it contains. This information can help you locate the files stored in the tarball. In addition, it can help you spot problems before they should occur in case a tarball does *not* contain a neat directory structure, but instead contains files that would all be dropped in the current directory.

Tarballs Compared to Other Package Formats

Although all Linux distributions ship with `tar`, `gzip`, and usually `bzip2`, few use these tools as a means of distributing packages that are part of the OS. The reason is that `tar` lacks any means of maintaining a package database. Although it's possible to create a set of tarballs that together contain a complete Linux distribution, and it's even possible to write installation scripts to install appropriate subsets of tarballs, maintaining such a system poses certain challenges. Without dependency information or information on what files belong to what packages, it can become difficult to remove packages or even install new ones that may depend on other packages. If you install a new mail reader, for instance, it might crash or fail to start because it depends on a library you don't have or on a newer version of a library that you do have. Experienced system administrators can sometimes diagnose such problems without too much trouble, but these difficulties frequently stump new administrators.

One feature of `tar` that can be very useful is that the program can be used to easily create packages, as well as extract files from them. You can use this feature to move data files, documentation, or programs you've written or built yourself. Of course, you can also create RPM or Debian packages, but this process is more complex, and the usual method of doing this requires that you provide a tarball of the source code to begin with. It's easiest to create a tarball of all the files in a single directory, thus:

```
# tar cvzf my-stuff.tgz my-stuff-dir
```

A similar command can be used to back up a directory or even an entire computer to tape. Instead of providing a tarball filename, though, you specify the name of a tape device file, such as `/dev/st0`, for the first SCSI tape unit. This topic is further discussed in Chapter 7.

Whether or not you create your own tarballs or your distribution uses tarballs for its packages, you should be familiar with `tar` because of the common nature of tarballs as a source code distribution mechanism and because of `tar`'s utility as a tape backup tool. (You may choose to use other tools for tape backup, but `tar` is the lowest-common-denominator choice for this task.)

Converting between Package Formats

Sometimes you're presented with a package file in one format, but you want to use another format. This is particularly common when you use a Debian-based distribution and can only find tarballs or RPM files of a package, or

when you use a tarball-based distribution and want to use an RPM or Debian package. When this happens, you can keep looking for a package file in the appropriate format, install the tools for the foreign format, create a package from a source tarball using the standard RPM or Debian tools, or convert between package formats with a utility like `alien`.

This section focuses on this last option. The `alien` program comes with Debian and a few other distributions. This program can convert between RPM packages, Debian packages, Stampede packages (used by Stampede Linux), and tarballs. There are some caveats, however. For one thing, `alien` requires that you have appropriate package manager software installed—for instance, both RPM and Debian to convert between these formats. `alien` doesn't always convert all dependency information completely correctly. When converting from a tarball, `alien` copies the files directly as they had been in the tarball, so `alien` works only if the original tarball has files that should be installed off of the root directory (`/`) of the system.

Although `alien` requires both RPM and Debian package systems to be installed to convert between these formats, `alien` doesn't use the database features of these packages unless you use the `--install` option. The presence of a foreign package manager isn't a problem so long as you don't use it to actually install software that might duplicate or conflict with software installed with your primary package manager.

The basic syntax of `alien` is as follows:

```
alien [options] file[...]
```

The most important options are `--to-deb`, `--to-rpm`, `--to-slp`, and `--to-tgz`, which convert to Debian, RPM, Stampede, or tarball format, respectively. (If you omit the destination format, `alien` assumes you want a Debian package.) The `--install` option installs the converted package and removes the converted file. Consult the `alien` man page for additional options.

For instance, suppose you have a Debian package called `someprogram-1.2.3-4_i386.deb`, but you want to create an RPM from this. You could issue the following command to create an RPM called `someprogram-1.2.3-5.i386.rpm`:

```
# alien --to-rpm someprogram-1.2.3-4_i386.deb
```

If you use a Debian-based system and want to install a tarball but keep a record of the files it contains in your Debian package database, you can do so with the following command:

```
# alien --install binary-tarball.tar.gz
```

It's important to remember that converting a tarball converts the files in the directory structure of the original tarball using the system's root directory as the base. Therefore, you may need to unpack the tarball, juggle files around, and repack it to get the desired results *prior to* installing the tarball with `alien`. For instance, suppose you've got a binary tarball that creates a directory called `program-files`, with `bin`, `man`, and `lib` directories under this. The intent might have been to unpack the tarball in `/usr` or `/usr/local` and create links for critical files. To convert this tarball to an RPM, you might issue the following commands:

```
# tar xvfz program.tar.gz
# mv program-files usr
# tar cvfz program.tgz usr
# rm -r usr
# alien --to-rpm program.tgz
```

By renaming `program-files` to `usr` and creating a new tarball (`program.tgz` as opposed to the original `program.tar.gz`), you've created a tarball that, when converted to RPM format, will have files in the locations you want—`/usr/bin`, `/usr/man`, and `/usr/lib`. You might need to perform more extensive modifications, depending upon the contents of the original tarball.

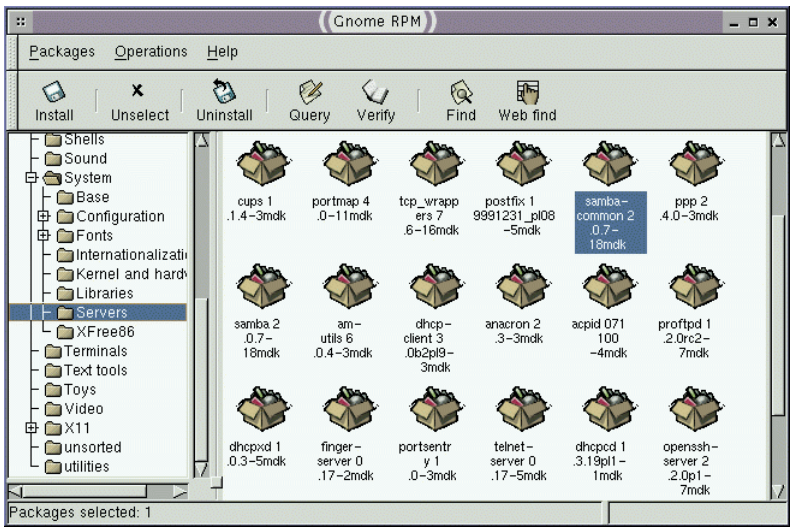
GUI Package Management Tools

The text-mode utilities that underlie all Linux package management tools are very flexible, but they can be intimidating to new administrators. For this reason, several GUI administrative tools also exist. These programs typically serve as front ends to the text-based programs described earlier—that is, they call the text-mode programs and interpret the results in a GUI window. Many distributions ship with their own unique GUI package management tools, and it's impossible to cover them all here. Therefore, this section outlines just one: GNOME RPM. Others are similar to this one.

GNOME RPM is useful only on systems that use RPM. If you're using a Debian-based system, a similar utility is the Storm Package Manager, which is part of Storm Linux (`http://www.stormix.com`), but it can be used on Debian or other Debian-based systems.

The GNOME RPM program (see Figure 3.1) is a popular GUI interface to the RPM utilities. GNOME RPM ships with the Red Hat and Mandrake distributions, and it can be installed and used on other RPM-based distributions. GNOME RPM supports basic package installation, removal, upgrading, and maintenance functions. It lacks interfaces to some of the more advanced RPM features, such as the tools used to build binary RPMs from source RPMs, or to create your own RPM packages. To start it, type **`gnorpm`** in a command prompt window such as an `xterm`.

FIGURE 3.1 GNOME RPM provides a point-and-click interface to package management on RPM-based systems.



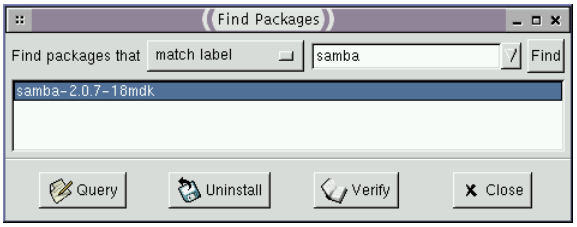
The left pane of GNOME RPM contains a list of package groups. Some of these groups have subgroups—for instance, Figure 3.1 shows the System group open, revealing the subgroups of Base, Configuration, Fonts, and so on. These groups are not unique to GNOME RPM; they're stored as part of

an RPM file's basic information. You can learn an RPM's group from the command line by typing **`rpm -qi packagename`**.

Not all distributions use package groups. In particular, SuSE doesn't group its packages. This can make it difficult to locate packages by type, but because the classification isn't entirely standardized in distributions that use groups, the presence of groups isn't always a great help.

One trouble with RPMs is that the structure of package groups isn't always consistent. You might think that a package should be in System/Servers, but it might actually be in Networking/Daemons. If you know the package name, this isn't a problem with text-based tools, but it can be with GNOME RPM. The solution is to use the Find Packages feature, which you can open by choosing Operations ➢ Find. This produces the Find Packages dialog box (Figure 3.2). Enter search parameters in the Find Packages That field and select a search criterion, such as Match Label, shown in Figure 3.2. Click Find and Find Packages will display all the packages that match the criterion you entered. You can then select the package from the list (only one is shown in Figure 3.2) and query, uninstall, or verify the package by clicking the appropriate button.

FIGURE 3.2 The GNOME RPM Find Packages dialog box locates packages that meet your criteria and lets you perform certain actions on them.



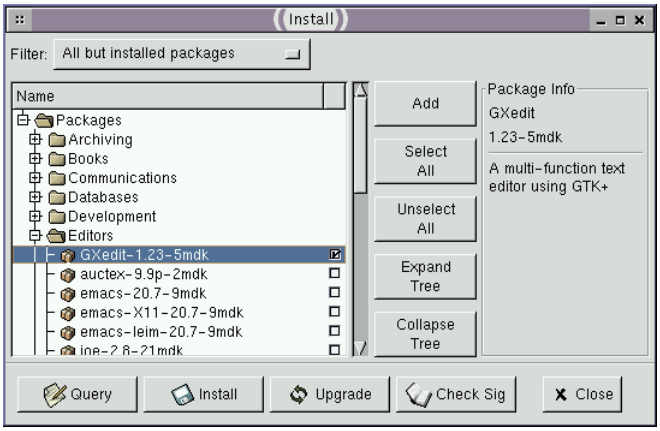
The right pane of the main GNOME RPM window shows all the packages installed in whatever group you've selected in the left pane of the window. If you want to perform an operation on an installed package, you can click one or more of these packages and choose the appropriate action from the menu bar. For instance, to uninstall a package, select it and choose Packages ➢ Uninstall. To verify a package's integrity, select the package and choose Packages ➢ Verify. (Alternatively, you can use the appropriate

buttons just below the menu bar.) In both cases, GNOME RPM will show a dialog box with information on the tool's progress, and then a status dialog box. When verifying a package, the status dialog box will include information on any deviations from the original installed state of the package, such as missing or altered files.

GNOME RPM separates its package group and package listings into two panes, but this isn't true of all package management utilities. Some use a single pane similar to the package group listing of GNOME RPM, but individual packages are included in this list.

To install a package, choose Operations ➢ Install. The result is the Install dialog box, shown in Figure 3.3. If you've mounted your installation CD-ROM, you should see uninstalled packages listed in an expandable group tree. Locate the package you want and check the box to the right of its name, as shown for GXedit in Figure 3.3; then click Install. If you want to install a package you've downloaded, click Add. This produces a file selection dialog box in which you can pick the packages you've obtained and add them to the list of packages, then install them.

FIGURE 3.3 The GNOME RPM Install dialog box lets you install packages from your distribution's installation CD-ROM or from any other source.



Real World Scenario

Why Use GUI over Command-Line Package Management Tools?

Even experienced administrators who understand the command-line tools can make good use of GNOME RPM, Storm Package Manager, or similar utilities. The fact that such programs provide browseable listings of all packages installed in a system makes them excellent tools for pruning a system of unneeded packages. By spending a few minutes examining all the package categories, you can get a good idea of what your distribution has installed in your computer and remove anything that's unnecessary. This action can save many megabytes of disk space, and it can even improve your system's security, if you remove unwanted servers or programs that might pose security risks.

Validating Proper Program Functioning

Unfortunately, it's usually not sufficient to install a package and then walk away from the system. Programs frequently require configuration, and even those that *should not* require explicit attention by the system administrator may misbehave in various ways. Problems may be caused by incompatible libraries, missing dependencies, bugs, invalid assumptions the program makes about the working system, and so on. Some of these issues are covered in greater detail in Chapter 9,"Troubleshooting." Before jumping the gun and assuming problems exist, though, you should perform at least minimal tests of the software.

Checking an Application on a Test System

In a mission-critical production environment, it's seldom wise to install new software directly on the main production system. Even a trivial upgrade (say, version 2.4.6 to 2.4.7) can introduce bugs, changes to configuration file formats, incompatibilities with interdependent programs, and so on. For this reason, evaluating new software on a test system is generally a good idea.

Ideally, your test system should be configured identically to the ultimate production system you plan to use—or as identically as can be managed. For instance, you should have the same hardware, the same version of Linux, and all the same libraries and other support packages. The systems should be configured the same way in terms of networking (but they probably won't have the same IP addresses unless you can set up a small isolated test network). If the software you want to test is an upgrade of a package that's already installed on the production system, you should first install the version that's working on the production system on the test system, and then configure it in the same way. If possible, copy the production system's configuration files to the test system, along with any data files. For instance, if you were testing an upgrade to a Web server, you would copy the Web server's configuration files and your entire collection of Web pages. Note that in some cases, these files will require modification so that they reference the test system rather than the production system. Web pages may have URLs that point to the production system by name, for instance.

In practice, however, you may not have the luxury of creating a near-exact copy of the production system for testing purposes. You may have to make do with a previous-generation computer system with radically different hardware, for instance. Depending upon what software you're testing, this may not be too big of a handicap, though, because most software doesn't really care about your hardware. Be aware, however, that drivers are a notable exception to this rule. Therefore, it's important that you test with the same Linux distribution you use on the production system. Distributions frequently contain radically different package collections, so a product that runs perfectly on the test system could fail miserably on the production system.

Once you've configured your test system to be as similar to your production computer as possible, *read the documentation!* Some administrators take an install-first-and-ask-questions-later approach to software installation. Such an approach may work well on occasion, but it's like playing Russian Roulette—sooner or later, you'll end up killing your system, or at least doing it serious damage. Of course, if you're doing the installation on a disposable test system, this won't be a crippling setback, but it will still cost you the time it took to prepare the test system. Reading the documentation can alert you to important software features, changes since the release you're currently running, bugs, and what files the software will be installing or replacing.

If you're using a package system like RPM or Debian, you may think you have to install the software in order to read its documentation, but you can

often find the documentation on the package's Web site. If you do have to install the documentation, see if the program you are using comes with separate documentation packages; that way you can install and read the documentation before you install the main package. Of course, one of the benefits of package management systems is that you can easily back out of an upgrade, so it's unlikely you'll do too much damage even if you install the main package without checking out its documentation first. If you do this, though, be sure to read the documentation soon after installing the software, so you'll know what to expect when you try to use it.

When you install or upgrade your test software, pay particular attention to what happens to configuration files. RPM and Debian packages usually leave configuration files alone, or at least back up your old configuration files; but sometimes the package isn't built correctly, and installing the package wipes out your old configuration files. If this happens, you'll need to restore your original configuration files, presumably from the production system.

If you're lucky, the software will run perfectly the first time you try it. If not, you'll have to track down the cause of the problem. This is as much an art as a science, so it's hard to say where to begin, especially as a general rule—different packages are so different that good advice for one can be useless for another. As a general rule, configuration files and library problems are good places to start looking. In the case of a package upgrade, configuration file formats sometimes change. This is particularly true of major version upgrades and upgrades within alpha- or beta-test cycles. (Running such prerelease software is generally inadvisable on a production system, but sometimes you have no choice because nothing else is available or you need the advanced features being added in the alpha or beta software.)

You can refer to Chapter 9 for more detailed information on troubleshooting installation errors and setbacks.

Precisely how you test the software depends on the program in question. A small utility might require simply using it on a few test cases. A major package like a new Web server for an important Web site might require extensive formal tests, using multiple Web browsers to access all of the site's pages, and perhaps even performing benchmarks of its operation compared to that of its predecessor. Upgrading important support libraries, such as libc, can have consequences for literally hundreds of other programs, so you

may need to test lots of separate packages when you do such an upgrade. You'll have to be the judge of how much testing to do. Keep in mind the consequences of installing a bad package on the production system. If such an upgrade could cause you or your users serious grief, substantial verification on the test system is in order.

During this entire process, you should take notes. If you run into problems and find solutions, your notes will help you implement these solutions when it comes time to install the software on the production system. If you encounter many problems or have trouble finding a solution, you may want to get your notes in order, restore the test system to its original condition, and practice reinstalling the new software so that you're sure you can do it with minimal downtime on the production system.

Checking an Application on a Production System

Once you've checked the application on the test system, it's time to install it on a production system. It's usually wise to back up this computer before the upgrade, particularly if it's a major upgrade. This way, if some snafu occurs, you can restore the system to its original state and go back to your test system to try to determine what went wrong.

After backing up the production system, install the new software, following your notes if necessary. If you're upgrading a program and this upgrade requires taking down a server program or the entire computer, be sure to schedule the upgrade for some time when it will cause the least disruption, such as a weekend or late at night. If such a swap is practical, you might also want to have your test system standing by, ready to take over the main system's duties in case you run into unexpected difficulties. Be sure to notify your users of the upgrade so that they're not caught unaware.

Once you've performed the installation or upgrade, you should run through at least minimal tests of the software in its new production environment. You may not have time to perform a full suite of tests, particularly for a big package, but you should at least be able to assess minimal functionality.

Some packages are so trivial that the preceding discussion may seem like overkill. Likewise, in a small shop, you may not have the luxury of hardware to use for a test system. In such situations, you should be extra careful to check the documentation before you install, and in the case of an upgrade,

make sure to back up the configuration files and have a plan to restore the original software should you encounter problems. It's easy to install a "new and improved" package, find that it doesn't work, and then spend an hour or more frantically searching for the version you had been using, or perhaps you will find it but you'll have trouble compiling or installing it. If you plan properly, you can avoid the aggravation and embarrassment of this situation.

Ongoing Application Monitoring

Many programs require ongoing monitoring. Shortly after installing a major package, you should certainly check that it's working correctly however you can, say by running it with test data or checking the operation of a server from another system. You can periodically use `ps` to see that it's still running (if it *should* be running constantly), or try to use the software (for instance, by using another system's Web browser to access a new Web server).

Some programs produce log files. You can use these to monitor their performance. Log files generally go in the `/var/log` directory tree. Consult your package's documentation for details. It's usually a good idea to monitor these logs with extra diligence soon after an upgrade. Doing so can alert you to problems that you might not otherwise notice, like a Web server that's delivering Web pages only to local systems and not to the Internet at large.

Log files are discussed in more detail in the "Monitoring Log Files" section of Chapter 4.

Kernel Issues

The Linux kernel is at the heart of any Linux system. In fact, technically speaking, the kernel *is* Linux. Everything else—the C libraries, the startup files, the X servers, and so on—is independent of the Linux kernel, and most of this code is used in other Unix-like OSs, such as FreeBSD. Many of these programs were written under the auspices of the Free Software Foundation's GNU's Not Unix (GNU) project. For this reason, some people refer to Linux as an OS as *GNU/Linux*.

In any event, the Linux kernel is both critically important and unusual in its requirements. Therefore, it's important that you understand how this particular piece of the software puzzle fits into Linux as a complete OS.

The Role of the Kernel

Any OS's kernel provides critical low-level facilities. These include the following:

- Memory management, including handling swap space, if present
- Most low-level hardware drivers (X's video drivers and printer drivers are two major exceptions)
- Process scheduling—determining when specific programs get access to the CPU
- Large chunks of the network stack, providing programs with network access
- Filesystems, for access to files on disk

For the most part, users don't interact directly with the kernel, except insofar as the kernel handles keyboard input and text-based video output. Programs, however, rely upon the kernel for the just-mentioned functions. Because of this, programs can only be as reliable as the kernel—an unreliable kernel can manifest itself in unreliable programs or overall system instability. Similarly, applications may be limited in what they can accomplish if the kernel lacks features. Most 2.2.*x* Linux kernels, for instance, lacked support for Universal Serial Bus (USB) devices, so programs using these kernels couldn't use USB hardware. (The 2.2.18 and 2.4.*x* kernels include much-improved USB support.)

As a system administrator, you have several kernel-related duties:

- You must determine what kernel to use. A too-old kernel can limit your hardware options, but a too-new kernel can be unreliable. The upcoming section, "Kernel Version Numbering," includes important information about this.
- You must determine how to install a kernel. All Linux distributions ship with precompiled kernels in packages similar to other software. You may prefer to compile a kernel yourself, though. "When to Recompile the Kernel" will help you decide when to do this.

- You must configure your system to boot the kernel you desire. "Configuring Boot Loaders" covers this topic. (This is done automatically when you install Linux, but you'll need to deal with this if you upgrade your kernel.)
- You must configure kernel modules for new or upgraded hardware. This topic is covered in Chapter 8, "Hardware Issues."

Kernel Version Numbering

Because the kernel is arguably the most important software component in a Linux computer, it's particularly important that you understand how Linux kernel version numbers work. There's a definite system to the Linux kernel version numbers, and understanding it can help you determine when you might want to upgrade the kernel. Also, this method of version numbering has been adopted by some non-kernel software projects, and so it can be useful in understanding them, as well.

Don't confuse the Linux *kernel* version number with the Linux *distribution* version number. The two are unrelated. For instance, Linux Mandrake 8.0 (a distribution) ships with kernel 2.4.3. Some new Linux administrators merge the two, and say they're running (for instance) "Linux 8.0." There is no such thing—at least, not in 2001. "Linux 2.4.3" unambiguously refers to the kernel version 2.4.3, but when referring to a distribution, you *must* specify its name.

Linux kernel versions all have three numbers, separated by periods (`.`):

- The first number is the *major version number*. In 2001, this number is 2. A change in the major version number typically indicates a fairly substantial change in the way the kernel works.
- The second number carries special meaning. When this number is even, it indicates a *stable kernel* or *release kernel*—one that the kernel developers believe to be fairly bug-free and ready for use by the public at large. Odd second numbers indicate *development kernels*. These kernels are experimental—they contain major new drivers, major rewrites of existing code, and so on. You should only use a development kernel if you're desperate for some feature it provides—and even then, you can often find a backport (transfer of newer drivers to older kernels) of such features to a stable kernel.

- The final number indicates a minor upgrade. In the case of stable kernels, these numbers typically indicate small bug fixes and occasionally the addition of an important but well-tested new driver. In the case of development drivers, the final number indicates progression in adding features, fixing bugs, and (being realistic) adding new bugs along with the new features.

At any given time, *two* Linux kernels are current—one in the stable release tree, and another in the development line. When Linus Torvalds decides that a development kernel is approaching stability with the target features for the next stable release, he calls for a feature freeze so that no new drivers or features will be added or changed, except to fix bugs. Over the next few weeks or months, the development kernel's stability improves, and it's eventually released as the next stable kernel. For instance, the 2.3.*x* kernel series led to the release of a 2.4.0 kernel. When a new stable kernel is released, the former development kernel is abandoned and work is begun on a new line—2.5.*x* after the release of 2.4.0, for instance. This line will eventually lead to the release of a 2.6.0 or 3.0.0 kernel.

Major Linux distributions always ship with stable kernels. In early 2001, these were late in the 2.2.*x* series, but with the release of the 2.4.0 kernel, distributions released by mid-2001 had 2.4.*x* kernels. As a general rule, it's best to use a kernel in the same series that the distribution uses. Changes to the second value of the kernel number tend to introduce changes to kernel interfaces required by some low-level utilities, modifications to filesystem formats, new driver names, and the like. Although most distributions continue to function with such an upgraded kernel, there may be a few glitches, such as drivers that don't load automatically. Near the release of a new kernel, some distributions try to make their systems ready for the new kernel. Red Hat 7.0, for instance, aimed to be compatible with the 2.4.0 kernel. Such attempts are not always completely successful, but if you must upgrade soon after a new kernel is released, using such a distribution can help.

Within a single series, upgrades are usually safe. The most common reason to upgrade is if a new version fixes a bug that's been found in an older version. Another reason to upgrade is if a new version adds a driver. For instance, the 2.2.18 kernel added a large number of drivers that had been developed in the 2.3.*x* series. This allowed people to take advantage of many 2.4.0 kernel features, such as support for USB devices, without upgrading to 2.4.0 and running into the glitches that would be likely with such an upgrade.

When to Recompile the Kernel

Kernels can be upgraded much like other packages, by installing a single binary package file. (You may also need to explicitly reconfigure your boot loader, although this may be done by automatic installation scripts.) Many Linux administrators, though, prefer to compile their own kernels from source code. Doing so offers several advantages:

Optimizing the kernel If you compile your own kernel, you can optimize it for your particular level of CPU (486, Pentium, and so on). This can improve performance slightly.

Individual driver configuration Most Linux drivers can be compiled either as part of the main kernel file or as add-in modules. Most distributions ship with most drivers compiled as modules, but you can configure a custom kernel to include in the main kernel file drivers you normally have loaded, such as drivers for your SCSI adapter or Ethernet card. Likewise, you can completely remove drivers that you don't need, such as EIDE drivers on a SCSI-only system.

Kernel standardization Many distributions include modified versions of the kernel by default. Changes may include adding non-standard or updated drivers. These changes are often beneficial, but sometimes they cause problems. Compiling your own kernel from a standard kernel source tarball ensures you're using a standard kernel, or at least one with just the changes you've decided to make. Note that many distribution maintainers provide kernel source in RPM or Debian package format, but these sources may include changes, as well. You're best off getting a kernel from a generic Linux kernel repository site, like `http://www.kernel.org`.

Ability to apply patches Some drivers are developed separately from the main kernel source tree, at least initially. Sometimes, these must be patched into the kernel source code and compiled with the rest of the kernel. This is impossible if you use a precompiled kernel.

Kernel patches are generally intended for specific kernel versions. You can often get away with applying a patch for a close version (such as a patch intended for 2.2.16 applied to 2.2.17), but the greater the version difference, the more likely it is that a problem will crop up.

As a general rule, it's seldom strictly necessary to compile your own kernel; however, it's often beneficial to do so, for any or all of the preceding reasons. The details of kernel configuration and compilation are beyond the scope of this book, however. Consult the Linux Kernel HOWTO document (`http://www.linuxdoc.org/HOWTO/Kernel-HOWTO.html`) for more details.

Configuring Boot Loaders

The kernel is an unusual program in many ways, not the least of which is how it's loaded. Because the kernel must run before Linux is completely booted, the kernel must be loaded into memory in a unique way. A program known as a *boot loader* handles this task. Several boot loaders are available, some of which can boot a Linux kernel directly, others of which require help to do the job.

This section discusses boot loaders for *x*86 systems. If you're using Linux on another architecture, such as PPC (Macintosh) or Alpha, the available boot loaders will be different. Consult your distribution's documentation for details.

The Role of the Boot Loader

When it's first powered up, an *x*86 CPU checks a specific area of memory for code to execute. This code is the Basic Input/Output System (BIOS). You're probably familiar with the BIOS through your computer's BIOS setup screens, which allow you to configure features such as RAM timing and whether or not on-board ports are active. The BIOS also provides code that allows the computer to boot. The BIOS checks the first sector of your hard disk (or of your floppy disk, CD-ROM, or other disk devices, depending upon the BIOS's capabilities and configuration) for a small boot loader program. This program normally resides on the *master boot record (MBR)* of a hard disk, or the *boot sector* of a floppy disk. The MBR resides on the first sector of a hard disk, and controls the boot process. A boot sector is the first sector of a floppy or of a hard disk partition and also controls the boot process. (In the case of a partition's boot sector, it's used after the MBR.)

In the case of a PC that runs nothing but Windows, the boot loader in the MBR is hard-coded to check for a secondary boot loader in the active primary partition. This secondary boot loader directly loads the Windows kernel. The approach in Linux is similar, but standard Linux boot loaders are somewhat more complex. The most popular Linux boot loader, the *Linux Loader (LILO)*, allows you to select from several different boot options, for both Linux and non-Linux OSs. Other boot loaders are also available, as described shortly, most of which also allow you to select which of several OSs to run. Unfortunately, this added complexity comes at a cost: Advanced boot loaders are more difficult to configure than is the simple single-OS boot loader used with Windows.

In some cases, a system uses multiple boot loaders. One resides in the MBR, and another resides in the boot sector of an individual disk partition. (OSs on different partitions can each have their own boot sector–based boot loaders.) In this configuration, the MBR-based boot loader is the *primary boot loader*, and the one in a partition's boot sector is a *secondary boot loader*. Some boot loaders work in only one of these positions. It's often possible for a secondary boot loader to redirect the boot process to a different partition, in which case that partition's boot loader becomes the tertiary boot loader, although the configuration is the same as for secondary status.

Available Boot Loaders

Many OSs ship with their own boot loaders, and others are available from third parties. Here are some of the most common boot loaders:

LILO As mentioned earlier, LILO is the most popular boot loader for Linux. It can directly boot a Linux kernel, and it can function as either a primary or a secondary boot loader. It may also be installed on a floppy disk, which is unusual for a boot loader. When used as a secondary boot loader, LILO should *only* be installed in a Linux partition; it will damage the contents of most non-Linux filesystems. Installing LILO in a swap partition is also inadvisable since it will be wiped out by swap activity. LILO can redirect the boot process to non-Linux partitions, and so it can be used to select Linux or Windows in a dual-boot system.

GRUB The *GRand Unified Bootloader (GRUB)* is an alternative to LILO that's becoming more common. GRUB was the first boot loader that could directly boot Linux from above the 1024th cylinder of a hard disk, which gained it some popularity. LILO has since achieved similar capabilities, though.

OS Loader This is one name by which Windows NT/2000's boot loader goes. Another is `NTLDR`. This is a secondary boot loader that cannot directly boot Linux, but it can boot a disk file that can contain LILO, and hence boot Linux indirectly. It's common on some dual-boot installations.

System Commander This boot loader, from V Communications (`http://www.v-com.com`), is the Cadillac of boot loaders, with some very advanced features. It cannot directly boot Linux, but like many others, it can direct the boot process to a Linux partition on which LILO is installed.

`LOADLIN` This is an unusual boot loader in that it's neither a primary nor a secondary boot loader. Rather, it's a DOS program that can be used to boot Linux after DOS has already loaded. It's particularly useful for emergency situations since it allows you to boot a Linux kernel using a DOS boot floppy, and you can also use it to pass kernel parameters to influence the booted system's behavior. `LOADLIN` comes with most Linux distributions, generally in a directory on the main installation CD-ROM.

After installing Linux, create a DOS boot floppy with `LOADLIN` and a copy of your Linux kernel. You can then use this boot floppy to boot Linux if LILO misbehaves or your kernel is accidentally overwritten.

There are many additional third-party boot loaders, most of which, like System Commander, cannot directly boot a Linux kernel but can boot a partition on which LILO is installed. For this reason, this chapter emphasizes LILO configuration—LILO can be used to boot Linux, whether LILO functions as a primary, secondary, or tertiary boot loader. If you opt to use LILO as a secondary boot loader, you'll need to consult the documentation for your primary boot loader to learn how to configure it.

On a Linux-only system, there's no need to deal with an advanced third-party boot loader; LILO can function as a primary boot loader without trouble on such systems. Third-party boot loaders are most useful when you have two or more OSs installed, and particularly when LILO has troubles redirecting the boot process to the other OSs, which is rare.

The usual configuration for LILO places it in the MBR. Even in a Linux-only situation, however, it's sometimes desirable to place LILO in the Linux boot partition. Used in this way, a standard DOS/Windows MBR will boot Linux *if* the Linux boot partition is a primary partition that's marked as active. This configuration can be particularly helpful in DOS/Linux or Windows/Linux dual-boot configurations because DOS and Windows tend to overwrite the MBR at installation. Therefore, putting LILO in the Linux boot sector puts it out of harm's way, and you can get LILO working after installing or reinstalling DOS or Windows by using the DOS or Windows `FDISK` program and marking the Linux partition as active. If LILO is on the MBR and is wiped out, you'll need to boot Linux in some other way, such as by using `LOADLIN`, and then rerunning the `lilo` program to restore LILO to the MBR.

The 1024-Cylinder Limit

One bane of the PC world that reared its ugly head twice in the 1990s was the so-called *1024-cylinder limit*. This limit is derived from the fact that the *x*86 BIOS uses a 3-number scheme for addressing hard disk sectors. Each sector is identified by a cylinder number, a head number, and a sector number, known collectively as the sector's CHS address. The problem is that each of these values is limited in size. The cylinder number, in particular, is allotted only 10 bits, and so cannot exceed 2^{10}, or 1024, values. In conjunction with the limits for sectors and heads, this limited addressable EIDE hard disk size to precisely 504MB in the early 1990s.

When disks larger than 504MB became common, BIOSes were adjusted with *CHS translation* schemes, which allowed them to juggle numbers between cylinders, heads, and sectors. This increased the effective limit to just under 8MB. A similar scheme abandoned CHS addressing for BIOS-to-disk communications but retained it for BIOS-to-software communications. This was known as *linear block addressing (LBA)* mode.

These limits never affected Linux once it had booted, because Linux could handle more than 10-bit cylinder values, and it could access disks directly using LBA mode. The Linux boot process was limited, however, because LILO relied upon CHS addressing via the BIOS to boot the kernel. Therefore, the Linux kernel has traditionally had to reside below the 1024-cylinder mark.

> Today, all new BIOSes include support for so-called *extended INT13* calls, which bypass the CHS addressing scheme. These BIOSes allow booting an OS from past the 1024-cylinder mark on a hard disk, but only if the boot loader and OS support this feature. Recent versions of LILO support extended INT13 calls, so new Linux distributions can be installed anywhere on a hard disk—if the BIOS supports this feature.

An Overview of the LILO Configuration File

LILO is configured through the `/etc/lilo.conf` file. This file consists of lines with general configuration options, followed by one or more stanzas—groups of lines that define a single OS to be booted. For instance, Listing 3.3 shows a simple `lilo.conf` file that defines a system that can boot either Linux or Windows.

Listing 3.3: Sample `lilo.conf` File

```
boot=/dev/hda
prompt
delay=40
map=/boot/map
install=/boot/boot.b
default=linux
lba32
message=/boot/message
image=/boot/bzImage-2.2.17
        label=linux
        root=/dev/hda9
        append="mem=128M"
        read-only
other=/dev/hda3
        label=windows
        table=/dev/hda
```

Each line contains a command that defines some aspect of LILO's operation. The following entries describe important general configuration options shown in Listing 3.3:

`boot=/dev/hda` This option tells LILO that it will install itself to `/dev/hda`—the MBR of the first physical EIDE disk. To install LILO as a secondary boot loader, put it in a Linux partition, such as `/dev/hda9`.

`prompt` This option tells LILO to prompt the user. By default, LILO uses a simple text-mode prompt, such as `lilo:`. Many modern distributions include additional parameters that add menu-based and graphical prompts.

`delay=40` LILO boots a default OS after a configurable delay, expressed in tenths of a second. This line sets the delay to 4 seconds.

`default=linux` This specifies the default OS or kernel to boot; it refers to the `label` line in the stanza in question. If this option is omitted, LILO uses the first stanza as the default.

`lba32` This option enables the ability to boot kernels located past the 1024th cylinder of the disk (about 8MB on most modern hard drives).

Other options are present, but you're not likely to need to change them unless you need to customize how LILO appears for users or enable advanced features.

Each stanza begins with its own line—`image=` for Linux kernels or `other=` for other OSs, such as DOS or Windows. Listing 3.3 shows all but the first line of each stanza indented, but this isn't required; it simply helps distinguish the stanzas from each other. Important options for specific stanzas include the following:

`label` This is the name by which an OS or kernel will be known. In the default LILO configuration, the user types this name at the `lilo:` prompt. If LILO is configured to use a menu, the name appears in that menu. Every stanza must have a `label` definition. Although Listing 3.3 shows labels named after the OSs in question, these labels are, in fact, arbitrary.

`root` This option sets the root (`/`) filesystem for a Linux system. Once booted, the Linux kernel looks here for startup scripts, `/etc/fstab` (for the locations of other filesystems), and so on.

`append` This optional line lets you pass parameters to the kernel. These parameters influence the way the kernel treats hardware. Listing 3.3 includes an `append` option that tells the kernel that the system has 128MB of RAM. (Linux usually detects this correctly, but some BIOSes throw Linux off.) You can also tell Linux what settings (IRQs and DMA channels) to use for hardware, if the drivers are built into the kernel.

read-only Linux normally starts up by booting the root filesystem in read-only mode, and later switches it to full read/write mode. This option tells Linux to behave in this way; it's a standard part of a Linux boot stanza.

table This option allows LILO to pass the location of the boot disk's partition table to a non-Linux OS. This is required for some OSs to boot. Its normal value is `/dev/hda` for EIDE disks or `/dev/sda` for SCSI disks.

LILO is a complex program that has many additional options. Consult the `lilo.conf` man page for more information on these options.

It's important to realize that there are three aspects to LILO:

- The LILO configuration file, `/etc/lilo.conf`
- The installed boot loader, which resides in the MBR or boot sector
- The `lilo` program, which converts a `lilo.conf` file into an installed boot loader

After you've edited `/etc/lilo.conf`, you *must* type **`lilo`** to activate your changes. If you omit this step, your system will continue to use the old boot loader.

Adding a New Kernel to LILO

It's possible to configure LILO to boot either of two or more kernels using the same distribution. This can be very convenient when you want to test a new kernel. Rather than eliminate your old working configuration, you install a new kernel alongside the old one and create a new `lilo.conf` entry for the new kernel. The result is that you can select either the old kernel or the new one at boot time. If the new kernel doesn't work as expected, you can reboot and select the old kernel. This procedure allows you to avoid otherwise ugly situations should a new kernel not boot at all.

Assuming you don't need to change kernel `append` options or other features, one procedure for adding a new kernel to LILO is as follows:

1. Install the new kernel file, typically in `/boot`. Ensure that you *do not* overwrite the existing kernel file, though. If you compile your own kernel, remember to install the kernel modules (with **`make modules_install`**), as well.

2. Copy the stanza for the existing kernel file in `/etc/lilo.conf`. The result is two identical stanzas.
3. Modify the name (`label`) of one of the stanzas to reflect the new kernel name. You can use any arbitrary name you like, even a numeric one, such as `242` for the 2.4.2 kernel.
4. Adjust the `image` line in the new kernel's stanza to point to the new kernel file.
5. If you want to make the new kernel the default, change the `default` line to point to the new kernel.

It's generally best to hold off on making the new kernel the default until you've tested it. If you make this change too early and then can't get around to fixing problems with the new kernel for a while, you might find yourself accidentally booting the bad kernel. This is normally a minor nuisance.

6. Save your `/etc/lilo.conf` changes.
7. Type **`lilo`** to install LILO in the MBR or boot partition's boot sector.

Real World Scenario

Naming Kernel Files

A good practice when adding a new kernel is to give it a name that includes its version number or other identifying information. For instance, Listing 3.3's kernel is called `bzImage-2.2.17`, identifying it as a 2.2.17 kernel. If you had such a kernel and wanted to try adding, say, the experimental filesystem XFS, you might call this new kernel `bzImage-2.2.17-xfs`. There are no hard-and-fast rules for such naming, so use whatever system you like. As a general rule, though, the base of the name begins with `vmlinux` (for a "raw" kernel file), `vmlinuz` (for a kernel compressed with `gzip`), `zImage` (another name for a kernel compressed with `gzip`), or `bzImage` (for a kernel compressed with `bzip2`). Most distributions use `vmlinuz` for their kernels, but locally compiled kernels usually go by the `zImage` or `bzImage` names. `bzip2` provides greater compression than `gzip` does, and so it is more common with today's large kernels.

Once you've done this, you can reboot the computer to load the new kernel. Be sure to select the new kernel at the `lilo:` prompt, or you'll boot the old one. If everything works, you can go back to step 5 if you skipped it initially (remember to repeat steps 6 and 7, as well). If the new kernel doesn't work properly, you can reboot the computer and select the old kernel in LILO to boot it.

Adding a New OS to LILO

Adding a new OS to LILO works much as does adding a new Linux kernel. There are two basic ways to do this:

Multiple Linuxes You may want to install two or more Linuxes on one computer—say, to have a small emergency system for disaster recovery or to be able to run and test multiple distributions on one computer. When doing this, the procedure is basically the same as that for adding a new kernel, except that you must also specify the correct root partition (with the `root` parameter). In many cases, you'll need to mount your alternate Linux's root partition within the first one's filesystem and point to the alternate system's kernel on this mount point. For instance, when installing an emergency boot system and configuring it from the main Linux system, you might mount the emergency installation's root filesystem at `/emerg`, so the `image` line might read `image=/emerg/boot/bzImage-2.2.17`.

Linux and another OS LILO can boot most non-Linux OSs using the `other` line in `/etc/lilo.conf`, as shown in Listing 3.3. Model the entry for your non-Linux OS after this, pointing to the correct boot partition for the alternate OS.

In either case, once you've saved your changes, you must remember to type **`lilo`**. This action writes a new customized LILO to the MBR or Linux boot partition. If you fail to do this, you'll continue to use the old configuration the next time you boot.

Summary

It is likely that you will use a text-based shell, such as bash, for much of the work you do at a Linux system. A handful of commands can go a long way towards making such a shell useful, giving you the ability to move about the computer's directory, view files, and so on.

One of your primary duties as a system administrator is to manage the packages installed on a computer. To do this, you must often remove unused programs, install new ones, and upgrade existing packages. You may also need to verify the integrity of installed programs or track down what libraries or other programs another one uses. In all these tasks, the RPM and Debian package management systems can be extremely helpful. These systems track installed files and dependencies, giving you access to information that's not otherwise available. On occasion, though, you may need to use the simpler tarballs—particularly if you use a tarball-based distribution like Slackware. Sometimes you can convert between package formats using `alien` or other package conversion tools.

Another of a system administrator's duties is to check the operation of a new package after installing it. Ideally, you can do a test installation on a non-production system before installing the package on your main system.

Perhaps the most important package is the Linux kernel, which presents some unique issues for package management. Many system administrators prefer to compile their own kernels, even when they normally use binary packages. Compiling your own kernel lets you introduce optimizations that influence many other programs.

After you compile your kernel, you run it through the system boot process, which is controlled by a boot loader program such as LILO. LILO is capable of booting many OSs, not just Linux, and of selecting which of several Linux kernels you use.

Exam Essentials

Identify critical features of RPM and Debian package formats. RPM and Debian packages store all files for a given package in a single file that also includes information on what other packages the software depends upon. These systems maintain a database of installed packages and their associated files and dependencies.

Describe the process of installing an RPM or Debian package. Use the `rpm` program to install an RPM package, or use `dpkg` or `apt-get` to install a Debian package. These programs install, upgrade, or remove all files associated with a package and maintain the associated databases.

Describe the ways tarballs differ from RPM or Debian packages. Tarballs contain no dependency information, and the `tar` utility doesn't maintain package databases comparable to those maintained by the RPM or Debian systems.

Summarize procedures for verifying proper program functioning. Ideally, programs should be tested on a disposable test system before being installed on a production system. On both the test and production systems, basic program functioning should be tested by using the program in typical ways before ordinary users can do so.

Explain the Linux kernel version numbering system. Kernel version numbers consist of three components corresponding to less and less important changes. The middle number takes on even values for stable kernels and odd values for development kernels.

Describe the basic function of a boot loader. A boot loader is a low-level program run by the BIOS as a step towards booting an operating system. The common LILO boot loader for Linux can boot a Linux kernel or redirect the boot process to another boot loader stored on a partition's boot sector.

Describe how LILO is configured. LILO uses a configuration file called `/etc/lilo.conf`, which consists of general-purpose lines and one or more stanzas of multiple lines associated with specific Linux kernels or non-Linux OSs. After editing this file, an administrator must type **`lilo`** to install the boot loader in the MBR or Linux boot sector.

Commands in This Chapter

Command	Description
`rpm`	Installs, removes, updates, queries, or verifies packages on an RPM-based Linux distribution
`dpkg`	Installs, removes, updates, queries, or verifies packages on a Debian-based Linux distribution
`apt-get`	Installs, removes, or updates packages on a Debian-based Linux distribution; can automatically retrieve packages from a remote site
`tar`	Adds to, deletes from, or displays the contents of a tarball
`alien`	Converts between Linux package formats
`gnorpm`	GUI front-end to the `rpm` utility
`lilo`	Installs a configured LILO boot loader in the MBR or boot sector

Key Terms

Before you take the exam, be sure you're familiar with these terms:

1024-cylinder limit
binary package
boot loader
boot sector
build number
CHS translation
command prompt
Debian package
dependency
development kernel
extended INT13
filename completion
GNU/Linux
linear block addressing (LBA)
Linux Loader (LILO)
major version number

parameter
primary boot loader
Red Hat Package Manager (RPM)
release kernel
release number
secondary boot loader
shell
source package
source RPM
stable kernel
tarball

Review Questions

1. Which of the following procedures normally launches a shell? (Choose all that apply.)
   - **A.** By starting an xterm window
   - **B.** By typing **`shell`** at a command prompt
   - **C.** By logging in using SSH
   - **D.** You can't; the shell is started automatically at boot time
2. What key does the bash shell use to complete filenames based on the first few characters?
   - **A.** End
   - **B.** Tab
   - **C.** Enter
   - **D.** Insert
3. What features are shared by tarballs, RPMs, and Debian packages? (Choose all that apply.)
   - **A.** They all hold collections of files, including directory structures.
   - **B.** They all provide dependency information.
   - **C.** They all individually compress the files they contain.
   - **D.** They can all be used to store cross-platform files.
4. Which of the following is *not* an advantage of a source package over a binary package?
   - **A.** A single source package can be used on multiple CPU architectures.
   - **B.** By recompiling a source package, you can sometimes work around library incompatibilities.
   - **C.** You can modify the code in a source package, altering the behavior of a program.
   - **D.** Source packages can be installed more quickly than binary packages can.

5. Which is true of using both RPM and Debian package management systems on one computer?

   A. It's generally inadvisable because the two systems don't share installed file database information.

   B. It's impossible because their installed file databases conflict with one another.

   C. It causes no problems if you install important libraries once in each format.

   D. It's a common practice on Red Hat and Debian systems.

6. Which of the following statements is true about binary RPM packages that are built for a particular distribution?

   A. They can often be used on another RPM-based distribution for the same CPU architecture, but this isn't guaranteed.

   B. They may be used in another RPM-based distribution only when using the `--convert-distrib` parameter to `rpm`.

   C. They may be used in another RPM-based distribution only after converting the package with `alien`.

   D. They can be recompiled for an RPM-based distribution running on another type of CPU.

7. Which is true of source RPM packages?

   A. They consist of three files: an original source tarball, a patch file of changes, and a PGP signature indicating the authenticity of the package.

   B. They require programming knowledge to rebuild.

   C. They can sometimes be used to work around dependency problems with a binary package.

   D. They are necessary to compile software for RPM-based distributions.

8. Which of the following do RPM filenames conventionally include?

   A. Single-letter codes indicating Red Hat-certified build sites

   B. Build date information

   C. Version number and CPU architecture information

   D. The initials of the package's maintainer

9. To use `dpkg` to remove a package called `theprogram`, including its configuration files, which of the following commands would you issue?

   A. **`dpkg -P theprogram`**

   B. **`dpkg -p theprogram`**

   C. **`dpkg -r theprogram`**

   D. **`dpkg -r theprogram-1.2.3-4.deb`**

10. Which of the following describes a difference between `apt-get` and `dpkg`?

    A. `apt-get` provides a GUI interface to Debian package management; `dpkg` does not.

    B. `apt-get` can install RPMs and tarballs in addition to Debian packages; `dpkg` can not.

    C. `apt-get` can automatically retrieve and update programs from Internet sites; `dpkg` can not.

    D. `apt-get` is provided only with the original Debian distribution, but `dpkg` comes with Debian and its derivatives.

11. Which of the following is true of an attempt to use a Debian package from one distribution on another Debian-derived distribution?

A. It's unlikely to work because of library incompatibilities and divergent package-naming conventions.

B. It's guaranteed to work because of Debian's strong package definition and enforcement of standards for startup scripts and file locations.

C. It will work only when the distributions are built for different CPUs or when the `alien` package is already installed on the target system.

D. It's likely to work because of the close relationship of Debian-based distributions, assuming the two distributions are for the same CPU architecture.

12. The `tar` program may be used to complete which of the following tasks? (Choose all that apply.)

A. Install RPM and Debian packages.

B. Install software from binary tarballs.

C. Back up a computer to tape.

D. Create source code archive files.

13. `tar` provides a much easier _____ process than do RPM and Debian package tools.

A. Dependency tracking

B. Source code compilation

C. File ownership setting

D. Package creation

14. Which of the following is the default destination format when using `alien`?

A. Tarball

B. RPM

C. Debian package

D. Stampede package

15. Which of the following methods is least likely to be useful for monitoring the operation of a program once you've installed it on a production system?

    A. Checking log entries in the `/var/log` directory tree

    B. Periodically testing the program using test data

    C. Periodically using `ps` to see if a server is still running

    D. Running the program on a test system with a different configuration

16. In which of the following situations does recompiling a Linux kernel yourself make the most sense?

    A. You need to upgrade to a new kernel to obtain a bug fix included in that kernel.

    B. Your system is not recognizing all of its installed memory.

    C. You need to be sure the kernel resides below the 1024-cylinder mark because you have an old BIOS.

    D. You need to include an unusual experimental driver for your system.

17. You're administrating a system that uses a 2.2.4 kernel. Which of the following kernel changes is *least* likely to cause problems?

    A. Upgrading to kernel 2.4.2

    B. Upgrading to kernel 2.3.27

    C. Upgrading to kernel 2.2.17

    D. Applying kernel patches intended for 2.4.2 and recompiling

18. Where may LILO be installed?

    A. The MBR, a Linux partition's boot sector, or a floppy disk

    B. The MBR, a Linux partition's boot sector, or a Windows partition's boot sector

    C. A Linux partition's boot sector or a Windows partition's boot sector

    D. The MBR, a floppy disk, or a swap partition

19. Which of the following is an advantage of installing LILO in a primary Linux partition's boot sector?

A. LILO can then boot a kernel from beyond the 1024-cylinder mark.

B. LILO can then redirect the boot process to other OSs' boot sectors.

C. The DOS or Windows `FDISK` utility can be used to reset LILO as the boot loader if the MBR is overwritten.

D. LILO can work in conjunction with `LOADLIN` to boot multiple kernels.

20. Which of the following is true of the hardware on a system used to test new software?

A. It must always be identical to that of the targeted production system.

B. It is unimportant only when testing new drivers.

C. It is usually less important to make sure it matches the hardware on the production system than it is to make sure that the system software matches.

D. It must be used with identical drivers to that of the production system, even if the hardware differs.

Answers to Review Questions

1. A, C. Shells are started automatically when you log in or start xterm windows unless you configure your account strangely or specify another program to run when you launch an xterm. Typing **`shell`** won't start a shell, because no standard shell is called `shell`. (Typing the shell name will do the job, though.) Shells aren't normally started when the computer boots; you must first log in.

2. B. When you press the Tab key when you are typing a command or filename, bash checks to see if the characters you've typed so far are enough to uniquely identify the command or filename. If they are, bash completes the command or filename, saving you keystrokes.

3. A, D. Tarballs, RPMs, and Debian packages all hold collections of files, which *may* be usable on multiple platforms, although this isn't guaranteed. Tarballs do *not* provide dependency information. All three formats compress an archive of multiple files, rather than archive a set of compressed files.

4. D. Because they must be compiled prior to installation, source packages require *more* time to install than binary packages do.

5. A. Package management systems don't share information, but neither do their databases actively conflict. Installing the same libraries using both systems would almost guarantee that the files served by both systems would conflict with one another. Actively using both RPM and Debian packages isn't common on any distribution, although it's possible with all of them.

6. A. RPMs are usually portable across distributions, but occasionally they contain incompatibilities. There is no `--convert-distrib` parameter to `rpm`, nor is `alien` used to convert from RPM format to RPM format. Binary packages can't be rebuilt for another CPU architecture, but source packages may be rebuilt for any supported architecture, provided the source code doesn't rely on any CPU-specific features.

7. C. Some dependencies result from dynamically linking binaries to libraries at compile time, and so they can be overcome by recompiling the software from a source RPM. Option A describes Debian source packages, not RPM packages. Recompiling a source RPM requires only issuing an appropriate command, although you must also have appropriate compilers and libraries installed. Source tarballs can also be used to compile software for RPM systems, although this results in none of RPM's advantages.

8. C. The package version number (as well as an RPM build number) and CPU architecture code (or `src` for source code or `noarch` for architecture-independent files) are included in most RPM package filenames. Red Hat does not provide certification for RPM maintainers. Build dates and package maintainers' names are stored in the RPM, but not in the filename. (Some distributions include a code for the distribution name in the RPM filename, but this is not a universal practice.)

9. A. An uppercase `-P` invokes the purge operation, which completely removes a package and its configuration files. The lowercase `-p` causes `dpkg` to print information on the package's contents. The `-r` parameter removes a package, but leaves configuration files behind. The final variant (option D) also specifies a complete filename, which isn't used for removing a package—you should specify only the shorter package name.

10. C. You can specify Debian package archive sites in `/etc/apt/sources.list`, then you can type **`apt-get update`** and **`apt-get upgrade`** to quickly update a Debian system to the latest packages. Storm and Corel Linux both ship with GUI package management tools, but they aren't `apt-get`. The `alien` program can convert an RPM file and install the converted package on a Debian system. `dpkg` and `apt-get` both come with all Debian-based distributions.

11. D. In 2001, systems that use Debian are based on the same core OS, and so they share most components, making package transplants likely—but not certain—to succeed. Library incompatibilities *could* cause problems, but aren't likely to, especially if you use recent packages and distributions. Although Debian has clearly defined key file locations, startup scripts, and so on, these can't guarantee success. Binary packages built for *different* CPUs are almost guaranteed *not* to work, although scripts or other non-binary packages most likely will work across CPU types.

12. B, C, D. `tar` can do all these things except for directly installing RPM or Debian packages, although it could be used to do so after converting the package with `alien`.

13. D. The `tar --create` command creates an archive from any specified directory; RPM and Debian package creation tools are more complex than this. `tar` provides no dependency tracking mechanisms at all, making you do that work. Although `tar` can be used to distribute source code, it's not used in compiling it per se. All the package tools discussed in this chapter automatically set file ownership appropriately.

14. C. `alien` can convert to any of these formats, but if you omit the specification, it defaults to a Debian output package.

15. D. A test system can be useful for familiarizing yourself with a new program, but once that's done and the program is up and running on a production system, a test system will not be likely to give you information about the production system that you could not obtain from the production system itself.

16. D. Unusual experimental drivers generally come in source-only form, and therefore they require you to recompile your kernel yourself. Upgrading a kernel may be done via binary RPM or Debian packages. Problems recognizing memory can generally be overcome by `append` options in `lilo.conf`. 1024-cylinder problems can be overcome by placing the kernel in a partition that resides entirely below the 1024-cylinder mark.

17. C. The 2.2.4 and 2.2.17 kernels are part of the same kernel series, and so they will interface with other OS components in the same way. The 2.3.27 kernel is a development kernel in-between the 2.2.*x* and 2.4.*x* kernel series, and so it is likely to be unstable and have some changed interfaces. The 2.4.2 kernel is part of the next kernel series, and so it has some OS interface requirements that may not completely match other software on the system. Patching the 2.2.4 kernel with patches intended for 2.4.2 is unlikely to work correctly, because so many kernel files will have changed.

18. A. LILO may reside in any of the locations listed in Option A. If you install it in a FAT or NTFS partition (used by DOS or Windows), these partitions will be damaged, and if you install LILO in a swap partition that is then used, LILO will be wiped out.

19. C. When installed in the MBR, LILO is susceptible to being completely wiped out by other OSs' installation routines. Installing LILO in a primary Linux partition's boot sector eliminates this risk, making recovery easier. LILO's ability to boot from beyond the 1024-cylinder mark or to boot multiple OSs is identical no matter where it's installed. Likewise, LILO can boot multiple OSs without the use of `LOADLIN` no matter where LILO is installed.

20. C. It's usually more important to match the system software than the hardware when testing new software on a test system. One exception to this is when testing new drivers, in which case matching the hardware is *more* important. Using identical drivers when the hardware doesn't match is generally pointless since the drivers are tied to particular hardware components.

Users and Security

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **4.1 Create and delete users.**
- ✓ **4.2 Modify existing users (e.g., password, groups, personal information).**
- ✓ **4.3 Create, modify and delete groups.**
- ✓ **4.4 Identify and change file permissions, modes and types by using `chmod`, `chown`, and `chgrp`.**
- ✓ **4.7 Perform administrative tasks while logged in as `root`, or by using the `su` command (e.g., understand commands that are dangerous to the system).**
- ✓ **4.9 Describe and use the features of a multi-user environment (e.g., virtual terminals, multiple logins).**
- ✓ **5.8 Monitor system log files regularly for errors, logins, and unusual activity.**
- ✓ **5.11 Perform and verify security best practices (e.g., passwords, physical environments).**
- ✓ **5.12 Assess security risks (e.g., location, sensitive data, file system permissions, remove/disable unused accounts, audit system services/programs).**
- ✓ **5.13 Set daemon and process permissions (e.g., SUID - SGID - Owner/groups).**

Traditional PC operating systems, such as DOS and early versions of Windows, are basically single-user OSs. Although it's certainly possible for two or more people to use computers running these OSs, the OSs themselves provide no mechanisms to help keep users from reading or even damaging each other's files. Linux, on the other hand, is modeled after Unix, which was designed as a multiuser OS. In Linux and Unix, the OS provides tools designed to help keep users from harming each other's files. The same mechanisms are used to provide security and to keep users from damaging the OS as a whole. For these reasons, Linux system administrators must understand how the OS handles users and what tools are available to help you manage the users on your own system.

One particularly important aspect of user configuration is security—but security encompasses additional territory, such as controlling remote access to your system. The Computer Emergency Response Team (CERT; `http://www.cert.org`) tracks security violations (which it calls incidents) and vulnerabilities in software. In 1998, CERT recorded 3,734 incidents and 262 new vulnerabilities; in 1999, the numbers were 9,859 incidents and 417 new vulnerabilities; and in 2000, CERT recorded a total of 21,756 incidents and 774 new vulnerabilities. These increasing numbers probably don't mean that software is becoming shoddier; they most likely reflect the increasing numbers of computers and computer server programs. Nonetheless, the large number of problems emphasizes the need for caution in dealing with computer security. There are many issues related to this topic, ranging from network security to account maintenance to keeping programs up-to-date and installed properly. This chapter covers these issues, in addition to basic user administration.

Linux Multiuser Concepts

Before dealing with the nitty-gritty details of administering user accounts on a Linux system, it's necessary to understand the underlying concepts, including a few implementation details. Understanding this information will help you to plan an effective account structure or expand an existing one to meet new needs. This information may also be critically important when moving accounts from one computer to another, when adding a new hard disk, or when performing other types of system maintenance.

User Accounts: The Core of a Multiuser System

Linux user accounts are basically the same as user accounts in other Unix-like OSs. They allow several people to use the same system, either at different times or at the same time, without interfering with each other. A single user can even have several simultaneous logins active, which is sometimes convenient. It's important to understand what user accounts allow you to do with a system, and also how users are identified.

Accounts in a Multiuser System

Technically, a user is a person, whereas an account is a set of data structures and permissions associated with that user. Frequently, though, the term *user* is used as if it were synonymous with *account*, as in "you must delete this user." Don't take such language literally—delete the account, not the user.

Several important features have been associated with Linux accounts, including the following:

Username The *username* is the name by which the account is known to humans, such as `ellen`. The characteristics of Linux usernames are described in more detail shortly, in "Linux Usernames."

Login privileges An account allows an individual to log into a Linux computer. Depending upon the system's configuration, this could be a login at the console (that is, the keyboard and monitor that are directly connected to the computer) or remotely (via serial line, modem, or network). When an individual logs in, that person may use some or all of the programs and resources available on the computer. Some other resources, like files delivered by a Web server, don't require a login.

Password protection Linux accounts are protected by a password. A person attempting to log in must provide both a username and a password. The username is generally public knowledge, but the password is secret. Some forms of login bypass the password protection, usually by deferring to authentication performed by another computer.

Permissions Every account has permission to run certain programs and access certain files. These permissions are controlled on a file-by-file basis, as described later in this chapter, in "File Permissions."

Home directory Every account has a home directory associated with it. This is a directory in which the user can store data files. Typically, each user has his or her own home directory, although it's possible to configure a system so that two or more users share a home directory. It's also possible, but seldom useful, to specify a home directory to which a user cannot write. (You might use such a configuration if a user should be able to run programs that don't generate their own data but should not be able to store files on the computer.)

User and Group IDs Computers operate on numbers, not words—the words we see on computer screens are encoded as numbers internally. To save storage space, Linux associates two numbers with each account. The first is the *user ID (UID)*, which is mapped to a specific username. The second is the *group ID (GID)*, which is mapped to a specific group of users. Both these processes are described further shortly, in "Mapping UIDs and GIDs to Users and Groups."

Default shell When using a Linux computer at a text-based login (say, at the console without the X Window System running, or via a text-based network protocol like Telnet), Linux presents users with a program known as a shell. The shell accepts commands, such as `ls` and `cd`, and allows the user to run additional programs. Several shells are available for Linux and can be set on an account-by-account basis.

Program-specific files Some programs generate files that are associated with a particular user, in or out of that user's home directory. Many programs create configuration files in the user's home directory, for instance. Another important example is the mail spool, in which a Linux system stores incoming e-mail messages for a user. Assuming the basic mail software is installed, creating a mail account is usually necessary and sufficient for a user to receive mail, although there are exceptions to this rule, particularly with some mail server packages.

Some of these features are defined in one or two critical system configuration files: `/etc/passwd` and `/etc/shadow`. `/etc/passwd` is the traditional repository for most critical account information, including the username, UID number, GID number, password, home directory location, and default shell specification. Creating or modifying an account is mostly a matter of modifying this one file. There are enough additional details, though, that most administrators use special tools to perform these tasks, as described shortly, in "Configuring User Accounts."

Unfortunately, the needs of the system dictate that `/etc/passwd` be readable by all users. This fact makes the placement of password information in `/etc/passwd`–even in encrypted form—a risky proposition. For this reason, most Linux distributions in 2001 ship with *shadow password* support. In this system, users' passwords are stored in a separate file, `/etc/shadow`. This file cannot be read by most users, making it more difficult for a miscreant with an account on the computer to break into other users' accounts.

Accounts in a Multitasking System

Linux is both a multiuser and a multitasking system. Linux's multiuser nature allows multiple people to use one computer without causing problems for one another. Linux's multitasking ability allows multiple programs to run at one time. Although there are single-user multitasking OSs available, combining the two has many advantages, particularly in a networked environment. Specifically, several people can be logged onto a Linux computer at one time, and they can run the same or different programs simultaneously. For instance, Sally can run the Emacs editor while Sam and Ellen both run the Netscape Web browser and George runs the GNU C compiler.

Although it's possible to use a single account for multiple simultaneous logins, using multiple accounts can be very helpful, particularly when multiple individuals are involved. Each account can be configured with its owner's preferences in mind, and therefore, simultaneous logins can present different defaults for things like the placement of icons on a desktop environment or the command shell to be used. Furthermore, if a user changes a default value, that change will not affect other users currently logged on to the system. If the system were a single-user computer that allowed multiple logins, changes to system defaults could adversely affect other users or be undone when other users logged out.

Of course, Linux's multitasking ability doesn't mean that the computer can support an unlimited number of simultaneous users. Some activities,

such as George's C program compilation, are likely to consume a great deal of RAM, CPU time, or disk I/O. If many users try to run such resource-intensive programs simultaneously, all the users will see a performance decrease. Just how many simultaneous users a Linux computer can support depends on many factors, including the types of programs they're likely to run and how much of critical system resources (RAM, CPU speed, network speed, disk speed, and disk capacity) the system has. If the applications used aren't very resource-intensive, a single modern computer can support dozens or hundreds of simultaneous users; but if the programs are hogs of one or more resources, one user per computer may seem like too many.

Simultaneous use of one computer by multiple users generally requires some form of network connectivity, although it can also be handled through terminals connected to serial ports. Typically, remote login protocols like Telnet or the Secure Shell (SSH) allow for text-mode logins. Linux's GUI environment, the X Window System (or X for short), is network-enabled, and so it permits remote use of GUI programs. Alternatively, the VNC program (`http://www.uk.research.att.com/vnc`) allows similar connectivity.

Linux supports multiple simultaneous logins through its standard console through a feature known as *virtual terminals (VTs)*. From a text-mode login, hitting the Alt key along with a function key from 1–6 typically switches to a different virtual screen, and you can log into as many of these as you like. You can even run multiple X sessions at different resolutions by issuing appropriate parameters to `startx`. Ordinarily, the first X session runs on VT 7. When switching out of a VT that's running X, you must add Ctrl to the key sequence—for instance, you must hit Ctrl+Alt+F1 to switch from X to the first text-mode VT. You can run a second X session by logging into a text VT and issuing the following command:

```
$ startx -- :1 vt8
```

This will run X in VT 8. You can switch back and forth between it and the first X session by typing Ctrl+Alt+F7 and Ctrl+Alt+F8.

Of course, this VT capability is most useful for a single-user workstation—two people can't make practical use of the same keyboard at the same time. Nonetheless, it's still useful if you as an administrator want to run Linux under multiple accounts or X configurations or if you want to easily switch between multiple text-based programs without running X.

The Superuser Account

One particularly important account on all Linux systems is that of the *superuser*. The superuser is also referred to as the administrator. The account used by the superuser is normally known as `root`.

Whenever you perform system administration tasks on a Linux computer, you'll do so as `root`. You can do this in any of several ways:

`root` login You can log into the computer as `root`. Thereafter, any action you perform will be done as the superuser. This can be a very dangerous way to use the system, so it's best to do this only for brief periods. Most systems contain restrictions on `root` logins, so they can only be done from the console. This helps prevent outsiders from gaining access to a system over a network by using a stolen password.

`su` The `su` program lets you temporarily acquire superuser privileges or take on any other user's identity. Type **`su`** and press the Enter key after logging on as an ordinary user, and the system will prompt you for the `root` password. If you type that password correctly, subsequent commands will be executed as `root`. Type **`exit`** to return to your normal user privileges. To take on a non-`root` user's privileges, add that user's name, as in **`su george`**, to take on the `george` account's role. If you're already `root`, you can take on another user's identity without that user's password; `su` doesn't ask `root` for a password. This can be useful when debugging problems that may be related to a particular user's configuration.

`sudo` Once configured, the `sudo` command allows you to execute a single command as `root`. This limits the danger of running as `root`, and so it can be a good way to run the programs that you most frequently run as `root`. The `/etc/sudoers` file contains a list of users who may use `sudo`, and with what commands. You can edit this file with the `visudo` command, which invokes the `vi` editor (Chapter 7, "Managing Partitions and Processes") in such a way that it helps you get the format of the configuration file right.

SUID root files As described later in this chapter, in "Interpreting File Access Codes," it's possible to set a file to execute as if run by `root` even when it's run by another user. This must be set on a program-by-program basis.

Program prompts Some configuration tools prompt you for the `root` password and then run themselves as `root`. This setup is most common with the GUI configuration tools that ship with many Linux distributions.

The root account is special because it bypasses normal security features. Specifically, the superuser may read, write, or delete any file on the computer, no matter who owns that file or whether the owner has granted other users read or write access to it. This sort of power is dangerous not just because of the ability to invade other users' privacy, but because it allows root to do serious damage to the computer. For instance, suppose you want to delete a directory and its contents. You might issue the following command to do so:

```
# rm -r /home/george/olddir
```

This command deletes the /home/george/olddir directory and all its files and subdirectories. Unfortunately, a single typo can create a much more destructive command:

```
# rm -r / home/george/olddir
```

Note the stray space between / and home/george/olddir. This typo causes the computer to delete all files in the / directory—that is, all files on the computer as well as files in home/george/olddir. This is the sort of power that you should grant yourself only when you absolutely need it.

Linux Usernames

Linux is fairly flexible about its usernames. Most versions of Linux support usernames consisting of any combination of upper- and lowercase letters, numbers, and many punctuation symbols, including periods and spaces. Some punctuation symbols, however, such as spaces, cause problems for certain Linux utilities, so it's generally best to avoid using punctuation in Linux usernames. Underscores (_) and periods (.) are relatively unlikely to cause problems and so are occasionally used. Also, usernames must begin with a letter, so a username such as 45u is invalid, although u45 is fine. Although usernames may be up to 32 characters in length, many utilities truncate usernames longer than eight characters or so in their displays, so many administrators try to limit username length to eight characters.

Linux treats usernames in a case-sensitive way. Therefore, a single computer can support both ellen and Ellen as separate users. This practice can lead to a great deal of confusion, however, so it's best to avoid creating accounts whose usernames differ only in case. In fact, the traditional practice is to use entirely lowercase letters in Linux usernames, such as sally, sam, ellen, and george. Usernames don't need to be based on first names, of

course—you could use `sam_jones`, `s.jones`, `sjones`, `jones`, `jones17`, or `u238`, to name just a few possibilities. Most sites develop a standard method of creating usernames, such as using the first initial and the last name. Creating and following such a standard practice can help you locate an account that belongs to a particular individual. If your computer has many users, though, you may find a naming convention produces duplicates, particularly if your standard uses initials to shorten usernames. You may therefore be forced to deviate from the standard or incorporate numbers to distinguish between all the Davids or Smiths of the world.

Groups: Linking Users Together for Productivity

Linux uses *groups* as a means of organizing users. In many ways, groups parallel users. Groups are similar to users in ways such as the following:

- Groups are defined in a single file, `/etc/group`, which has a structure similar to that of `/etc/passwd`.
- Groups have names similar to usernames.
- Group names are tied to group IDs (GIDs).

Groups are *not* accounts, however. Rather, groups are a means of organizing collections of accounts, largely as a security measure. As described shortly, in "File Permissions," every file on a Linux system is associated with a specific group, and various permissions can be assigned to members of that group. For instance, group members (such as faculty at a university) might be allowed to read a file, but others (such as students) might be disallowed such access. Because Linux provides access to hardware (such as serial ports and tape backup units) through files, this same mechanism can be used to control access to hardware.

Every group has anywhere from no members to as many members as there are users on the computer. Group membership is controlled through the `/etc/group` file. This file contains a list of groups and the members belonging to each group. The details of this file's contents are described later in this chapter, in "Configuring Groups."

In addition to membership defined in `/etc/group`, each user has a default or primary group. The user's primary group is set in the user's configuration in `/etc/passwd`. When users log onto the computer, their group membership is set to their primary groups. When users create files or launch programs, those files and running programs are associated with a single group—the current group membership. A user can still access files belonging to other

groups, so long as the user belongs to that group and the group access permissions allow the access. To run programs or create files with other than the primary group membership, however, the user must run the `newgrp` command to switch current group membership. For instance, to change to the `project2` group, you might type the following:

```
$ newgrp project2
```

If the user typing this command is listed as a member of the `project2` group in `/etc/group`, the user's current group membership will change. Thereafter, files created by that user will be associated with the `project2` group. Alternatively, users can change the group associated with an existing file by using the `chgrp` or `chown` commands, as described shortly, in "Changing File Permissions."

This group structure allows you to design a security system that permits different collections of users to easily work on the same files while simultaneously keeping other users of the same computer from prying into files they should not be able to access. In a simple case, you might create groups for different projects, classes, or workgroups, with each user restricted to one of these groups. A user who needs access to multiple groups could be a member of each of these groups—for instance, a student who takes two classes could belong to the groups associated with each class, or a supervisor might belong to all the supervised groups. "Common User and Group Strategies," later in this chapter, describes the approaches taken by various Linux distributions by default, and it then explains how you can expand and use these strategies to suit your own needs.

Mapping UIDs and GIDs to Users and Groups

As mentioned earlier, Linux defines users and groups by numbers (UIDs and GIDs, respectively). Internally, Linux tracks users and groups by these numbers, not by name. For instance, the user `sam` might be tied to UID 523, and `ellen` might be UID 609. Similarly, the group `project1` might be GID 512, and `project2` might be GID 523. For the most part, these details take care of themselves—you use names, and Linux uses `/etc/passwd` or `/etc/group` to locate the number associated with the name. You may occasionally need to know how Linux assigns numbers when you tell it to do something, though. This is particularly true when you are troubleshooting or if you have cause to manually edit `/etc/passwd` or `/etc/group`.

Linux distributions reserve the first hundred user and group IDs (0–99) for system use. The most important of these is 0, which corresponds to `root`

(both the user and the group). Subsequent low numbers are used by accounts and groups that are associated with specific Linux utilities and functions. For instance, UID 2 and GID 2 are generally the `daemon` account and group, respectively, which are used by various servers; and UID 8 and GID 12 are usually the `mail` account and group, which can be used by mail-related servers and utilities. Not all account and group numbers from 0–99 are in use; there are usually only one or two dozen accounts and a dozen or so groups used in this way. You can check your `/etc/passwd` and `/etc/group` files to determine which user and group IDs are so used.

Beyond 100, user and group IDs are available for use by ordinary users and groups. Many distributions, however, reserve up to 500 or even 1000 for special purposes. Frequently, therefore, the first normal user account is assigned a UID of 500 or 1000. When you create additional accounts, the system typically locates the next-highest unused number, so the second user you create is UID 501, the third is 502, and so on. When you remove an account, that account's ID number may be reused, but the automatic account-creation tools typically don't do so if subsequent numbers are in use, leaving a gap in the sequence. This gap causes no harm unless you have so many users that you run out of ID numbers. (The limit is 65,536 users with the 2.2.*x* kernels and over 4.2 billion with the 2.4.*x* kernels, including `root` and other system accounts. The limit can be set lower in configuration files or because of limits in support programs.) In fact, reusing an ID number can cause problems if you don't clear away the old user's files—the new user will become the owner of the old user's files, which can lead to confusion.

Typically, GID 100 is `users`–the default group for some distributions. (See "Common User and Group Strategies" later in this chapter.) On any but a very small system with few users, you'll probably want to create your own groups. Because different distributions have different default ways of assigning users to groups by default, it's best that you familiarize yourself with your distribution's way of doing this, and plan your own group-creation policies with this in mind. For instance, you might want to create your own groups within certain ranges of IDs to avoid conflicts with the distribution's default user- and group-creation processes.

Intruders sometimes create accounts with UID 0 to give themselves `root` privileges on the systems they invade. *Any* account with a UID of 0 is effectively the `root` account, with all the power of the superuser. If you spot a suspicious account in your `/etc/passwd` file with a UID of 0, your system has most probably been compromised.

It's possible to create multiple usernames that use the same UID, or multiple group names that use the same GID. In some sense, these are different accounts or groups; they have different entries in `/etc/passwd` or `/etc/group`, so they can have different home directories, different passwords, and so on. Because these users or groups share IDs with other users or groups, though, they're treated identically in terms of file permissions. Unless you have a compelling reason to do so, you should avoid creating multiple users or groups that share an ID.

Real World Scenario

Coordinating UIDs and GIDs across Systems

If you maintain several Linux computers and want to set up Network Filesystem (NFS) file sharing, one problem that can arise is keeping UIDs and GIDs synchronized across systems. Because all Linux filesystems, including NFS, track numeric IDs rather than the names that humans use, mismatched UIDs and GIDs can cause one person's files to appear to be owned by another person on an NFS mount. For instance, suppose that two computers each have two users, `ellen` and `george`. On one computer, `ellen` has UID 500 and `george` has UID 501, but these numbers are reversed on the other. As a consequence, when one computer mounts the other's files via NFS, the UID values will indicate that `ellen` owns files that are really owned by `george`, and vice-versa.

One solution to this problem is to keep UIDs and GIDs consistent across computers. This isn't too difficult with a handful of small systems with few users, but it becomes tedious with larger or more systems. NFS also supports various mapping options, such as querying a remote login database, using a static map file, or using a user ID mapping server run on the client system. These options are described in the `exports` man page.

The Importance of Home Directories

A user's *home directory* is a directory on the disk that's usually intended for one user alone. On Linux systems, the standard placement of home directories is in the `/home` directory tree, with each user's home directory named after the user's account name. For instance, the home directory for the `sally`

account would be /home/sally. This naming and placement is only a convention, though—it's not a requirement. The /etc/passwd file contains the location of each user's home directory, so you can modify this location by editing that file. You can also specify an alternative location when you create an account (as described shortly in "Adding Users"), or use the usermod utility to change it after the fact.

Typically, a user's home directory belongs to that user only. Therefore, it's created with fairly restrictive permissions, particularly for writing to the directory. The exact permissions used by default vary from one distribution to another, so you should check yours to see how it's done. If you want to create more stringent or laxer permissions, you'll have to do so yourself after creating an account, or you'll need to create your own account-creation scripts to automate the process.

You can create separate directories for shared projects, if you like. For instance, you might want to have a directory in which group members can store files that belong to the group as a whole, or in which group members may exchange files. Linux distributions don't create such directories automatically when creating groups, so you'll have to attend to this task yourself, as well as decide where to store them. (Somewhere in /home is a logical choice, but it is up to you.)

One problem that's commonly faced by Linux system administrators is the depletion of available disk space. The /home directory frequently resides on a separate partition, and sometimes an entirely separate physical hard disk, from other Linux files. This arrangement can help to make the system more secure because it helps to isolate the data—filesystem corruption on one partition need not affect data on another. It also limits room for expansion, however. If your users begin creating very large files, or if the number of users you must support grows and causes your initial estimates of required /home disk space to be exceeded, you'll need to take action to correct this matter. For instance, you might move home directories to some other partition, enlarge the home partition with a tool like resize2fs or PartitionMagic (http://www.powerquest.com), or add a new hard disk to store some or all of the user home directories.

File Permissions

Support for individual user accounts and groups is wasted if it's not used. In Linux, these features are applied in various ways, the most obvious

and important from a system administration point of view being file permissions. (User and group information is also associated with running programs and may be used by specific programs—say to locate and load a specific user's settings.) File permissions are at the heart of Linux's local security configuration, as discussed later in this chapter. In order to use these features, though, you must understand how Linux treats file permissions, and what tools the OS provides to allow you to manipulate permissions.

File Access Permissions

File access permissions in Linux involve several components, which combine to determine who may access a file and in what way. These components also help to determine precisely what a file is—an ordinary data file, a program file, a subdirectory, or so on. Understanding this setup is necessary if you're to manipulate file permissions.

File Access Components

There are three components to Linux's file permission handling:

Username (or UID) A username (or UID, as it's stored in this form) is associated with each file on the computer. This is frequently referred to as the *file owner*.

Group (or GID) Every file is associated with a particular GID, which links the file to a group. This is sometimes referred to as the *group owner*. Normally, the group of a file is one of the groups to which the file's owner belongs, but `root` may change the file's group to one unassociated with the file's owner.

File access permissions The *file access permissions* (or *file permissions* for short) are a code that represents who may access the file, relative to the file's owner, the file's group, and all other users.

You can see all three elements by using the **`ls -l`** command on a file, as shown here:

```
$ ls -l /usr/sbin/lsof
-rwxr-xr-x  1 root   kmem  84124 Oct  3 02:37
↳/usr/sbin/lsof
```

The output of this command has several different components, each with a specific meaning:

Permission string The first component, `-rwxr-xr-x`, is the permission string. Along with the user and group names, it's what determines who may access a file. As displayed by `ls -l`, the permission string is a series of codes, which are described in more detail shortly, in "Interpreting File Access Codes." Sometimes the first character of this string is omitted, particularly when discussing ordinary files, but it's always present in an `ls -l` listing.

Number of hard links Linux supports *hard links* in its filesystems. A hard link allows one file to be referred to by two or more different filenames. Internally, Linux uses a data structure known as an *inode* to keep track of the file, and multiple filenames point to the same inode. The number `1` in the preceding example output means that just one filename points to this file; it has no hard links. Larger numbers indicate that hard links exist—for instance, `3` means that the file may be referred to by three different filenames.

Hard links and *soft* (or *symbolic*) *links* are handled differently. There's no way to tell if a file has soft links from its directory listing alone, although the soft links to a file are indicated by an `l` in the first character of the permission string.

Owner The next field, `root` in this example, is the owner of the file. In the case of long usernames, the username may be truncated.

Group `kmem` is the group to which the example file belongs. Many system files belong to the `root` owner and `root` group; for this example, I picked a file that belongs to a different group.

File size The next field, `84124` in this example, is the size of the file in bytes.

Creation time The next field contains the file creation time and date (`Oct 3 02:37` in this example). If the file is older than a year, you'll see the year rather than the creation time, although the time is still stored with the file.

Filename The final field is the name of the file. Because the `ls` command in the preceding example specified a complete path to the file, the complete path appears in the output. If the command had been issued without that path but from the `/usr/sbin` directory, `lsof` would appear alone.

Although information such as the number of hard links and file creation date may be useful on occasion, it's not critical for determining file access rights. For this, you need the file's owner, group, and file access permission string.

Interpreting File Access Codes

The file access control string is ten characters in length. The first character has special meaning—it's the *file type code*. The type code determines how Linux will interpret the file—as ordinary data, a directory, or a special file type. Table 4.1 summarizes Linux type codes.

TABLE 4.1 Linux File Type Codes

Code	Meaning
-	Normal data file; may be text, an executable program, graphics, compressed data, or just about any other type of data.
d	Directory; disk directories are files just like any others, but they contain filenames and pointers to disk inodes.
l	Symbolic link; the file contains the name of another file or directory. When Linux accesses the symbolic link, it tries to read the linked-to file.
p	Named pipe; a pipe allows two running Linux programs to communicate with each other. One opens the pipe for reading, and the other opens it for writing, allowing data to be transferred between the programs.
s	Socket; a socket is similar to a named pipe, but it permits network and bidirectional links.

TABLE 4.1 Linux File Type Codes *(continued)*

Code	Meaning
b	Block device; a hardware device to and from which data is transferred in blocks of more than one byte. Disk devices (hard disks, floppies, CD-ROMs, and so on) are common block devices.
c	Character device; a hardware device to and from which data is transferred in units of one byte. Examples include parallel and serial port devices.

The remaining nine characters of the permission string (`rwxr-xr-x` in the preceding example) are broken up into three groups of three characters. The first group controls the file owner's access to the file, the second controls the group's access to the file, and the third controls all other users' access to the file (often referred to as world permissions).

In each of these three cases, the permission string determines the presence or absence of each of three types of access: read, write, and execute. Read and write permissions are fairly self-explanatory, at least for ordinary files. If the execute permission is present, it means that the file may be run as a program. (Of course, this doesn't turn a non-program file into a program; it only means that a user may run a program if it is a program. Setting the execute bit on a non-program file will probably cause no real harm, but it could be confusing.) The absence of the permission is denoted by a hyphen (`-`) in the permission string. The presence of the permission is indicated by a letter—`r` for read, `w` for write, or `x` for execute.

Thus, the example permission string of `rwxr-xr-x` means that the file's owner, members of the file's group, and all other users can read and execute the file. Only the file's owner has write permission to the file. You can easily exclude those who don't belong to the file's group, or even all but the file's owner, by changing the permission string, as described shortly.

Individual permissions, such as execute access for the file's owner, are often referred to as *permission bits*. This is because Linux encodes this information in binary form. Because it is binary, the permission information can be expressed as a single 9-bit number. This number is usually expressed in octal (base 8) form because a base-8 number is three bits in length, which means that the base-8 representation of a permission string is three digits long, one digit for each of the owner, group, and world permissions. The

read, write, and execute permissions each correspond to one of these bits. The result is that you can determine owner, group, or world permissions by adding base-8 numbers: 1 for execute permission, 2 for write permission, and 4 for read permission. Table 4.2 shows some examples of common permissions and their meanings. This table is necessarily incomplete, though; with nine permission bits, the total number of possible permissions is 2^9, or 512. Most of those possibilities are very peculiar, and you're not likely to encounter or create them except by accident.

TABLE 4.2 Example Permissions and Their Likely Uses

Permission string	Octal code	Meaning
`rwxrwxrwx`	777	Read, write, and execute permissions for all users.
`rwxr-xr-x`	755	Read and execute permission for all users. The file's owner also has write permission.
`rwxr-x---`	750	Read and execute permission for the owner and group. The file's owner also has write permission. Non-group members have no access to the file.
`rwx------`	700	Read, write, and execute for the file's owner only; all others have no access.
`rw-rw-rw-`	666	Read and write permissions for all users. No execute permissions to anybody.
`rw-rw-r--`	664	Read and write permissions to the owner and group. Read-only permission to all others.
`rw-rw----`	660	Read and write permissions to the owner and group. No world permissions.
`rw-r--r--`	644	Read and write permissions to the owner. Read-only permission to all others.

TABLE 4.2 Example Permissions and Their Likely Uses *(continued)*

Permission string	Octal code	Meaning
`rw-r-----`	640	Read and write permissions to the owner, and read-only permission to the group. No permission to others.
`rw-------`	600	Read and write permissions to the owner. No permission to anybody else.
`r--------`	400	Read permission to the owner. No permission to anybody else.

Execute permission makes sense for ordinary files, but it's meaningless for most other file types, such as device files. Directories, though, make use of the execute bit in another way. When a directory's execute bit is set, that means that the directory's contents may be searched. This is a highly desirable characteristic for directories, so you'll almost never find a directory on which the execute bit is *not* set in conjunction with the read bit.

Directories can be confusing with respect to write permission. Recall that directories are files that are interpreted in a special way. As such, if a user can write to a directory, that user can create, delete, or rename files in the directory, even if the user isn't the owner of those files and does not have permission to write to those files. The *sticky bit* (described shortly) can alter this behavior.

Symbolic links are unusual with respect to permissions. This file type always has 777 (`rwxrwxrwx`) permissions, thus granting all users full access to the file. This access applies only to the link file itself, however, not to the linked-to file. In other words, all users can read the contents of the link to discover the name of the file to which it points, but the permissions on the linked-to file determine its file access. Attempting to change the permissions on a symbolic link affects the linked-to file.

Many of the permission rules do not apply to `root`. The superuser can read or write any file on the computer—even files that grant access to nobody (that is, those that have 000 permissions). The superuser still needs an execute bit to be set to run a program file, but the superuser has the power to change the permissions on any file, so this limitation isn't very substantial.

Some files may be inaccessible to `root` but only because of an underlying restriction—for instance, even `root` can't access a hard disk that's not installed in the computer.

There are a few special permission options that are also supported, and they may be indicated by changes to the permission string:

Set user ID (SUID) The *set user ID (SUID)* option is used in conjunction with executable files, and it tells Linux to run the program with the permissions of whoever owns the file, rather than with the permissions of the user who runs the program. For instance, if a file is owned by `root` and has its SUID bit set, the program runs with `root` privileges and can therefore read any file on the computer. Some servers and other system programs run in this way, which is often called SUID `root`. SUID programs are indicated by an `s` in the owner's execute bit position of the permission string, as in `rwsr-xr-x`. (As discussed later in this chapter, SUID programs can pose a security risk.)

Set group ID (SGID) The *set group ID (SGID)* option is similar to the SUID option, but it sets the group of the running program to the group of the file. It's indicated by an `s` in the group execute bit position of the permission string, as in `rwxr-sr-x`.

Sticky bit The sticky bit has changed meaning during the course of Unix history. In modern Linux implementations (and most modern versions of Unix), it's used to protect files from being deleted by those who don't own the files. When this bit is present on a directory, the directory's files can only be deleted by their owners, the directory's owner, or `root`. The sticky bit is indicated by a `t` in the world execute bit position, as in `rwxr-xr-t`.

Changing File Ownership and Permissions

Changing who can read, write, or execute a file can be done using several programs, depending upon the desired effect. Specifically, `chown` changes a file's owner, and optionally, its group; `chgrp` changes the file's group; and `chmod` changes the permissions string.

Ownership Modification

To begin, `chown`'s syntax is as follows:

```
chown [options] [newowner][.newgroup] filename...
```

newowner and *newgroup* are, of course, the new owner and group of the file. One or both are required. If both are included, there must be no space

between them, only a single period (.). For instance, the following command gives ownership of the file report.tex to sally, and sets the file's group to project2:

```
# chown sally.project2 report.tex
```

The chown command supports a number of options, such as --dereference (which changes the referent of a symbolic link) and --recursive (which changes all the files within a directory and all its subdirectories). The latter is probably the most useful option for chown.

The chgrp command is similar to chown, except that it doesn't change or alter the file's owner—it works only on the group. The group name is *not* preceded by a period. For instance, to change the group of report.tex to project2, you could issue the following command:

```
# chgrp project2 report.tex
```

chgrp takes the same options as does chown. One caveat to the use of both commands is that even the owner of a file may not be able to change the ownership or group of a file. The owner may change the group of a file to any group to which the file's owner belongs, but not to other groups. Normally, only root may change the owner of a file.

Permissions Modification

You can modify a file's permissions using the chmod command. This command may be issued in many different ways to achieve the same effect. Its basic syntax is as follows:

```
chmod [options] [mode[,mode...]] filename...
```

The chmod options are similar to those of chown and chgrp. In particular, --recursive (or -R) will change all the files within a directory tree.

Most of the complexity of chmod comes in the specification of the file's *mode*—that is, its permissions. There are two basic forms in which you can specify the mode: as an octal number or as a symbolic mode, which is a set of codes related to the string representation of the permissions.

The octal representation of the mode is the same as that described earlier and summarized in Table 4.2. For instance, to change permissions on report.tex to rw-r--r--, you could issue the following command:

```
# chmod 644 report.tex
```

In addition, it's possible to precede the three digits for the owner, group, and world permissions with another digit that sets special permissions. Three bits are supported (hence values between 0 and 7): adding 4 sets the set user ID (SUID) bit; adding 2 sets the set group ID (SGID) bit; and adding 1 sets the sticky bit. If you omit the first digit (as in the preceding example), Linux clears all three bits. Using four digits causes the first to be interpreted as the special permissions code. For instance, the following command sets the program file `bigprogram` to have both SUID and SGID bits set (6), to be readable, writeable, and executable by the owner (7), to be readable and executable by the group (5), and completely inaccessible to all others (0):

```
# chmod 6750 bigprogram
```

A symbolic mode, by contrast, consists of three components: a code indicating the permission set you want to modify (the owner, the group, and so on); a symbol indicating whether to add, delete, or set the mode equal to the stated value; and a code specifying what the permission should be. Table 4.3 summarizes all these codes. Note that these codes are all case-sensitive.

TABLE 4.3 Codes Used in Symbolic Modes

Permission set code	Meaning	Change type code	Meaning	Permission to modify code	Meaning
u	owner	+	add	r	read
g	group	-	remove	w	write
o	world	=	set equal to	x	execute
a	all			X	execute only if file is directory or already has execute permission
				s	SUID or SGID
				t	sticky bit

TABLE 4.3 Codes Used in Symbolic Modes *(continued)*

Permission set code	Meaning	Change type code	Meaning	Permission to modify code	Meaning
				u	existing owner's permissions
				g	existing group permissions
				o	existing world permissions

To use symbolic permission settings, you combine together one or more of the codes from the first column of Table 4.3 with one symbol from the third column and one or more codes from the fifth column. You can combine multiple settings by separating them by commas. Table 4.4 provides some examples of `chmod` using symbolic permission settings.

TABLE 4.4 Examples of Symbolic Permissions with `chmod`

Command	Initial Permissions	End Permissions
`chmod a+x bigprogram`	`rw-r--r--`	`rwxr-xr-x`
`chmod ug=rw report.tex`	`r--------`	`rw-rw----`
`chmod o-rwx  bigprogram`	`rwxrwxr-x`	`rwxrwx---`
`chmod g=u report.tex`	`rw-r--r--`	`rw-rw-r--`
`chmod g-w,o-rw report.tex`	`rw-rw-rw-`	`rw-r-----`

As a general rule, symbolic permissions are most useful when you want to make a simple change (such as adding execute or write permissions to one or more class of users), or when you want to make similar changes to many files without affecting their other permissions—for instance, adding write permissions without affecting execute permissions. Octal permissions are most

useful when you want to set some specific absolute permission, such as `rw-r--r--` (644). In any event, a system administrator should be familiar with both methods of setting permissions.

A file's owner and `root` are the only users who may adjust a file's permissions. Even if other users have write access to a directory in which a file resides and write access to the file itself, they may not change the file's permissions (but they may modify or even delete the file). To understand why this is so, you need to know that the file permissions are stored as part of the file's inode, which isn't part of the directory entry. Read/write access to the directory entry, or even the file itself, doesn't give a user the right to change the inode structures (except indirectly—for instance, if a write changes the file's size or a file deletion eliminates the need for the inode).

Setting Default Permissions

When a user creates a file, that file has default ownership and permissions. The default owner is, understandably, the user who created the file. The default group is the user's primary group, as described earlier. The default permissions, however, are configurable. These are defined by the *user mask (umask)*, which is set by the `umask` command. This command takes as input an octal value that represents the bits to be removed from 777 permissions for directories, or from 666 permissions for files, when creating a new file or directory. For instance, to have Linux create files with 640 permissions, and 0750 for directories, you would enter the following command:

```
$ umask 027
```

Note that the umask isn't a simple subtraction from the values of 777 or 666; it's a bit-wise removal. Any bit that's set in the umask is removed from the final permission for new files, but if the execute bit isn't set (as in ordinary files), its specification in the umask doesn't do any harm. For instance, consider the trailing `7` in the preceding `umask` command. This corresponds to a binary value of 111. An ordinary file might have `rw-` (110) permissions, but applying the umask's 7 (111) eliminates 1 values but doesn't touch 0 values, thus producing a 000 (binary) value—that is, `---` permissions, expressed symbolically.

Ordinary users can enter the `umask` command to change the permissions on new files they create. The superuser can also modify the default setting for all users by modifying a system configuration file. Typically, `/etc/profile`

contains one or more `umask` commands. Setting the umask in `/etc/profile` might or might not actually have an effect, because it can be overridden at other points, such as a user's own configuration files. Nonetheless, setting the umask in `/etc/profile` or other system files can be a useful procedure if you want to change the default system policy. Most Linux distributions use a default umask of 002 or 022.

To find what the current umask is, type **`umask`** alone, without any parameters. Typing **`umask -S`** produces the umask expressed symbolically, rather than in octal form. You may also specify a umask in this way when you want to change it, but in this case, you specify the bits that you *do* want set. For instance, **`umask u=rwx,g=rx,o=rx`** is equivalent to **`umask 022`**.

Configuring User Accounts

How frequently you'll do user maintenance depends on the nature of the system you administer. Some systems, such as small personal workstations, will need changes very rarely. Others, such as large systems in environments in which users are constantly coming and going, may require daily maintenance. The latter situation would seem to require more knowledge of user account configuration tools, but even in a seldom-changing system, it's useful to know how to do these things so that you can do them quickly and accurately when you do need to add, modify, or delete user accounts.

Adding Users

Adding users can be accomplished through the `useradd` utility. (This program is called `adduser` on some distributions.) Its basic syntax is as follows:

```
useradd [-c comment] [-d home-dir] [-e expire-date]
↳[-f inactive-time] [-g initial-group] [-G group[,...]]
↳[-m [-k skeleton-dir] | -M] [-p password] [-s shell]
↳[-u UID [-o]] [-r] [-n] username
```

Some of these parameters modify settings that are valid only when the system uses shadow passwords. This is the standard configuration for most distributions today.

In its simplest form, you may type just **useradd *username***, where *username* is the username you want to create. The rest of the parameters are used to modify the default values for the system, which are stored in the file `/etc/login.defs`. The parameters and their meanings are as follows:

`-c` *`comment`* The comment field for the user. Some administrators store public information like a user's office or telephone number in this field. Others store just the user's real name, or no information at all.

`-d` *`home-dir`* The account's home directory. This defaults to `/home/`*`username`* on most systems.

`-e` *`expire-date`* The date on which the account will be disabled, expressed in the form `YYYY-MM-DD`. (Many systems will accept alternative forms, such as `MM-DD-YYYY`, or a single value representing the number of days since January 1, 1970, as well.) The default is for an account that does not expire. This option is most useful in environments in which user accounts are inherently time-limited, such as accounts for students taking particular classes or temporary employees.

`-f` *`inactive-days`* The number of days after a password expires after which the account becomes completely disabled. A value of `-1` disables this feature, and is the default.

`-g` *`default-group`* The name or GID of the user's default group. The default for this value varies from one distribution to another, as described later, in "Common User and Group Strategies."

`-G` *`group`*`[,...]` The names or GIDs of one or more groups to which the user belongs. These groups need not be the default group, and more than one may be specified by separating them with commas.

`-m [-k` *`skeleton-dir`*`]` The system automatically creates the user's home directory if `-m` is specified. Normally, default configuration files are copied from `/etc/skel`, but you may specify another template directory with the `-k` option. Many distributions use `-m` as the default when running `useradd`.

`-M` This option forces the system to *not* automatically create a home directory, even if `/etc/login.defs` specifies that this action is the default.

`-p` *`encrypted-password`* This parameter passes the *pre-encrypted* password for the user to the system. The *`encrypted-password`* value will be added, *unchanged*, to the `/etc/passwd` or `/etc/shadow` file. This means that if you type an unencrypted password, it won't work as you probably expected. In practice, this parameter is most useful in scripts, which can encrypt a password (using `crypt`) and then send the encrypted result through `useradd`. The default value disables the account, so you must run `passwd` to change the user's password.

`-s` *`shell`* The name of the user's default login shell. On most systems, this defaults to `/bin/bash`, but Linux supports many alternatives, such as `/bin/tcsh` and `/bin/zsh`.

`-u` *`UID`* `[-o]` Creates an account with the specified user ID value (*`UID`*). This value must be a positive integer, and it is normally above 500 for user accounts. System accounts typically have numbers below 100. The `-o` option allows the number to be reused so that two usernames are associated with a single UID.

`-r` This parameter allows the creation of a system account—an account with a value lower than `UID_MIN`, as specified in `/etc/login.defs`. (This is normally 100, 500, or 1000.) `useradd` also doesn't create a home directory for system accounts.

`-n` In some distributions, such as Red Hat, the system creates a group with the same name as the specified username. This parameter disables this behavior.

Suppose you've added a new hard disk in which some users' home directories are located and mounted it as `/home2`. You want to create a user account in this directory and make the new user a member of the `project1` and `project4` groups, with default membership in `project4`. The user has also requested `tcsh` as her default shell. You might use the following commands to accomplish this goal:

```
# useradd -d /home2/sally -g project4 -G project1,project4
↳-s /bin/tcsh sally
# passwd sally
Changing password for user sally
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

The `passwd` command asks for the password twice, but it does not echo what you type. This prevents somebody who sees your screen from reading the password off of it. `passwd` is discussed in more detail in "Modifying User Accounts."

Modifying User Accounts

User accounts may be modified in many different ways: You can directly edit critical files such as `/etc/passwd`, modify user-specific configuration files in the account's home directory, or use system utilities like those used to create accounts. You usually modify an existing user's account at the user's request or to implement some new policy or system change, such as moving home directories to a new hard disk. Sometimes, though, you must modify an account immediately after its creation, in order to customize it in ways that aren't easily handled through the account creation tools or because you realize you forgot a parameter to `useradd`.

Setting a Password

Although `useradd` provides the `-p` parameter to allow you to set a password, this tool is not very useful when directly adding a user because it requires a pre-encrypted password. Therefore, it's usually easiest to create an account in disabled form (by not using `-p` with `useradd`) and set the password after creating the account. This can be done with the `passwd` command, which has the following syntax:

```
passwd [-k] [-l] [-u [-f]] [-d] [-S] [username]
```

The parameters to this command have the following meanings:

`-k` This parameter indicates that the system should update an expired account.

`-l` This parameter locks an account, by prefixing the encrypted password with an exclamation mark (`!`). The result is that the user can no longer log into the account, but the files are still available and the change can be easily undone.

-u [-f] The -u parameter unlocks an account by removing a leading exclamation mark. useradd creates accounts that are locked and have no password, so using this command on a fresh account would result in an account with no password. Normally, passwd doesn't allow this—it returns an error if you attempt it. Adding -f forces passwd to turn the account into one with no password.

-d This parameter removes the password from an account, rendering it password-less.

-S This option displays information on the password for an account—whether or not it's set and what type of encryption it uses.

Ordinary users may use passwd to change their passwords, but many passwd parameters may only be used by root. Specifically, -l, -u, -f, -d, and -S are all off-limits to ordinary users. Similarly, only root may specify a username to passwd. When ordinary users run the program, they should omit their usernames, and passwd will change the password for the user who ran the program. As a security measure, passwd asks for a user's old password before changing the password when an ordinary user runs the program. This precaution is *not* taken when root runs the program so that the superuser may change a user's password without knowing it. Since the administrator normally doesn't know the user's password, this is necessary.

Linux passwords may consist of letters, numbers, and punctuation. Linux distinguishes between upper- and lowercase letters in passwords, which means you can use mixed-case passwords to improve security.

"Account Security" later in this chapter includes additional information on selecting good passwords.

Using *usermod*

The usermod program closely parallels useradd in its features and parameters. This utility changes an existing account rather than create a new one, however. The major differences between useradd and usermod are as follows:

- usermod allows the addition of a -m parameter when used with -d. -d alone changes the user's home directory, but it does not move any files. Adding -m causes usermod to move the user's files to the new location.

- `usermod` supports a `-l` parameter, which changes the user's login name to the specified value. For instance, **`usermod sally -l sjones`** changes the username from `sally` to `sjones`.
- You may lock or unlock a user's password with the `-L` and `-U` options, respectively. This duplicates functionality provided by `passwd`.

`usermod` changes the contents of `/etc/passwd` or `/etc/shadow`, depending upon the option used. If `-m` is used, `usermod` also moves the user's files, as already noted.

Changing an account's characteristics while the owner is logged in can have undesirable consequences. This is particularly true of the `-d -m` combination, which can cause the files a user was working on to move. Most other changes, such as changes to the account's default shell, simply don't take effect until the user has logged out and back in again.

If you change the account's UID, this action does *not* change the UIDs stored with a user's files. Because of this, the user may lose access to these files. You can manually update the UIDs on all files by using the `chown` command described earlier, in "Ownership Modification." Specifically, a command like the following, issued after changing the UID on the account `sally`, will restore proper ownership on the files in `sally`'s home directory:

```
# chown -R sally /home/sally
```

This action does *not* change the ownership of files that aren't in `sally`'s home directory. If you believe such files exist, you may need to track them down with the `find` command, as described shortly in "Deleting Accounts." Also, this command blindly changes ownership of *all* files in the `/home/sally` directory. This is probably desirable, but it's conceivable that some files in that directory *should* be owned by somebody else—say, because `sally` and another user are collaborating on a project.

When using the `-G` option to add a user to new groups, be aware that any groups *not* listed will be removed. The `gpasswd` command, described shortly, provides a way to add a user to one or more specific groups without affecting existing group memberships, and so it is generally preferable for this purpose.

Using *chage*

The chage command allows you to modify account settings relating to account expiration. It's possible to configure Linux accounts so that they automatically expire if either of two conditions is true:

- The password hasn't been changed in a specified period of time.
- The system date is past a predetermined time.

The first option is generally used to enforce password changes—say, to get users to change their passwords once a month. The second option is useful when an account should exist for a specific limited period of time, such as until the end of an academic semester or until a temporary employee leaves. These settings are controlled through the chage utility, which has the following syntax:

```
chage [-l] [-m mindays] [-M maxdays] [-d lastday]
↳[-I inactive] [-E expiredate] [-W warndays] username
```

Each parameter has a specific meaning:

-l This option causes chage to display account expiration and password aging information for a particular user.

-m *mindays* This is the minimum number of days between password changes. 0 indicates that a user can change a password multiple times in a day; 1 means that a user can change a password once a day; 2 means a user may change a password once every two days; and so on.

-M *maxdays* This is the maximum number of days that may pass between password changes. For instance, 30 would require a password change approximately once a month.

If the user changes a password before the deadline, the counter is reset from the password change date.

-d *lastday* This is the last day a password was changed. This value is normally maintained automatically by Linux, but you can use this parameter to artificially alter the password change count. For instance, you could use this to set the last changed date to force a password change in some period of time you determine. *lastday* is expressed in the format YYYY/MM/DD, or as the number of days since January 1, 1970.

-I *inactive* This is the number of days between password expiration and account disablement. An expired account may not be used or may force the user to change the password immediately upon logging in, depending upon the distribution. A disabled account is completely disabled, however.

-E *expiredate* You can set an absolute expiration date with this option. For instance, you might use **-E 2003/05/21** to have an account expire on May 21, 2003. The date may also be expressed as the number of days since January 1, 1970. A value of -1 represents no expiration date.

-W *warndays* This is the number of days before account expiration that the system will warn the user of the impending expiration. It's generally a good idea to use this feature to alert users of their situation, particularly if you make heavy use of password change expirations.

chage can normally only be used by root. The one exception to this rule is if the -l option is used; this feature allows ordinary users to check their account expiration information.

Directly Modifying Account Configuration Files

It's possible to directly modify user configuration files. /etc/passwd and /etc/shadow control most aspects of an account's basic features, but many files within a user's home directory control user-specific configuration; for instance, .bashrc can be used to set user-specific bash shell features. This latter class of configuration files is far too broad to cover here (Chapter 6, "Managing Files and Services," covers a few of them), but /etc/passwd and /etc/shadow are not. Both files consist of a set of lines, one line per account. Each line begins with a username and continues with a set of fields, delimited by colons (:). Many of these items may be modified with usermod or passwd. A typical /etc/passwd entry resembles the following:

```
sally:x:529:100:Sally Jones:/home/sally:/bin/bash
```

Each field has a specific meaning, as follows:

Username The first field in each /etc/passwd line is the username (sally in this example).

Password The second field has traditionally been reserved for the password. Most Linux systems, however, use a shadow password system in which the password is stored in /etc/shadow. The x in the example's password field is an indication that shadow passwords are in use. In a system that does not use shadow passwords, an encrypted password will appear here instead.

UID Following the password is the account's user ID (529 in this example).

Primary GID The default login group ID is next in the `/etc/passwd` line for an account. The example uses a primary GID of 100.

Comment The comment field may have different contents on different systems. In the preceding example, it's the user's full name. Some systems place additional information here, in a comma-separated list. Such information might include the user's telephone number, office number, title, and so on.

Home directory The user's home directory is next up in the list.

Default shell The default shell is the final item on each line in `/etc/passwd`. This is normally `/bin/bash`, `/bin/tcsh`, or some other common command shell. It's possible to use something unusual here, though. For instance, many systems include a shutdown account with `/bin/shutdown` as the shell. If you log into this account, the computer immediately shuts down. You can create user accounts with a shell of `/bin/false`, which prevents users from logging in as ordinary users but leaves other utilities intact. Users can still receive mail and retrieve it via a remote mail retrieval protocol like POP or IMAP, for instance. A variant on this scheme uses `/bin/passwd` so that users may change their passwords remotely but not actually log on using a command shell.

You can directly modify any of these fields, although in a shadow password system, you probably do *not* want to modify the password field; you should make password-related changes via `passwd` so that they can be properly encrypted and stored in `/etc/shadow`. As with changes initiated via `usermod`, it's best to change `/etc/passwd` directly only when the user in question isn't logged in, to prevent a change from disrupting an ongoing session.

Like `/etc/passwd`, `/etc/shadow` may be edited directly. An example `/etc/shadow` line resembles the following:

```
sally:E/moFkeT5UnTQ:11372:0:-1:7:-1:-1:
```

Most of these fields correspond to options set with the `chage` utility, although some are set with `passwd`, `useradd`, or `usermod`. The meaning of each colon-delimited field of this line is as follows:

Username Each line begins with the username. Note that the UID is *not* used in `/etc/shadow`; the username links entries in this file to those in `/etc/passwd`.

Password The password is stored in encrypted form, so it bears no obvious resemblance to the actual password.

If you've forgotten the root password for a system, you can boot with an emergency recovery system (as discussed in Chapter 9, "Troubleshooting") and copy the contents of a password field for an account whose password you do remember. You can then boot normally, log in as `root`, and change the password. In a real pinch, you can delete the contents of the password field, which results in a `root` account with *no* password. Be *sure* to *immediately* change the `root` password after rebooting if you do this, though!

Last password change The next field (`11372` in this example) is the date of the last password change. This date is stored as the number of days since January 1, 1970.

Days until change allowed The next field (`0` in this example) is the number of days before a password change is allowed.

Days before change required This field is the number of days after the last password change before another password change is required. Values of `-1` or `9999` typically indicate that this feature has been disabled.

Days warning before password expiration If your system is configured to expire passwords, you may set it to warn the user when an expiration date is approaching. A value of `7`, as in the preceding example, is typical.

Days between expiration and deactivation Linux allows for a gap between the expiration of an account and its complete deactivation. An expired account either cannot be used, or requires that the user change the password immediately after logging in. In either case, its password remains intact. A deactivated account's password is erased, and the account cannot be used until it's reactivated by the system administrator.

Expiration date This field shows the date on which the account will be expired. As with the last password change date, the date is expressed as the number of days since January 1, 1970.

Special Flag This field is reserved for future use and is normally not used or contains a meaningless value. This field is empty in the preceding example.

These values are generally best left to modification through the `usermod` or `chage` commands because they can be tricky to set manually—for

instance, it's easy to forget a leap year or the like when computing a date as the number of days since January 1, 1970. Similarly, because of its encrypted nature, the password field cannot be edited effectively except through `passwd` or similar utilities. (You could cut-and-paste a value from a compatible file, or use `crypt` yourself, but it's usually easier to use `passwd`. Copying encrypted passwords from other systems is also somewhat risky because it means that the users will have the same passwords on both systems, and this fact will be obvious to anybody who's acquired both encrypted password lists.)

`/etc/shadow` is normally stored with very restrictive permissions—`rw------`, with ownership by `root`. This fact is critical to the shadow password system's utility since it keeps non-`root` users from reading the file and obtaining the password list, even in an encrypted form. Therefore, if you manually modify `/etc/shadow`, be sure it has the correct permissions when you're done.

Deleting Accounts

On the face of it, deleting user accounts is easy. You may use the `userdel` command to do the job of removing a user's entries from `/etc/passwd` and, if the system uses shadow passwords, `/etc/shadow`. `userdel` takes just one parameter: `-r`. This parameter causes the system to remove all files from the user's home directory, as well as the home directory itself. Thus, removing a user account such as `sally` is easily accomplished with the following command:

```
# userdel -r sally
```

You may omit the `-r` parameter if you want to preserve `sally`'s files. There is one potential complication, however: Users may create files *outside* of their home directories. For instance, many programs use the `/tmp` directory as "scratch space," so user files often wind up there. These will be deleted automatically after a certain period, but you may have other directories in which users might store files. To locate all such files, you can use the `find` command with its `-uid` parameter. For instance, if `sally` had been UID 529, you might use the following command to locate all her files:

```
# find / -uid 529
```

The result will be a list of files owned by UID 529 (formerly `sally`). You can then go through this list and decide what to do with the files—change their ownership to somebody else, delete them, back them up to floppy, or what have you. It's wise to do *something* with these files, though, or else they may be assigned ownership to another user if `sally`'s UID is reused. This could become awkward if the files exceed the new user's disk quota or if they contain information that the new user should not have—such a person might mistakenly be accused of indiscretions or even crimes.

A few servers—most notably Samba—may keep their own list of users. If you run such a server, it's best to remove the user's listing from that server's user list when you remove the user's main account. In the case of Samba, this is normally done by manually editing the `smbpasswd` file (usually located in `/etc`, `/etc/samba`, or `/etc/samba.d`) and deleting the line corresponding to the user in question.

Configuring Groups

Linux provides group configuration tools that parallel those for user accounts in many ways. Groups are not accounts, however, so many features of these tools differ. Likewise, you can create or modify groups by directly editing the configuration files in question. Their layout is similar to that for account control files, but the details differ.

Adding Groups

Linux provides the `groupadd` command to add a new group. This utility is similar to `useradd`, but has fewer options. The `groupadd` syntax is shown here:

```
groupadd [-g GID [-o]] [-r] [-f] groupname
```

Each parameter to this command has a specific meaning:

`-g GID [-o]` You can provide a specific GID with the `-g` parameter. If you omit this parameter, `groupadd` uses the next available GID. Normally, the GID you specify must be unused by other groups, but the `-o` parameter overrides this behavior, allowing you to create multiple groups that share one GID.

-r This parameter instructs groupadd to create a group with a GID of less than 500. Not all distributions support this option; it was added by Red Hat and has been used on some related distributions. Red Hat uses GIDs of 500 and above for user private groups, as discussed shortly, in "The User Private Group," hence the -r parameter.

-f Normally, if you try to create a group that already exists, groupadd returns an error message. This parameter suppresses that error message. Not all versions of groupadd support this parameter.

In most cases, you'll create groups without specifying any parameters except for the group name itself, thus:

```
# groupadd project3
```

This command creates the project3 group, giving it whatever GID the system finds convenient—usually the highest existing GID plus 1. Once you've done this, you can add users to the group, as described shortly in "Modifying Group Information." When you add new users, you can add users directly to the new group with the -g and -G useradd parameters, described earlier.

Modifying Group Information

Group information, like user account information, may be modified either using utility programs or by directly editing the underlying configuration file, /etc/group. As with creation, there are fewer options for modifying groups than for modifying accounts, and the utilities and configuration files are similar. In fact, usermod is one of the tools that's used to modify groups.

Using *groupmod* and *usermod*

The groupmod command modifies an existing group's settings. Its syntax is shown here:

```
groupmod [-g GID [-o]] [-n newgroupname ] oldgroupname
```

The meanings of each of these options is as follows:

-g *GID* [-o] Specify the new group ID using the -g option. groupmod returns an error if you specify a new group ID that's already in use, unless you include the -o parameter, in which case you can create two groups that share a single GID.

-n *newgroupname* Specify a new group name with the -n option.

One of the most common group manipulations you'll perform, however, is not handled through `groupmod`; it's done with `usermod`. Specifically, `usermod` allows you to add a user to a group with its `-G` parameter. For instance, the following command sets `sally` to be a member of the `users`, `project1`, and `project4` groups, and it removes her from all other groups:

```
# usermod -G users,project1,project4 sally
```

Be sure to list all the user's current groups in addition to any groups to which you want to add the user. Omitting any of the user's current groups will remove the user from those groups. You can discover the groups to which a user currently belongs with the `groups` command, as in **`groups sally`**. To avoid accidentally omitting a group, many system administrators prefer to modify the `/etc/group` file in a text editor, or use `gpasswd`. Both options allow you to add users to groups without specifying a user's existing group memberships.

Using *gpasswd*

The `gpasswd` command is the group equivalent to `passwd`. `gpasswd` also allows you to modify other group features and to assign *group administrators*—users who may perform some group-related administrative functions for their groups. This is the basic syntax for this command:

```
gpasswd [-a user] [-d user] [-R] [-r] [-A user[,...]]
↳[-M user[,...]] group
```

The meanings of the options are as follows:

`-a user` This option adds the specified user to the specified group.

`-d user` This option deletes the specified user from the specified group.

`-R` This option configures the group to not allow anybody to become members through `newgrp`.

`-r` This option removes the password from a group.

`-A user[,...]` `root` may use this parameter to specify group administrators. Group administrators may add and remove members from a group and change the group password. Using this parameter completely overwrites the list of administrators, so if you want to add an administrator

to an existing set of group administrators, you must specify *all* their usernames.

`-M user[,...]` This option works like `-A`, but it also adds the specified user(s) to the list of group members.

If entered without any parameters except a group name, `gpasswd` changes the password for the group.

Group passwords allow you to control temporary membership in a group, as granted by `newgrp`. Ordinarily, members of a group may use `newgrp` to change their current group membership (affecting the group of files they create). If a password is set, even those who aren't members of a group may become temporary group members; `newgrp` prompts for a password that, if entered correctly, gives the user temporary group membership.

Unfortunately, some of these features are not implemented correctly in all distributions. In particular, password entry by non-group members sometimes does *not* give group membership—the system responds with an `access denied` error message. The `-R` option also sometimes doesn't work correctly—group members whose primary group membership is with another group may still use `newgrp` to set their primary group membership.

Directly Modifying Group Configuration Files

Group information is stored primarily in the `/etc/group` file. Like account configuration files, the `/etc/group` file is organized as a set of lines, one line per group. A typical line in this file resembles the following:

```
project1:x:501:sally,sam,ellen,george
```

Each field is separated from the others by a colon. The meanings of the four fields are as follows:

Group name The first field (`project1` in the preceding example) is the name of the group.

Password The second field (`x` in the preceding example) is the group password. Distributions that use shadow passwords typically place an `x` in this field; others place the encrypted password directly in this field.

GID The group ID number goes in this field.

User list The final field is a comma-separated list of group members.

Users may also be members of a group based on their own `/etc/passwd` file primary group specification. For instance, if `george` had `project1` listed as his primary group, he need not be listed in the `project1` line in `/etc/group`. If `george` uses `newgrp` to change to another group, though, he won't be able to change back to `project1` unless he's listed in the `project1` line in `/etc/group`.

Systems with shadow passwords also use another file, `/etc/gshadow`, to store shadow password information on groups. This file stores the shadow password and information on group administrators, as described earlier, in "Using `gpasswd`."

Deleting Groups

Deleting groups is done via the `groupdel` command, which takes a single parameter: a group name. For instance, **`groupdel project3`** removes the `project3` group. You can also delete a group by editing the `/etc/group` file (and `/etc/gshadow`, if present) and removing the relevant line for the group. It's generally better to use `groupdel`, though, because `groupdel` checks to see if the group is any user's primary group. If it is, `groupdel` refuses to remove the group; you must change the user's primary group or delete the user account first.

As with deleting users, deleting groups can leave "orphaned" files on the computer. You can locate them with the `find` command. For instance, if the deleted group had used a GID of 503, you can find all the files on the computer with that GID by using the following command:

```
# find / -gid 503
```

Once you've found any files with the deleted group's ownership, you must decide what to do with them. In some cases, leaving them alone won't cause any immediate problems, but if the GID is ever reused, it could lead to confusion and even security breaches. Therefore, it's usually best to delete the files or assign them other group ownership using the `chown` or `chgrp` commands.

Common User and Group Strategies

Linux's tools for handling users and groups can be quite powerful, but until you have some experience using them in a practical working environment, it's not always easy to see how best to use them. This is also one area

of system configuration that can't be preconfigured by distribution maintainers in a way that's very helpful. After all, user accounts and groups are necessarily local features—your system's users and groups will almost certainly be different from those of a system across town. Nonetheless, Linux distributions need to have some sort of default scheme for handling users and groups—what UIDs and GIDs to assign, and what groups to use for newly created users by default. Two such schemes are in common use, and each may be expanded in ways that may be useful to your system.

The User Private Group

The *user private group* scheme is used by Red Hat Linux and some of its derivative distributions, such as Mandrake. This scheme creates an initial one-to-one mapping of users and groups. In this system, whenever a user account is created, a matching group is created, usually of the same name. This matching group has one member—its corresponding user. For instance, when you create the account `sally`, a group called `sally` is also created. The account `sally`'s primary group is the group `sally`. When used without modification, the user private group strategy effectively eliminates groups as a factor in Linux security—because each group has just one member, group permissions on files become meaningless.

It's possible to modify group membership to control file access, however. For instance, if you want `george` to be able to read `sally`'s files, you can add `george` to the `sally` group and set `sally`'s umask to provide group read access to new files created by `sally`. Indeed, if you make all users group administrators of their own groups, users may control who has access to their own files by using `gpasswd` themselves. Overall, this approach provides considerable flexibility, particularly when users are sophisticated enough to handle `gpasswd`. Giving users such power may run counter to your system's security needs, though. Even when security policy dictates against making users group administrators, a user private group strategy can make sense if you need to fine-tune file access on a user-by-user basis. This approach can also provide asymmetrical file access. For instance, `george` may be able to read `sally`'s files (at least, those with appropriate group permissions), but `sally` might not have access to `george`'s files (unless `george` sets the world read bit on his files).

Project Groups

A second approach to group configuration is to create separate groups for specific purposes or projects. Therefore, I refer to this as the project group

approach. Consider an example of a company that's engaged in producing three different products. Most employees work on just one product, and for security reasons, you don't want users working on one product to have access to information relating to the other two products. In such an environment, a Linux system may be well served by having three main user groups, one for each product. Most users will be members of precisely one group. If you configure the system with a umask that denies world access, those who don't belong to a specific product's group won't be able to read files relating to that product. You can set read or read/write group permission to allow group members to easily exchange files. (Individual users may use `chmod` to customize permissions on particular files and directories, of course.) If a user needs access to files associated with multiple products, you can assign that user to as many groups as are needed to accommodate the need. For instance, a supervisor might have access to all three groups.

The project group approach tends to work well when a system's users can be easily broken down into discrete groups whose members must collaborate closely. It can also work well when users need not (and even should not) have ready access to each others' files, as with students taking the same class. In such a case, you would set the umask to block group access to users' files. The logical grouping of users can still be helpful to you as an administrator, however, because you can easily track users according to their group—you can easily find all files owned by users taking a particular class, for instance. (This tracking ability breaks down when users are members of multiple groups, though.)

Many Linux distributions default to using a type of project group approach. The default primary group for new users on such systems is typically called `users`. You can, and in most cases should, create additional groups to handle all your projects. You can leave the `users` group intact but not use it, rename it to the first project group name, or use `users` as an overarching group for when you want to give access to a file to most ordinary users, but perhaps not everyone (such as guest users on an FTP server).

Multiple Group Membership

On any system but a very simple one, it's likely that at least some users will be members of multiple groups. This means that these users will be able to do the following things:

- Read files belonging to any of the user's groups, provided that the file has group read permission.

- Write files belonging to any of the user's groups, provided that the file has group write permission.
- Run programs belonging to any of the user's groups, provided that the file has group execute permission.
- Change the group ownership of any of the user's own files to any of the groups to which the user belongs.
- Use `newgrp` to make any of the groups to which the user belongs the user's primary group. Files created thereafter will have the selected group as the group owner.

Multiple group membership is extremely important when using user private groups, as described above—without this, it's impossible to fine-tune access to users' files. Even in a project group configuration, though, multiple group membership is critical for users who need access to multiple groups' files.

You may find yourself creating a group membership scheme that's some combination of these two, or one that's unique unto itself. For instance, you might create multiple overlapping subgroups in order to fine-tune access control. It might be common in such a situation for users to belong to multiple groups. Part of the problem with such configurations is in teaching users to properly use the `newgrp` and `chgrp` commands, though—many less technically savvy users prefer to simply create files and not worry about such details.

Account Security

Creating, maintaining, and removing user accounts are obviously important activities on a Linux system. One particularly important account maintenance task (or set of tasks) is maintaining account security. Miscreants sometimes attack a system through vulnerable user accounts. Once access to a normal account is achieved, bugs or lax internal security can be exploited to allow the cracker to acquire `root` access, or the account can be used to attack other systems. Therefore, it's important that you attend to this matter, and periodically review your configuration to see that it remains secure.

The popular media uses the term *hacker* to refer to computer miscreants. This term has an older meaning, however—somebody who enjoys programming or doing other technically challenging work on computers but not in an illegal or destructive sense. Many Linux programmers consider themselves hackers in this positive sense. Thus, I use the term *cracker* to refer to those who break into computers.

Enforcing User Password Security

As a general rule, people tend to be lazy when it comes to security. In computer terms, this means that users tend to pick passwords that are easy to guess, and they change them infrequently. Both these conditions make a cracker's life easier, particularly if the cracker knows the victim. Fortunately, Linux includes tools to help make your users pick good passwords and change them regularly.

Common (and therefore poor) passwords include those based on the names of family members, friends, and pets; favorite books, movies, television shows, or the characters in any of these; telephone numbers, street addresses, or Social Security numbers; or other meaningful personal information. Any single word that's found in a dictionary (in *any* language) is a poor choice for a password. The best possible passwords are random collections of letters, digits, and punctuation. Unfortunately, such passwords are difficult to remember. A reasonable compromise is to build a password in two steps: First, choose a base that's easy to remember but difficult to guess. Second, modify that base in ways that increase the difficulty of guessing the password.

One approach to building a base is to use two *unrelated* words, such as "bun" and "pen." You can then merge these two words (`bunpen`). Another approach, and one that's arguably better than the first, is to use the first letters of a phrase that's meaningful to the user. For instance, the first letters of "yesterday I went to the dentist" become `yiwttd`. In both cases, the base should not be a word in any language. As a general rule, the longer the password the better. Older versions of Linux had password length limits of eight characters, but those limits have now been lifted. Many Linux systems require passwords to be at least four or five characters in length; the `passwd` utility won't accept anything shorter than that.

With the base in hand, it's time to modify it to create a password. Listed below are several possible modifications; the user should apply at least a couple of them:

Adding numbers or punctuation The single most important modification is to insert random numbers or punctuation in the base. This step might yield, for instance, `bu3npe&n` or `y#i9wttd`. As a general rule, add at least two symbols or numbers.

Mixing case Linux uses case-sensitive passwords, so jumbling the case of letters can improve security. Applying this rule might produce `Bu3nPE&n` and `y#i9WttD`, for instance.

Order reversal A change that's very weak by itself but that can add somewhat to security when used in conjunction with the others is to reverse the order of some or all letters. You might apply this to just one word of a two-word base. This could yield `nu3BPE&n` and `DttW9i#y`, for instance.

Your best tool for getting users to pick good passwords is to educate them. Tell them that passwords can be guessed by malicious individuals who know them, or even who target them and look up personal information in telephone books, on Web pages, and so on. Tell them that, although Linux encrypts its passwords internally, programs exist that feed entire dictionaries through Linux's password encryption algorithms for comparison to encrypted passwords. If a match is found, the cracker has found the password. Therefore, using a password that's not in a dictionary, and that isn't a simple variant of a dictionary word, improves security substantially. Tell your users that their accounts might be used as a first step towards compromising the entire computer, or as a launching point for attacks on other computers. Explain to your users that they should never reveal their passwords to others, even people claiming to be system administrators—this is a common scam, but real system administrators don't need users' passwords. You should also warn them not to use the same password on multiple systems because doing so quickly turns a compromised account on one system into a compromised account on all the systems. Telling your users these things will help them understand the reasons for your concern, and it is likely to help motivate at least some of them to pick good passwords.

If your users are unconcerned after being told this (and in any large installation, some will be), you'll have to rely on the checks possible in `passwd`. Most distributions' implementations of this utility require a minimum password length (typically four or five characters). They also usually check the

password against a dictionary, thus weeding out some of the absolute worst passwords. Some require that a password contain at least one or two digits or punctuation.

Password cracking programs, such as Crack (`http://www.users.dircon.co.uk/~crypto`), are easy to obtain. You might consider running such programs on your own encrypted password database to spot poor passwords, and in fact, this is a good policy in many cases. It's also grounds for dismissal in many organizations, and can even result in criminal charges being brought, at least if done without authorization. If you want to weed out bad passwords in this way, discuss the matter with your superiors and obtain permission before proceeding. Take extreme care with the files involved, too; it's probably best to crack the passwords on a computer with *no* network connections.

Another password security issue is password changes. Changing passwords frequently minimizes the window of opportunity for crackers to do damage; if a cracker obtains a password but it changes before the cracker can use it (or before the cracker can do further damage using the compromised account), the password change has averted disaster. As described earlier in this chapter, you can configure accounts to require periodic password changes. When so configured, an account will stop accepting logins after a period of time if the password isn't changed periodically. (You can configure the system to warn users when this time is approaching.) This is a very good option to enable on sensitive systems or those with many users. Don't set the expire time too low, though—if users have to change their passwords too frequently, they'll probably just switch between a couple of passwords, or pick poor ones. Precisely what "too low" a password change time is depends on the environment. For most systems, 1–4 months is probably a reasonable change time, but for some it might be longer or shorter.

Steps to Reduce the Risk of Compromised Passwords

Passwords can end up in crackers' hands in various ways, and you must take steps to minimize these risks. Steps you can take to improve your system's security include the following:

Change passwords frequently. As just mentioned, doing this can minimize the chance of damage due to a compromised password. Likewise, choosing a good password can minimize the risk that it will be discovered through any of several means described here.

Use shadow passwords. If a cracker who's broken into your system through an ordinary user account can read the password file, or if one of your regular users is a cracker who has access to the password file, that individual can run any of several password-cracking programs on the file. For this reason, you should use shadow passwords stored in `/etc/shadow` whenever possible. Most Linux distributions use shadow passwords by default.

Keep passwords secret. You should remind your users not to reveal their passwords to others. Such trust is sometimes misplaced, and sometimes even a well-intentioned password recipient might slip up and let the password fall into the wrong hands. This can happen by writing the password down, storing it in electronic form, or sending it by e-mail or other electronic means. Indeed, users shouldn't e-mail their own passwords even to themselves, because e-mail can be intercepted at various points.

Use secure remote login protocols. Certain remote login protocols are inherently insecure; all data traverse the network in an unencrypted form. Intervening computers can be configured to snatch passwords from such sessions. Because of this, it's best to disable Telnet, FTP, and other protocols that use cleartext passwords, in favor of protocols that encrypt passwords, such as Secure Shell (SSH). Chapter 5, "Networking," discusses these protocols in more detail.

Be alert to shoulder surfing. If your users log on using public terminals, as is common on college campuses, in Internet cafes, and the like, it's possible that others will be able to watch them type their passwords (a practice sometimes called "shoulder surfing"). Users should be alert to this possibility and minimize such logins if possible.

Some of these steps are things you can do, such as replacing insecure remote login protocols with encrypted ones. Others are things your users must do. Once again, this illustrates the importance of user education, particularly on systems with many users.

Disabling Unused Accounts

Linux computers sometimes accumulate unused accounts. This occurs when employees leave a company, when students graduate, and so on. You should be diligent about disabling such accounts because they can easily be abused, either by the individual who's left your organization or by others who discover the password through any of the means already discussed. As

discussed in detail earlier in this chapter you do this with the `userdel` command.

If the individual has had legitimate access to the `root` account, you must carefully consider how to proceed. If you have no reason to suspect the individual of wrongdoing, changing the `root` password and deleting the user's regular account are probably sufficient. If the individual might have sabotaged the computer, though, you'll have a hard time checking for every possible type of damage, particularly if the individual was a competent administrator. In such situations, you're better off backing up user data and reinstalling from scratch, just as you should if your system is compromised by an outsider.

Many Linux distributions create a number of specialized accounts that are normally not used for conventional user logins. These may include `daemon`, `lp`, `shutdown`, `mail`, `news`, `uucp`, `games`, `nobody`, and others. Some of these accounts have necessary functions. For instance, `daemon` is used by some servers, `lp` is associated with Linux's printing system, and `nobody` is used by various programs that don't need high-privilege access to the system. Other accounts are likely to be unused on many systems. For instance, `games` is used by some games, and so it isn't of much use on most servers or true productivity workstations. You may be able to delete some of these unused specialty accounts, but if you do so, you should definitely record details on the accounts' configurations so that you can restore them if you run into problems because the account wasn't quite as unused as it first seemed.

Filesystem Security

Security isn't just about protecting your system from unauthorized access by outsiders; it's also about preventing local users from damaging the system or each other's files, whether intentionally or not. A large Linux system can easily have one or two users who'll try doing malicious things and more who'll do dangerous things without knowing any better. There's also the possibility that a cracker could break into a local account, even on a small system with trusted users. For these reasons, Linux provides filesystem security in the form of permissions on files, as described earlier in this chapter.

Evaluating Your User File Permissions Scheme

As discussed earlier in this chapter, you can adjust the umask to set the default permissions set on new files created by users. The `umask` command that accomplishes this task may reside in system-wide startup scripts like `/etc/profile`, but individual users can override this setting, so you should keep in mind that you don't have absolute control over this setting.

An appropriate umask depends on the security needs of your system and interacts strongly with the group plan you use—user private groups, project groups, or some other strategy. Some possible configurations include the following:

High security If users should not have routine access to each other's files, you may want to use a umask of 077, which eliminates all permissions for everybody but the file's owner. If you use a project group strategy, users will be able to give other group members selective access to files by changing access rights on specific files. If you use a user private group strategy and give users group administrative privileges over their own groups, they'll be able to fine-tune access. In either case, users can grant world access to their files, as well.

Default user private groups If users need to routinely collaborate with arbitrary other users, you may want to use a user private group strategy and a umask of 027 or even 007. This will give members of the users' private groups read or read/write access to users' files by default. Users will have to run `chmod`, change their umask, or remove members from their private groups to restrict access.

Default project groups You can use a umask of 027 or 007 and a project group strategy to give members of a project group access to each others' files. Again, individuals may further restrict or widen permissions on individual files by using `chmod`.

Open system On a system in which users might legitimately need to access each others' files, a umask of 022 or even 002 is common. These give world read-only access to users' files. A umask of 002 also allows group members to write to these files. It's generally unwise to use a umask of 000, which gives full read/write access to the world; even on a system in which users generally trust each other, this sort of access can easily be abused or cause accidents.

In addition to evaluating default user permissions, you should examine permissions on common directories in the Linux filesystem. Most directories should be world-readable, but not world-writeable. In most cases, directories should be owned by `root` (with the `root` group), and the group permissions should reflect the world permissions. (With this ownership, the group permissions are actually seldom important, but they could be important if the `root` group were compromised.) Exceptions to these rules include the following:

`/tmp` and `/var/tmp` These directories need to be world-writeable because any user should be able to store temporary files here. They should also have their sticky bits set so that users cannot delete each others' files.

`/var/spool` directories As a general rule, `/var/spool` subdirectories are an odd lot. Some, such as `/var/spool/mail`, may have group ownership by special groups, with write access to those groups. Others, like `/var/spool/cron`, may not be readable by anybody but the owner. Some may be owned by `daemon`, `uucp`, or some other special user. The details sometimes differ from one distribution to another because of differing servers or security philosophies.

`/mnt/`*`mountpoint`* Mount point permissions are largely irrelevant since they can change when a partition or device is mounted.

`/home/`*`username`* Users' home directories need to be owned by the users in question, usually with group ownership set to the user's default group. Permissions should be set in accordance with your overall permissions policy—typically the same as directories created using the default umask.

`/root` The `root` user's home directory normally has 0700 permissions so that ordinary users can't see what's in that directory.

Unfortunately, it's impossible to give hard and fast rules about what directories should have particular permissions. If you're in doubt, try to find out what programs use a particular directory, and check their documentation. This can be a good way to learn about directories that are used by just one or two programs. Other times, the FHS (discussed in Chapter 7 and detailed at `http://www.pathname.com/fhs`) may provide clues for directories that are standardized.

Evaluating Permissions on Programs

Linux places most program files in binary directories, such as `/bin`, `/sbin`, and `/usr/X11R6/bin`. To be run, programs must have at least one execute

bit set. Most programs have all three execute bits set (owner, group, and world). This allows any user to run the program. You might, however, want to restrict access to certain programs to all but certain users. To do this, you should remove world execute permissions *and* world read permissions. (If users can read a file, they can copy it to their own directories, change permissions, and run the copy.) You can then create a group that contains the users who should be able to run the program, change the program's group ownership to that group, and be sure group execute permissions exist on the program file. If you want only the program's owner to be able to execute the program, give only the owner execute permission.

Removing execute permission from a file won't prevent a truly determined user from running most programs. Because source code to all standard Linux programs is readily available, users could download the source code and compile their own copies of utilities. This approach won't work for proprietary programs, though, and it might be ineffective if a program requires SUID access, as described shortly, in "Setting Process Permissions."

Removing Unnecessary Programs

You should take some time to get to know the programs that are installed on your computer. One excellent type of program to help you do this if your system is based on RPM or Debian packages is a GUI package installation utility. Red Hat and most of its derivative systems come with one called GNOME RPM (type **`gnorpm`** to launch it). SuSE's YaST (type **`yast`** or **`yast2`** to launch it) includes similar functionality. For Debian systems, the Storm Package Manager (type **`stormpkg`** to launch it) plays a similar role. Storm Package Manager comes with Storm Linux, but can be installed on other Debian-based systems. Chapter 3, "Software Management," discusses the use of such packages.

Removing packages you don't use saves disk space and can improve security by reducing the risk that a miscreant can abuse a bug in a program to do things you'd rather not be done. Crackers sometimes exploit buggy programs—particularly buggy SUID `root` programs, as discussed shortly—to do things the programs were never intended to do. On a system with many users, you may even want to remove many seemingly innocuous and common tools if they aren't being used. This may prevent miscreants from using your system as a launching pad for attacks on other computers. For instance, removing Linux's Telnet client and C compilers can go a long way toward

making a system an unappealing platform for attacks on others. Removing such tools is particularly important for systems that have a high profile, such as mail and Web servers, or for systems that serve as routers or firewalls. If your firewall lacks tools that an intruder might use to inflict further damage, you've made the network it protects safer in the event the firewall is compromised. Note the use of the relative term *safer*, though; a sufficiently resourceful cracker can certainly overcome a few missing utilities, particularly if just one opening is available. For instance, a cracker could use an FTP client to retrieve missing tools from another system.

You should be particularly diligent about removing unnecessary servers. If a local program, like `su` or `mount`, contains a bug that can be exploited, only local users or those who've already broken into the computer as local users can use the bug to wreak further havoc. If a server contains a bug, though, there's at least a chance that anybody on the Internet could abuse the bug and cause problems for you. This is true even if the server has no legitimate users.

You can find some unused servers by browsing through your installed packages as just described. There are some other places you should check, as well:

`/etc/inetd.conf` This file, described in Chapter 6, controls `inetd`, which starts many servers on some systems. As a general rule, if you don't understand a line in this file, you should comment it out by preceding it with a pound sign (#).

`/etc/xinetd.conf` or `/etc/xinetd.d` files `xinetd` is a replacement for `inetd` on some distributions. As with `inetd.conf`, if you see an entry you don't understand in `/etc/xinetd.conf` or a custom file in `/etc/xinetd.d`, you should disable that server. Do this by adding the line `disable = yes` to the server definition.

SysV startup scripts Check the SysV startup scripts in `/etc/rc.d/init.d` or `/etc/init.d`. Many of these are necessary for system operation, so you shouldn't be as quick to disable them as you are to disable `inetd` or `xinetd` services. You can find the package to which a startup file belongs by using your package management system, if your distribution uses one. Specifically, type something like **`rpm -qf /etc/rc.d/`*`scriptname`*** or **`dpkg -S /etc/rc.d/`*`scriptname`*** to locate the package that owns *`scriptname`*. You can then use **`rpm -qi`** ***`packagename`*** or **`dpkg -p`** ***`packagename`*** to find out about that package.

When removing servers, you should be sure you shut down the server either before or after changing the configuration. With servers that launch

through `inetd` or `xinetd`, you should restart the `inetd` or `xinetd` server itself *after* you remove the package or disable the server (by typing something like **`/etc/rc.d/inetd restart`**). For servers that start through SysV startup scripts, run the script with the `stop` parameter to stop the server before you remove the package.

Keeping Software Up-to-Date

Removing unnecessary programs can be important for various reasons, as just described. One of these reasons is that an unnecessary program may contain a bug that can be used to compromise your system. Unfortunately, this possibility exists even for *necessary* programs. Therefore, it's important that you keep your software up-to-date. This is particularly important for servers, but security flaws are found even in non-server programs.

The simplest way to keep a system current is to periodically check the distribution maintainer's Web or FTP site for updates. If an upgrade for a package you have installed is available, check its description to see if it fixes any security flaws. If it does (or even if it doesn't), you may want to install the update. Chapter 3 discusses such upgrades in more detail, including using common Linux package management tools to do it.

Detecting Intruders

If the worst should occur and a cracker breaks into your system, how can you detect the invasion? If the cracker is restricted to using a single account, such detection might be very difficult; you might not be able to tell the difference between normal users' activities and those of an unwelcome guest. You might, however, notice some unusual entries in your log files (as described shortly, in "Monitoring Log Files"), particularly if the intruder attempts to use utilities such as `su`, which produce log file entries. If your intruder is logging in from a distant computer, you might also notice the unusual login source, assuming your users don't normally do such things. Likewise, you might notice unusual temporal patterns in the account's login activities, such as back-to-back logins from different networks or logins when you know the account's owner is on vacation, camping without telecommunications equipment.

Attempts to break in are also likely to show up in log files—but as noted earlier, competent crackers frequently erase the evidence of such intrusions from log files, so you might not notice anything amiss from the log files alone. On the other hand, the vast majority of intruders aren't really competent crackers; they're *script kiddies*—so called because they break into

computers using intrusion scripts they did not devise. Such intruders frequently don't understand the subtleties of the systems they compromise, so they tend to leave traces in log files, and sometimes in the form of collateral damage—servers that mysteriously crash, features that cease working, and so on. Of course, you shouldn't jump to the conclusion that your system has been invaded because it's malfunctioning; many other factors can cause a computer to misbehave. It is one possibility, though.

Many crackers aren't content to merely break into a computer; they want to *do* something with that system. To that end, they modify the computer's files—they may create accounts for themselves, or give passwords to accounts that don't normally have them so they can log on more easily in the future. The intruders may install modified versions of critical system software. For instance, a modified `login` program might collect the passwords of other users, which sometimes allow entry to other systems. Some modified programs might be downright malicious or contain bugs that can cause serious damage to the computer. For these reasons, you should take the potential for damage to your system very seriously.

Perhaps the best way to detect system compromise is with an intrusion detection utility like Tripwire (`http://www.tripwire.org`). This utility records a set of information about all the important files on a computer, including various types of *checksums* and *hashes*—short digital "signatures" that allow you to quickly determine whether or not a file has been changed. (These can also be used in other ways; for instance, Linux uses hashes to store passwords.) With this database stored in a secure location, you can check your system periodically for alteration. If an intruder has modified any of your files, Tripwire will alert you to this fact. If you like, you can run a Tripwire verification on a regular basis—say, once a week in a cron job.

The RPM system also records checksums for packages. You can verify these with the `-V` parameter to `rpm`. Typing **`rpm -Va`** checks all files on the system. (This command also returns the names of missing files, files with altered date stamps, and so on.) RPM's checksums are easily overcome, though; a cracker need only install software via RPM to make the software appear legitimate to such a check.

Many distributions ship with Tripwire, but it may not be installed by default. The utility is controlled through two configuration files: `tw.cfg` and `tw.pol`, which often reside in `/etc/tripwire`. `tw.cfg` controls overall configuration options, such as where `tw.pol` resides, how Tripwire sends reports to the system administrator, and so on. `tw.pol` includes information on the files it's to include in its database, among other things. Both files are

binary files created from text-mode files called `twcfg.txt` and `twpol.txt`, respectively. You may need to edit `twpol.txt` to eliminate references to files you don't have on your system and to add information on files you do have but that the default file doesn't reference. Use the `twinstall.sh` program (which often resides in `/etc/tripwire`) to generate the binary configuration files and other critical database files. This utility will ask you to set a pair of pass phrases, which are like passwords but are typically longer, to control access to the Tripwire utilities. You'll then need to enter these pass phrases to have the utility do its encoding work.

Once you've generated the basic setup files, type **`tripwire --init`** to have it generate initial checksums and hashes on all the files it's configured to monitor. This process is likely to take a few minutes. Thereafter, typing **`tripwire --check`** will check the current state of the system against the database, and **`tripwire --update`** will update the database (say, in case you upgrade a package). The `--init` and `--update` operations require you to enter the pass phrase, but `--check` doesn't. Therefore, you can include an automated Tripwire check in a cron job.

Tripwire is best installed and initialized on a completely fresh installation, before connecting the computer to the Internet but after all programs have been configured. Although it's possible to install it on a system that's been up and running for some time, if that system has already been compromised without your knowledge, Tripwire won't detect that fact.

Setting Process Permissions

Most Linux programs run with the permissions of the user who executed them. For instance, if `jane` runs a program, that program can read precisely the same files that `jane` can read. A few programs, though, need additional privileges. For instance, `su`, which allows one user to take on another's identity, requires `root` privileges to do this identity switching. Such programs use the SUID bit, introduced earlier in this chapter, to have the program run with the privileges of the program file's owner. That is, the SUID bit alters the *effective user ID*. The SGID bit works in a similar manner, but it sets the group with which the process is associated. Although these

features are very useful and even occasionally necessary, they're also at least potential security risks, so you should be sure that as few programs use these features as possible.

The Risk of SUID and SGID Programs

There are two major potential risks with SUID and SGID programs:

- If the program allows users to do something potentially dangerous, ordinary users might abuse the program. For instance, Linux's `fdisk` program can modify a disk's partitions, potentially leading to a completely destroyed system if abused. Even comparatively innocuous programs like `cp` could be abused if set to be SUID `root`—if so configured, any user could copy any file on the computer, which is clearly undesirable in the case of sensitive files like `/etc/shadow`. For these reasons, neither `fdisk` nor `cp` is normally installed as an SUID program.
- Bugs in SUID and SGID programs can cause damage with greater than normal privileges. If some random program contains a bug that causes it to try to recursively remove all files on the computer, and if an ordinary user encounters this bug, Linux's filesystem security features will minimize the damage. If this program were SUID `root`, though, the entire system would be wiped out.

For these reasons, only programs that absolutely require SUID or SGID status should be so configured. Typically, these are programs that ordinary users might reasonably be expected to use and that require privileged access to the system. The programmers who work on such programs take great pains to ensure they're bug-free. As described earlier in this chapter, though, it's possible for `root` to set *any* program's SUID or SGID bit.

When Is SUID or SGID Necessary?

SUID and SGID are necessary when a program needs to perform privileged operations but may also legitimately be run by ordinary users. Some common programs that meet this description include `passwd`, `gpasswd`, `crontab`, `su`, `sudo`, `mount`, `umount`, and `ping`. This list is not complete, however.

You can remove the SUID bits on some of these programs, but that may mean that ordinary users won't be able to use them. Sometimes this may be acceptable—for instance, you might not want ordinary users to be able to mount and unmount filesystems. Other times, though, ordinary users really

do need access to these facilities. `su` is the best way for you to acquire `root` privileges in many cases, for instance; and ordinary users should be able to change their passwords with `passwd`.

Some programs have SUID or SGID bits set, but they aren't SUID or SGID `root`. These programs may need special privilege to access hardware device files or the like, but they don't need full `root` privilege to do so. For instance, SuSE and Debian both configure their xterm programs in this way. Such configurations are much less dangerous than are SUID `root` programs because these special users typically don't have unusual privileges except to a handful of device or configuration files.

Real World Scenario

Controlling Daemon Process Permissions

Servers are run in various ways, as described in Chapter 6. Some of these allow you to set the effective user IDs of the server processes. For instance, both `inetd` and `xinetd` allow you to specify the user under whose name the server runs. Sometimes a server needs to run with `root` permissions, but other times that's not necessary. You should consult a server's documentation to learn what its requirements are.

Some servers let you adjust their process ownership through configuration files. For instance, Apache lets you adjust the username used on most of its processes with the `User` option in its `httpd.conf` file. (In the case of Apache, one process still runs as `root`, but it spawns children that run with the ownership you specify.)

Finding SUID or SGID Programs

You can use the `find` command to locate files with their SUID or SGID bits set. Specifically, you need to use the `-perm` parameter to this command, and specify the `s` permission code in the user or group. For instance, the following command locates all SUID or SGID files on a computer:

```
# find / -perm +ug+s
```

You may want to run this command and study the results for your system. If you're uncertain about whether a command should have its SUID or SGID

bit set, check its man page and try to verify the integrity of its package using RPM, if your system uses RPM. For instance, type **`rpm -V packagename`**. This will turn up changes to the permissions of files in *packagename*, including changes to SUID or SGID bits. Of course, it's conceivable that a program might have had its SUID or SGID bit set inappropriately even in the original package file.

Monitoring Log Files

Keeping your filesystem in good order is important for security, but a Linux computer is a dynamic system. It's therefore important to monitor ongoing system activities. Many Linux programs, including most servers, login processes, and the kernel, record a summary of their activities in *log files*. As a result, these files contain a synopsis of important system activities. As such, they're useful security tools—if somebody attempts to do something unauthorized, there's a good chance that evidence of that activity will show up in one or more log files. Successful intruders often delete evidence of their activities from log files, though, so they aren't a 100 percent reliable means of detecting intruders. Nonetheless, monitoring your log files for evidence of intrusion attempts, or just plain malfunctioning software, is an important activity for a Linux system administrator.

Locating Important Log Files

The first trick to monitoring log files is knowing where they are. Most of these files are maintained by the system log daemon (`syslogd`) and the kernel log daemon (`klogd`). These utilities both rely upon the `/etc/syslog.conf` file for configuration. This file consists of a series of lines that define how to log certain types of messages. Lines that begin with pound signs (#) are comments and are ignored. A non-comment line begins with a definition of the type of log item (the "selector") and ends with an "action" for logging. For instance, consider the following entries:

```
mail.=debug;mail.=info;mail.=notice  /var/log/mail/info
mail.=warn                           /var/log/mail/warnings
mail.err                             @logger
```

The selectors are broken into two parts, separated by periods: a facility and a priority. The facility in each of these cases is `mail`, which defines messages related to e-mail delivery. The priorities are named codes, and in this example they're `debug`, `info`, `notice`, `warn`, and `err`. These correspond to increasing severity. Many servers let you adjust the verbosity of their logging at specific severity levels. Normally, any message of a specified priority or higher is logged using the action of the specified line; however, preceding the severity code with an equal sign (`=`) causes only messages of that severity to be logged. The preceding example uses this trick for most severities to keep log information isolated.

The action for most of these examples is to log the message to a file (`/var/log/mail/info` or `/var/log/mail/warnings`). The `err` priority, however, passes the log information to another computer—`logger`. Using one computer to hold logging information for others can be an important security precaution; if a cracker breaks into one system, the cracker can't modify logs without breaking into another computer.

Armed with this information, you can use `/etc/syslog.conf` to locate most of your important log files. These typically reside in `/var/log` and its subdirectories. On most distributions, `/var/log/messages`, `/var/log/secure`, and `/var/log/syslog` are particularly important. One or more of these files typically holds important general-purpose log information.

Some servers create their own log files. Typically, you configure the location of these log files in the server's own native configuration files. For instance, Samba uses the `log file` parameter in its `smb.conf` file to control the location of its logs. As with system logs, server-specific logs usually end up in `/var/log` or a subdirectory of it.

Most user programs don't log their activities. Shells, however, often keep simple records of their last few commands. Bash, for instance, uses the `.bash_history` file in the user's home directory for this purpose. Such logs are intended primarily to allow users to quickly repeat recent commands by pressing the up arrow key, but if you suspect a user of wrongdoing, you can check this file for evidence.

Information Recorded in Log Files

Servers and other processes record differing information depending upon their programming. The following is a list of information you might be particularly interested in monitoring:

- Logins by `root` and attempts to use `su` to acquire `root` privileges. Both successful and failed attempts are potentially important.

- Information logged by firewalls, which may include attempts to access closed ports. Although such accesses may be honest mistakes, they may also indicate malicious activity.
- Failed access attempts logged by TCP Wrappers, `xinetd`, or individual servers. As with firewall logs, these may indicate either honest mistakes or malicious activity.
- System shutdowns and startups. With Linux, these should be rare enough that any shutdown you didn't supervise is suspicious.
- Unscheduled server restarts, which, like system restarts, may indicate a successful intrusion.
- Bizarre server error messages. Sometimes attempts to break into a system using bugs in particular servers turn up as strange error messages, which may include strings of gibberish. Most often, if you see such an error, the attempt was not successful; but it's still good to know the attempt was made. Other errors may indicate server misconfiguration.
- Hardware error messages, such as *kernel oopses*. Messages that include the word `oops` indicate kernel bugs or failed hardware.

You may want to familiarize yourself with the contents of your system's log files. Although certain types of activity are common to many Linux systems, others are more system- or distribution-specific. For instance, you won't find sendmail log file entries on a stock Mandrake or Debian system since these distributions use other mail servers. Likewise, a system with lots of users will see many login attempts, some of which will be unsuccessful because users will forget or mistype their passwords; but a workstation with just one user may only see one or two logins a day.

Usual and Unusual Log File Activity

As an example, let's consider a few system log entries. These entries were produced on a system running Linux Mandrake 7.2; the log files and entry details may differ on other distributions. Let's look at a failed and successful user login, followed by a use of `su` to become `root`:

```
Apr 21 17:03:25 nessus PAM_unix[1302]: authentication
↳failure; LOGIN(uid=0) -> rodsmith for system-auth service
Apr 21 17:03:25 nessus login[1302]: FAILED LOGIN 1 FROM
↳(null) FOR rodsmith, Authentication failure
```

```
Apr 21 17:03:44 nessus PAM_unix[1302]: (system-auth)
↳session opened for user rodsmith by LOGIN(uid=0)
Apr 21 17:03:44 nessus  -- rodsmith[1302]: LOGIN ON tty2
↳BY rodsmith
Apr 21 17:04:54 nessus PAM_unix[3094]: (system-auth)
↳session opened for user root by rodsmith(uid=500)
```

The first two entries, with time stamps of 17:03:25, represent a failed login—the user entered the wrong password. The login process logged the time and username, among other information. The next two lines, with time stamps of 17:03:44, show a successful login. Because of the proximity in time of these actions, it's a fair bet that the user simply mistyped a password the first time—but you can't be sure of that. The final line, with a time stamp of 17:04:54, shows that the user `rodsmith` used `su` to acquire `root` privileges. Especially on sensitive systems, it's often wise to monitor such activities carefully.

System logs normally include a date and time, as well as the name of the computer logging the activity (`nessus` in this example). Depending upon the process making the entry, there may be information on the user or server making the entry, a process ID number (1302 and 3094 in the preceding examples), and more.

Tools to Aid in Log File Analysis

System log files can become quite large and unwieldy. Linux distributions include cron jobs (discussed in Chapter 7) that rotate log files. The logging process is temporarily shut down, existing log files are renamed, and the process is restarted. Such rotation might occur on a daily, weekly, or monthly basis, depending upon the amount of activity and the desire to keep log files available. Old log files are typically kept for a few rotations, then deleted.

Because daily and weekly cron jobs normally run late at night, Linux systems should be left powered up through the night, at least occasionally. This is standard practice for servers, but many people power down workstations every night. If you do this, Linux will never rotate its log files, so they may grow to a huge size, possibly even filling a partition.

Log rotation is helpful in keeping your system uncluttered, but other tools exist to help you analyze your log files. One of these is Logcheck (`http://www.psionic.com/abacus/logcheck`). This package comes with some distributions, such as Mandrake and Debian. Unfortunately, it requires a fair amount of customization for your own system, so it's most easily implemented if it comes with your distribution, preconfigured for its log file format. If you want to use it on another distribution, you must edit the `logcheck.sh` file that's at the heart of the package. This file calls the `logtail` utility that checks log file contents, so you must configure the script to check the log files you want monitored. You can also adjust features such as the user who's to receive violation reports and the locations of files that contain strings for which the utility should look in log files. Once it's configured, you call `logcheck.sh` in a cron job. Logcheck then e-mails a report concerning any suspicious system logs to the user defined in `logcheck.sh` (`root`, by default).

Physical Security

Although a lot of attention is focused upon network and other electronic forms of security, computer security really begins with *physical* security. If your computer isn't properly protected against physical tampering or theft, it becomes an easy target for abuse. There are several steps you can take to minimize the damage should an intruder gain physical access to your computer, so for any critical system, you should create and follow a plan to secure your computer starting with such mundane tools as locks on the door.

What an Intruder Can Do with Physical Access

Linux systems provide various software safeguards against abuse and unauthorized access, such as passwords, file permissions, and system logs. These mechanisms can be effective against remote attacks when used properly, but they're next to useless if an intruder can touch the computer hardware. Two obvious methods of attack, when given such access, are to steal the hard disk and to boot the system with the intruder's own boot medium.

If a thief takes your hard disk, that thief has access to all the data on the disk. Linux's password protection mechanisms are under the control of the OS, so all the burglar needs to do is install the disk in a system the burglar controls to gain access to your computer's files. Indeed, a spy could conceivably copy your hard disk's contents and you'd be none the wiser.

Even short of stealing a hard disk, if a computer can boot from a floppy disk, an intruder can gain access to your system. The miscreant need only bring a Linux emergency boot floppy and boot that. The end result is full access to your files. If the goal is destruction, the intruder need not even be versed in Linux—a DOS boot floppy with a few disk utilities can quite effectively wipe out your data.

Theft of the entire computer is also a possibility, of course. Such a theft might not even be motivated by a desire to steal your data or do you harm personally—the burglar might be after the hardware.

Steps to Mitigate Damage from Physical Attacks

You aren't completely powerless against the threat of physical attacks on your computer. The following are some of the steps you can take to protect yourself:

Remove removable media. If a computer has no floppy drive, no Zip drive, no CD-ROM drive, no tape backup drive, and so on, it will be difficult for an intruder to either boot the computer from anything other than its hard disk or walk out with data on a removable disk. Of course, an intruder could bring a hard disk for booting, but that would require opening the computer's case, thus slowing down the operation. Short of removing the drives, you can buy special locks that make them accessible only when the user has a key.

Restrict BIOS boot options. Most BIOSes include options to enable and disable particular boot media. If your computer must have removable media, you can set the BIOS to boot only from the hard disk. This will slow down an intruder, but these settings can be easily changed, so this measure has a noticeable security impact only if used in conjunction with the use of BIOS passwords (discussed next). This measure may still be worthwhile as a protection against viruses, however, some of which are transmitted on floppies. Although these viruses can't infect Linux, a few can damage LILO and render a system unbootable.

Use BIOS passwords. Most BIOSes have an option to set a password that must be entered before the system will boot or before BIOS settings can be changed. Setting this can go a long way toward preventing tampering, but it's not perfect. Motherboard BIOSes can be reset by modifying a jumper setting, so an intruder who can open the case can overcome this measure.

Use a LILO password. If a boot image includes the option `password = pass`, LILO will only boot that image if the user enters the password (`pass`). If the boot image also includes the `restricted` keyword, LILO only applies this password rule if the user tries to issue any boot parameters, such as `single`, which normally boots the system into a single-user mode.

Secure the computer. To prevent tampering with the insides of a computer, you can replace the normal screws used on most computer cases with screws that require special tools. Check with a locksmith or hardware store for such screws. You can also buy a hinge with a lock, if you need heavy-duty case security. Many computer shops sell kits that consist of chains and additional hardware to secure a computer to a desk or wall in order to deter outright theft of the entire computer.

Secure the room. Locks on the doors can go a long way towards keeping a computer secure. If an intruder can't touch the computer, the intruder can't do any of the other nasty things I've been describing. You may need to secure windows, as well—or better yet, place the computer in a room that doesn't *have* windows. Don't just install the locks, but be sure to *use* them, too.

Use data encryption. Assuming that an intruder can gain physical access to the computer, the best protection may not be a lock or a BIOS setting; it may be data encryption. Many applications provide some way to encrypt data. Some of these schemes are good, but some aren't. There are also separate programs that can encrypt any data file. In mid-2001, no standard Linux filesystem supports automatic data encryption, but this feature may arrive in the future. There's also a tool that lets you add automatic encryption to files through a loopback device. Check `http://www.linuxdoc.org/HOWTO/Loopback-Encrypted-Filesystem-HOWTO.html` for details.

The bottom line is that no security is perfect. You'll have to judge just *how much* security you need. In some environments, with some systems, you might be content to lock the door. In others, you may need to take extreme measures, up to and including routinely encrypting your data files.

Summary

Linux's accounts and its security model are inextricably intertwined. A single Linux system can support many users, who can be tied together in groups. Users can create files that they own, and that have permissions that define which other users may access the files and in what ways. To manage these users, you'll use a handful of commands, such as `useradd`, `groupadd`, `userdel`, `usermod`, and `passwd`. These commands allow you to manage your user accounts to suit your system's needs.

Once you've set up your user accounts, you'll need to engage in ongoing system security monitoring. This may include educating your users about the need for good passwords, enforcing password changes, and reviewing the permissions on common system files and directories. Particularly important are the SUID and SGID bits on programs, which cause the programs to be run with the authority of the user who owns the file, rather than the user who runs the program. If set incorrectly or if an SUID or SGID program has a bug, the result can be security breaches.

Another ongoing security task is monitoring log files. These files contain information on important system events, such as attempted logins and the major actions of servers. Crackers' actions sometimes manifest themselves in these files, so checking them can be important.

Last but not least is physical security. Somebody with physical access to your computer can, if given enough time, do just about anything to the system. Therefore, it's very important that unauthorized personnel not have physical access to a system.

Exam Essentials

Describe why accounts are important on a Linux system. Accounts allow several users to work on a computer with minimal risk that they'll damage or (if you so desire) read each others' files. Accounts also allow you to control normal users' access to critical system resources, limiting the risk of damage to the Linux installation as a whole.

Summarize the Linux ownership and permissions system. Files are owned by an individual account, and are also associated with one group. Permission bits allow the file's owner to control separately the read, write, and execute access for the file's owner, members of the file's group, and all other users.

Describe the purpose of groups in Linux. Groups allow system administrators and users to provide a level of access control between individual-user and world (all users) access. You can use groups to give an arbitrary set of users read or write access to a directory or to at least some of each others' files, for instance, while keeping other users out of those directories and files.

Locate important log files. Log files typically reside in `/var/log` or its subdirectories. Their exact locations are controlled through `/etc/syslog.conf` and the configuration files for specific servers.

Describe the characteristics of a good password. Good passwords resemble random strings of letters, numbers, and punctuation. To make them memorable to the account holder, they can be generated by starting from a base built on a personally-relevant acronym or a pair of unrelated words, then modified by adding letters and punctuation, mixing the case of letters, and reversing some sequences in the password.

Describe how to detect intruders on a computer. The RPM utility can spot files changed by careless intruders, but more sophisticated tools like Tripwire do so much more reliably. Sometimes unusual log file entries will tip you off to the activities of a cracker.

Evaluate the need for SUID or SGID programs. Some programs, such as `su` and `passwd`, must have enhanced privileges in order to operate. Most programs, though, do not need these privileges and so should not have their SUID or SGID bits set.

Summarize important physical security measures. Whenever possible, computers should be stored behind locked doors, or possibly chained in place. Depending upon your needs and environment, you may want to eliminate removable media, set the computer to boot only from the hard disk, lock the case shut, set a BIOS password, set a LILO password, or encrypt files on the hard disk.

Commands in This Chapter

Command	Description
`su`	Changes a user's login account. Often used to acquire superuser privileges after a normal user login.
`sudo`	Executes a single command with alternative permission. Often used to run administrative programs as `root`.
`newgrp`	Changes a user's login group.
`chown`	Changes a file's owner.
`chgrp`	Changes a file's group.
`chmod`	Changes a file's permissions.
`umask`	Changes the current umask; alters the permissions on files created by a process.
`useradd`	Creates a new user account.
`usermod`	Modifies settings for an existing user account.
`chage`	Changes account expiration (aging) information.
`userdel`	Deletes an existing user account.
`passwd`	Changes an account's password.
`groups`	Displays the groups to which a user belongs.
`groupadd`	Adds a new group.
`groupmod`	Modifies settings for an existing group.
`groupdel`	Deletes an existing group.

Command	Description
`gpasswd`	Changes a group password; adds and deletes users from a group.
`logcheck.sh`	Called from a cron job, this script checks your log files for suspicious or dangerous events and e-mails you a report.
`tripwire`	Records information on a system's programs in an encrypted database, updates that database, or verifies that the monitored files have not been altered.

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

account
checksum
cracker
effective user ID
file access permissions
file owner
file permissions
file type code
group administrator
group ID (GID)
group owner
group
hacker
hard link
hash
home directory
inode
kernel oops
log file
mode
permission bit
script kiddies
set group ID (SGID)
set user ID (SUID)

shadow password
soft link
sticky bit
superuser
symbolic link
user
user ID (UID)
user mask (umask)
user private group
username
virtual terminal (VT)

Review Questions

1. Which of the following are legal Linux usernames? (Choose all that apply.)

   A. `larrythemoose`

   B. `4sale`

   C. `PamJones`

   D. `Samuel_Bernard_Delaney_the_Fourth`

2. Why are groups important to the Linux user administration and security models?

   A. They can be used to provide a set of users with access to files, without giving *all* users access to the files.

   B. They allow you to set a single login password for all users within a defined group.

   C. Users may assign file ownership to a group, thereby hiding their own creation of the file.

   D. By deleting a group, you can quickly remove the accounts for all users in the group.

3. Which of the following actions allow one to perform administrative tasks? (Choose all that apply.)

   A. Logging in as an ordinary user and using the `chgrp` command to acquire superuser privileges

   B. Logging in at the console with the username `root`

   C. Logging in as an ordinary user and using the `su` command to acquire superuser privileges

   D. Logging in when nobody else is using the system, thus using it as a single-user computer

4. What command would you issue to change the ownership of `somefile.txt` from `ralph` to `tony`?

   A. **`chown ralph.tony somefile.txt`**

   B. **`chmod somefile.txt tony`**

   C. **`chown somefile.txt tony`**

   D. **`chown tony somefile.txt`**

5. Which of the following umask values will result in files with `rw-r-----` permissions?

   A. `640`

   B. `210`

   C. `022`

   D. `027`

6. Which of the following is true of Linux passwords?

   A. They are changed with the `password` utility.

   B. They may consist only of lowercase letters and numbers.

   C. They must be changed once a month.

   D. They may be changed by the user who owns an account or by `root`.

7. Which of the following commands configures the `laura` account to expire on January 1, 2003?

   A. **`chage -I 2003-01-01 laura`**

   B. **`usermod -e 2003-01-01 laura`**

   C. **`usermod -e 2003 laura`**

   D. **`chage -E 2003/01/01 laura`**

8. Which of the following does `groupadd` allow you to create?
   - **A.** One group at a time
   - **B.** An arbitrary number of groups with one call to the program
   - **C.** Only user private groups
   - **D.** Passwords for groups

9. Which of the following is true of the `groupdel` command? (Choose all that apply.)
   - **A.** It won't remove a group if that group is any user's default group.
   - **B.** It won't remove a group if the system contains any files belonging to that group.
   - **C.** It removes references to the named group in `/etc/group` and `/etc/gshadow`.
   - **D.** It won't remove a group if it contains any members.

10. Which of the following describes the user private group strategy?
    - **A.** It is a requirement of the Red Hat and Mandrake distributions.
    - **B.** It cannot be used with Debian GNU/Linux.
    - **C.** It lets users define groups independently of the system administrator.
    - **D.** It creates one group per user of the system.

11. Which of the following is true when a user belongs to the `project1` and `project2` groups?
    - **A.** The user must type **`newgrp project2`** to read files belonging to `project2` group members.
    - **B.** If group read permissions are granted, any file created by the user will automatically be readable to both `project1` and `project2` group members.
    - **C.** The user may use the `newgrp` command to change the default group associated with files the user subsequently creates.
    - **D.** The user's group association (`project1` or `project2`) just after login is assigned randomly.

12. Which of the following is an advantage of designating one well-protected computer to record log files for several other computers?

A. Logging information in this way minimizes network use.

B. The logging system can analyze the logs using Tripwire.

C. Logs stored on a separate computer are less likely to be compromised by a cracker.

D. You can log information to a separate computer that you can't log locally.

13. Why is a log file analysis tool like Logcheck useful?

A. Logcheck translates log file entries from cryptic comments into plain English.

B. Logcheck sifts through large log files and alerts you to the most suspicious entries.

C. Logcheck compares patterns of activity across several days or weeks and spots anomalies.

D. Logcheck uses information in log files to help identify a cracker.

14. A computer is chained firmly to the wall, all of its accounts are secured with good shadowed passwords, and it's configured to boot only from its hard disk, but the system has no BIOS or LILO password. No users are currently logged into this system. How might a malicious individual without an account on this system corrupt it if given a few minutes alone with it? (Choose all that apply.)

A. The intruder could reboot it, reconfigure it to boot from floppy, boot a DOS floppy, and use DOS's disk utilities to delete the Linux partitions and erase the hard disk.

B. The intruder could run a password-cracking program on the system's `/etc/passwd` file, thus obtaining all the user's passwords for use in further compromising the system at a later date.

C. The intruder could open the case, remove the hard disk and insert it in another computer, then modify the configuration files and return the hard disk to the original machine.

D. The intruder could utilize a bug in `su`, `passwd`, or some other SUID `root` program to acquire `root` privileges and then alter the system's configuration files.

15. Which of the following steps would *not* substantially improve security over the typical Linux installation to a computer with typical hardware and settings?

A. Set a BIOS password.

B. Set the computer to boot from a CD-ROM before a floppy.

C. Remove servers that were installed by default but that are not being used.

D. Lock the computer case shut.

16. How should you engage users in helping to secure your computer's passwords?

A. Educate them about the importance of security, the means of choosing good passwords, and the ways crackers can obtain passwords.

B. Give some of your users copies of the encrypted database file as backup in case a cracker breaks in and corrupts the original.

C. Enforce password change rules but don't tell users how crackers obtain passwords since you could be educating a future cracker.

D. Instruct your users to e-mail copies of their passwords to themselves on other systems so that they're readily available in case of an emergency.

17. Which of the following accounts is the most likely prospect for deletion on a mail server?

A. `daemon`

B. `games`

C. `mail`

D. `nobody`

18. Adjusting program files' execute permissions can limit who may run the programs. What is a limitation to such restrictions?

A. Users can compile their own copies of most programs.

B. If users know a program's location, they can run it by specifying a complete path.

C. Execute permission restrictions apply only to the bash shell.

D. Users can set the SUID bit to overcome restrictions.

19. At what point during system installation should you configure Tripwire?

A. Prior to installing major servers like Apache

B. After installing major servers but before configuring them

C. After installing and configuring major servers but before connecting the computer to the Internet

D. After connecting the computer to the Internet and running it for 1–4 weeks

20. Which of the following are risks of SUID and SGID programs? (Choose all that apply.)

A. The program files are large and so they may cause a disk to run out of space.

B. Bugs in the programs may cause more damage than they would in ordinary programs.

C. Users may be able to abuse a program's features, thus doing more damage than would otherwise be possible.

D. Because the programs require password entry, running them over an insecure network link runs the risk of password interception.

Answers to Review Questions

1. A, C. A Linux username must be less than 32 characters in length, may contain letters, numbers, and certain symbols, and must start with a letter. Options A and C both meet these criteria. (Option C uses mixed upper- and lowercase characters, which is legal but discouraged.) Option B begins with a number, which is invalid. Option D is longer than 32 characters.

2. A. Groups provide a good method of file-access control. Although they may have passwords, these are *not* account login passwords; those passwords are set on a per-account basis. Files do have associated groups, but these are *in addition* to individual file ownership, and so they cannot be used to mask the file's owner. Deleting a group *does not* delete all the accounts associated with the group.

3. B, C. Direct login as `root` and using `su` to acquire root privileges from an ordinary login both allow a user to administer a system. The `chgrp` command is used to change group ownership of a file, not to acquire administrative privileges. Although Linux does support a single-user emergency rescue mode, this mode isn't invoked simply by having only one user logged on.

4. D. **`chown ralph.tony somefile.txt`** sets the owner of the file to `ralph` and the group to `tony`. `chmod` is used to change file permissions, not ownership. Option C reverses the order of the filename and the owner.

5. D. `027` removes write permissions for the group and all world permissions. (Files normally don't have execute permissions set, but explicitly removing write permissions when removing read permissions ensures reasonable behavior for directories.) `640` is the octal equivalent of the desired `rw-r-----` permissions, but the umask sets the bits that are to be *removed* from permissions, not those that are to be set. `210` would remove write permission for the owner, but it would not remove write permission for the group, which is incorrect. This would also leave all world permissions open. `022` would not remove world read permission.

6. D. Both the superuser and the account owner may change an account's password. The utility for doing this is called `passwd`, not `password`. Although an individual user might use just lowercase letters and numbers for a password, Linux also supports uppercase letters and punctuation. The system administrator may enforce once-a-month password changes, but such changes aren't required by Linux per se.

7. D. Either **`chage -E`** or **`usermod -e`** may be used for this task, followed by a date expressed in `YYYY/MM/DD` format. Options A and B use dashes (`-`) instead of slashes (`/`) in the date format, and option A uses the wrong parameter (`-I`), as well. Option C is actually a legal command, but it specifies a date 2003 days after January 1, 1970—in other words, in mid-1975.

8. A. `groupadd` creates one group per call to the program. Such a group *may* be a user private group, but need not be. Group passwords are created with `gpasswd`, not `groupadd`.

9. A, C. `groupdel` modifies the group configuration files, but it checks the user configuration files to be sure that it doesn't "orphan" any users first. The group may contain members, though, so long as none list the group as their primary group. `groupdel` performs no search for files belonging to the group, but it's a good idea for you to do this manually after removing the group.

10. D. Although Red Hat and Mandrake use the user private group strategy by default, you can design and use another strategy yourself. Likewise, you *may* use the user private group strategy with any Linux distribution, even if it doesn't use this strategy by default. Ordinary users can't create groups by themselves, although if they're group administrators in a user private group system, they may add other users to their own existing groups.

11. C. The `newgrp` command changes the user's active group membership, which determines the group associated with any files the user creates. This command is *not* required to give the user access to files with other group associations, if the user is a member of the other group and the file has appropriate group access permissions. Files have exactly one group association, so a user who belongs to multiple groups must specify to which group any created files belong. This is handled at login by setting a default or primary group recorded with the user's other account defaults in `/etc/passwd`.

12. C. Crackers often try to doctor system logs to hide their presence. Placing logs on another computer makes it less likely that they'll be able to achieve this goal, so you're more likely to detect the intrusion. Logging to a separate computer actually *increases* network use. Tripwire doesn't do log analyses; that job is done by Logcheck, and Logcheck can run on any computer that stores logs. System loggers can record any information locally that can be logged remotely.

13. B. Logcheck uses pattern-matching rules to extract log file entries containing keywords associated with suspicious activity. Although the other options might be useful to have, Logcheck and other common log file analysis tools cannot perform these tasks.

14. A, C. BIOS options can easily be changed if the system has no BIOS password, and even a DOS floppy can be used to destroy a Linux installation. Linux's own filesystem security features are useless if the disk is moved to another computer. Password-cracking programs can only be run once the individual has access to the password file, which this malicious individual does not have. Furthermore, with shadow passwords enabled, the passwords are stored in `/etc/shadow`, not `/etc/passwd`, and good passwords are unlikely to yield to a password-cracking program. Without an account, the individual cannot exploit bugs in SUID `root` programs (and such bugs are extremely rare and are patched quickly once discovered, so the odds are they don't exist on this system).

15. B. Many computers boot from a floppy first, and this configuration is undesirable; but booting from a CD-ROM first is not significantly better, because bootable CD-ROMs are common and give a malicious individual as much control of the system as can a bootable floppy. The system should be set to boot from the hard disk first to improve security. The other options can all improve security.

16. A. Education allows users to understand the reasons to be concerned, which can motivate conformance with password procedures. Cracking procedures are common knowledge, so withholding general information won't keep that information out of the hands of those who want it. Copying password files and sending unencrypted passwords through e-mail are both invitations to disaster; encrypted files can be cracked, and e-mail can be intercepted.

17. B. One or both of `daemon` and `mail` might be required by the mail server or other system software, so these are poor prospects for removal. Likewise, `nobody` is used by a variety of processes that need only low-privilege access rights. `games` is most frequently used by games for high score files and the like, and so is most likely unused on a mail server.

18. A. Because source code is available for all standard Linux components, a user can compile a personal copy of any standard Linux program, thus bypassing any permissions restrictions you might place on the version installed with the system. Option B describes a way to run programs that aren't on the path; it won't have any effect if the file doesn't have appropriate execute permissions. Execute permission restrictions apply to all shells. Only `root` or the file's owner may set the SUID bit on a program.

19. C. Tripwire records checksums and hashes of major files, including server executables and configuration files. Thus, these files should be in place and properly configured before you configure Tripwire. Once the system has been running on the Internet, there's a chance that it's been compromised; you should install Tripwire prior to connecting the computer to the Internet in order to reduce the risk that its database reflects an already-compromised system.

20. B, C. SUID and SGID programs run with effective permissions other than those of the person who runs the program—frequently as `root`. Therefore, bugs or abuses perpetrated by the user may do more damage than could be done if the programs were not SUID or SGID. These programs don't consume more disk space than otherwise identical ordinary programs. Although some SUID and SGID programs ask for passwords (such as `passwd` and `su`), this isn't true of all such programs (such as `mount` and `ping`).

Networking

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **3.2 Configure the client's workstation for remote access (e.g., ppp, ISDN).**
- ✓ **3.4 Configure basic network services and settings (e.g., `netconfig`, `linuxconf`; settings for TCP/IP, DNS, DHCP).**
- ✓ **3.5 Configure basic server services (e.g., X, SMB, NIS, NFS).**
- ✓ **3.6 Configure basic Internet services (e.g., HTTP, POP, SMTP, SNMP, FTP).**
- ✓ **3.15 Configure access rights (e.g., `rlogin`, NIS, FTP, TFTP, SSH, Telnet).**
- ✓ **4.11 Use network commands to connect to and manage remote systems (e.g., `telnet`, `ftp`, `ssh`, `netstat`, transfer files, redirect Xwindow).**
- ✓ **5.4 Run and interpret `ifconfig`.**
- ✓ **7.8 Identify basic networking concepts, including how a network works.**

Networking is a complex topic that's touched upon in several chapters of this book. This chapter provides an introduction to basic Transmission Control Protocol/Internet Protocol (TCP/IP) network configuration and proceeds with an overview of many of the network client functions a Linux system can fulfill. (Although the CompTIA objectives 3.5 and 3.6 refer to "services," they mean clients.) This chapter also includes information on administering a Linux computer from a distance by using networking protocols. For more information on network clients and servers, you'll need to consult other books or documentation.

When considered broadly, networking is a way for computers to communicate with one another. Just as with human-to-human communication, though, computer communication can be used to accomplish many different goals. These goals are associated with one or more networking protocols. For instance, e-mail transfer uses certain protocols, which are different from the protocols used in file sharing. This chapter is devoted largely to these protocols and the basics of configuring them.

Understanding Networks

In the last two decades of the 20th century, networks grew dramatically. Both local networks and larger networks exploded in importance as increasingly sophisticated network applications were written. In order to understand these applications, it's useful to know something about network hardware and the most common network protocols. Both of these things influence what a network can do.

Basic Functions of Network Hardware

Network hardware is designed to allow two or more computers to communicate with one another. As described shortly, this hardware can take a variety of forms. Most network hardware comes as a card you plug into a computer, although some devices are external and interface through an ordinary port like a USB port or even an internal network card. Most networks rely upon wires or cables to transmit data between machines as electrical impulses, but some devices use radio waves or even light to do the job.

Sometimes the line between network hardware and peripheral interface ports can be blurry. For instance, a parallel port is normally not considered a network port; but when it is used with the Parallel Line Interface Protocol (PLIP; `http://www.linuxdoc.org/HOWTO/mini/PLIP.html`), the parallel port becomes a network device. More commonly, a USB or RS-232 serial port can become a network interface when used with the *Point-to-Point Protocol (PPP)*, as discussed later in this chapter.

At its core, network hardware is hardware that facilitates the transfer of data between computers. Hardware that's most often used for networking includes features that help this transfer in various ways. For instance, such hardware may include ways to address data intended for specific remote computers, as discussed later on in "Hardware Addresses." When basically non-network hardware is pressed into service as a network medium, the lack of such features may limit the utility of the hardware or require extra software to make up for the lack. If extra software is required, you're unlikely to notice the deficiencies as a user or system administrator because the protocol drivers handle the work.

Types of Network Hardware

Aside from traditionally non-network ports like USB, RS-232 serial, and parallel ports, common network hardware on Linux includes the following:

Ethernet Ethernet is the most common type of network hardware on local networks today. It comes in several varieties ranging from the old 10Base-2 and 10Base-5 (which use coaxial cabling similar to cable TV cable) to 10Base-T and 100Base-T (which use twisted-pair cabling that resembles telephone wire, but with broader connectors) to the cutting-edge 1000Base-T and 1000Base-SX (aka *gigabit Ethernet*, using twisted-pair or optical cables, respectively). In all these cases, the number preceding the "Base" indicates the technology's speed in megabits per second (Mbps). Plans are underway to develop another ten-fold speed increase.

Of the versions in use in 2001, 100Base-T is the most common for new installations, but gigabit Ethernet is likely to become more common as its price drops. Linux includes excellent Ethernet support, including drivers for almost every Ethernet card on the market.

Token Ring At one time an important competitor to Ethernet, IBM's *Token Ring* technology is rapidly falling behind. The fastest type of Token Ring clocks in at just 16Mbps. Just as important, it's costlier than Ethernet and has less in the way of hardware support. For instance, fewer printers support direct connection to Token Ring networks than to Ethernet networks. Linux includes support for several Token Ring cards, so if you need to connect Linux to an existing Token Ring network, you can do so.

FDDI *Fiber Distributed Data Interface (FDDI)* is a networking technology that's comparable to 100Base-T Ethernet in speed. FDDI uses fiber-optic cables, but a variant known as CDDI works over copper cables similar to those of 100Base-T. Both technologies are supported by the Linux FDDI drivers.

HIPPI *High-Performance Parallel Interface (HIPPI)* provides 800Mbps or 1600Mbps speeds. It's most commonly used to link computer clusters or supercomputers over dozens or hundreds of meters. Linux includes limited HIPPI support.

LocalTalk *LocalTalk* is a network hardware protocol developed by Apple for its Macintosh line. It's slow by today's standards (2Mbps), and Apple no longer includes LocalTalk connectors on modern Macintoshes. Nonetheless, there were a few *x*86 LocalTalk boards produced, and Linux supports some of these. Therefore, if you need to connect an *x*86 Linux system to a LocalTalk network, you can do so—if you can find a LocalTalk board on the used market. (Ironically, the PPC port of Linux doesn't support the LocalTalk hardware on older Macintoshes.)

Fibre Channel *Fibre Channel* supports both optical and copper media, with speeds of between 133Mbps and 1062Mbps. The potential reach of a Fibre Channel network is unusually broad—up to 10 kilometers. Linux support for Fibre Channel is relatively new and incomplete, but it does exist.

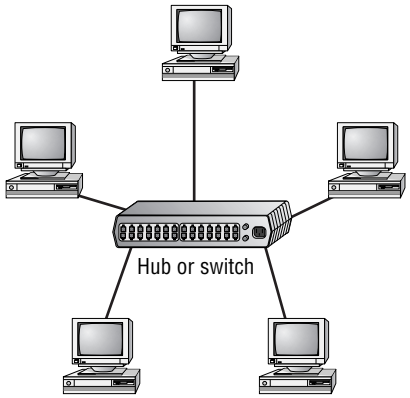
Wireless protocols Several wireless networking products are becoming popular, particularly in small offices and homes. These products vary in speed and range. Linux supports several of them, but because they're so new, you should make sure Linux supports a given device before you buy it.

If you're putting together a new network for a typical office, chances are that 100Base-T or gigabit Ethernet will be the best choice. If you need to connect to an existing network, you should find out what type of hardware it uses. If necessary, consult with your local network administrator to find out what type of network card you need.

Some computers ship with network hardware preinstalled. This is true of all modern Macintoshes and some *x*86 PCs, especially those sold as office workstations. This hardware is almost always Ethernet.

In addition to the network cards you place in your computers, you need network hardware outside of the computer. With the exception of wireless networks, you'll need some form of network cabling that's unique to your hardware type. (For 100Base-T Ethernet, get cabling that meets at least Category 5, or Cat-5, specifications.) Many network types, including twisted-pair Ethernet, require the use of a central device known as a *hub* or *switch*. You plug every computer on a local network into this central device, as shown in Figure 5.1. The hub or switch then passes data between the computers.

FIGURE 5.1 Many networks link computers together via a central device known as a hub or switch.



As a general rule, switches are superior to hubs. Hubs mirror all traffic to all computers, whereas switches are smart enough to send packets only to the intended destination. The result is that switches let two pairs of computers

engage in full-speed data transfers with each other; with a hub, these two transfers would interfere with each other. Switches also allow *full-duplex* transmission, in which both parties can send data at the same time (like two people talking on a telephone). Hubs permit only *half-duplex* transmission, in which the two computers must take turns (like two people using walkie-talkies).

A hub or switch is located centrally in a logical sense, but it doesn't have to be so located geographically. An approximately central location may help simplify wiring, but when you decide where to put the device, take into account the layout of your computers, your rooms, and available conduits between rooms.

Network Packets

Modern networks operate on discrete chunks of data known as *packets*. Suppose you want to send a 100KB file from one computer to another. Rather than send the file in one burst of data, you break it down into smaller chunks. You might send 100 packets of 1KB each, for instance. This way, if there's an error sending one packet, you can resend just that one packet, rather than the entire file. (Network protocols invariably include error-detection procedures.)

Typically, each packet includes an *envelope*, which includes the sender address, the recipient address, and other housekeeping information; and a *payload*, which is the data intended for transmission. When the recipient system receives packets, it must hold onto them and reassemble them in the correct order to re-create the complete data stream. (It's not uncommon for packets to be delayed or even lost in transmission, so these error-recovery procedures are critical. They're handled transparently by the computer's networking hardware.)

There are several different types of packets, and they can be stored within each other. For instance, Ethernet includes its own packet type (known as a *frame*), and the packets generated by networking protocols that run atop Ethernet, such as those discussed in the next section, "Network Protocol Stacks," are stored within Ethernet frames. All told, a data transfer can involve several layers of wrapping and unwrapping data. With each layer, packets from the layer above may be merged or split up.

Network Protocol Stacks

The packing and unpacking of network data is frequently described in terms of a *protocol stack*. Understanding how the pieces of such a stack fit together can help you understand networking as a whole, including the various network protocols used by Linux. Therefore, this section presents this information; it starts with a description of protocol stacks in general and moves on to the TCP/IP stack and alternatives to it.

What Is a Protocol Stack?

It's possible to think of network data at various levels of abstractness. For instance, at one level, a network carries data packets for a specific network type (such as Ethernet), which are addressed to specific computers on a local network. Such a description, while useful for understanding a local network, isn't very useful for understanding higher-level network protocols, such as those that handle e-mail transfers. These high-level protocols are typically described in terms of commands sent back and forth between computers, frequently without reference to packets. The addresses used at different levels also vary, as described in "Types of Network Addresses."

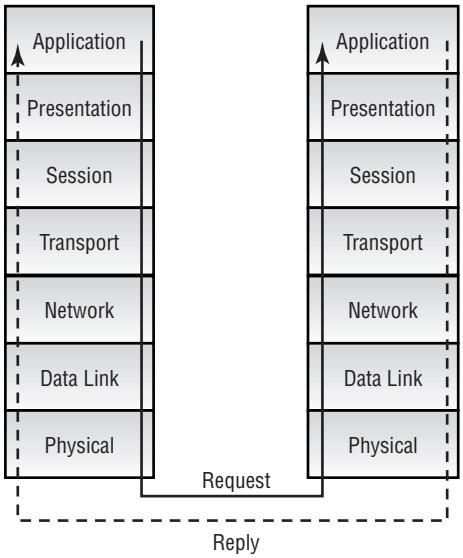
A protocol stack is a set of software that converts and encapsulates data between layers of abstraction. For instance, the stack can take the commands of e-mail transfer protocols, and the e-mail messages that are transferred, and package them into packets. Another layer of the stack can take these packets and repackage them into Ethernet frames. There are several layers to any protocol stack, and they interact in highly specified ways. It's often possible to swap out one component for another at any given layer. For instance, at the top of each stack is a program that uses the stack, such as an e-mail client. You can switch from one e-mail client to another without too much difficulty; both rest atop the same stack. Likewise, if you change a network card, you have to change the driver for that card, which constitutes a layer very low in the stack. Applications above that driver can remain the same.

Each computer in a transaction requires a compatible protocol stack. When they communicate, the computers pass data down their respective stacks, then send data to the partner system, which passes the data up its stack. Each layer on the receiving system sees the data as packaged by its counterpart on the sending computer.

The OSI Model

The interactions of a protocol stack should become clearer with an example. A common model used for describing protocol stacks generically is the *Open System Interconnection (OSI) model*, illustrated in Figure 5.2. This model breaks networking tasks down into seven layers, from the Application layer (in which users' clients and the servers to which they connect run) to the Physical layer (which consists of network hardware like Ethernet cards). Each layer in between these does some task related to the packaging of data for transport or its unpacking.

FIGURE 5.2 Information travels "down" and "up" protocol stacks, being checked and packed at each step of the way.



Each component layer of the sending system is equivalent to a layer on the receiving system, but these layers need not be absolutely identical. For instance, you can have different models of network card at the Physical layer, or you can even use entirely different network hardware types, such as Ethernet and Token Ring, if some intervening system translates between them. The computers may run different OSs entirely and hence use different—but logically equivalent—protocol stacks. What's important is that the stacks operate in compatible ways.

The TCP/IP Protocol Stack

The OSI model describes an idealized protocol stack; its features can be implemented in many different ways. One of the most common implementations is the *Transmission Control Protocol/Internet Protocol (TCP/IP)* stack. The TCP/IP stack is usually described in slightly different terms than is the OSI stack. Specifically, the TCP/IP stack is generally described using just four layers (Application, Transport, Internet, and Link), as opposed to OSI's seven (shown in Figure 5.2). The principles are the same for both models; the differences are just a matter of how the terms are applied and precisely how the stacks are implemented.

TCP/IP has several important features that make it a popular network protocol and the one upon which the Internet is based. These characteristics include the following:

Routable TCP/IP was designed so that computers configured in a particular manner could route packets between two networks. These computers (known as *gateways* or *routers*) make the Internet possible. A small network links to another one via a router, which links to another, and so on. Such a collection of networks is known as an *internet* (without capitalization). The *Internet* (capitalized) is a particularly large globe-spanning internet.

Flexible naming system TCP/IP supports two levels of names, one based on numbers and one based on text. The numeric system supports approximately four billion addresses, and the textual system supports multiple levels of names. Both features support large and complex network structures.

Multiple connection types TCP/IP supports several different types of connection, including the Transmission Control Protocol (TCP) after which the stack is named, the User Datagram Protocol (UDP), and the Internet Control Message Protocol (ICMP). These connection protocols support differing levels of complexity and error correction.

Standards-based The TCP/IP stack and many of the protocols that use it are described by documents maintained by the Internet Engineering Task Force (IETF; `http://www.ietf.org`), an international standards organization. IETF protocols are nonproprietary, so they may be implemented by anybody who cares to examine and put to use the IETF standards documents, which are known as *Requests for Comments (RFCs)*.

This combination has made TCP/IP a formidable protocol stack. It's been implemented in a huge array of OSs, ranging from DOS to Linux. A huge number of network tools are built atop TCP/IP, including everything related to the Internet—Web browsers, e-mail clients, and so on. A few networking programs, though, either don't use TCP/IP or use it only optionally. Other protocol stacks remain popular in certain environments, and you should be aware of these and how they interact and compete with TCP/IP.

Alternatives to TCP/IP

TCP/IP was initially developed using Unix, but today it is used on many other platforms. Some of these other OSs have their own protocol stacks. Most of these have also been implemented on other OSs, including Linux. These TCP/IP alternatives don't support as many networking applications, though, and they're generally limited to use on much smaller networks than TCP/IP supports. Nonetheless, you may encounter these protocol stacks in some environments. They include those listed here:

NetBEUI IBM and Microsoft have been the driving forces behind *NetBEUI*, which is a protocol stack that was developed for local networks of DOS and, later, OS/2 and Windows systems. NetBEUI is closely associated with *NetBIOS*, upon which Microsoft's file sharing protocols are built. For this reason, many Windows networks make extensive use of NetBEUI. It's also possible to use NetBIOS over TCP/IP, and this is the approach that Linux's Samba file server package uses to interconnect with Windows clients. Linux doesn't include a NetBEUI stack of its own, although Procom Technologies (`http://www.procom.com`) has developed one that is not part of the regular Linux kernel. Chances are you won't need to use this because Samba works well over TCP/IP, and Samba is the only Linux package that might use a NetBEUI stack.

IPX The *Internet Packet Exchange (IPX)* and *Sequenced Packet Exchange (SPX)* protocols constitute a protocol stack that's similar in broad strokes to TCP/IP or NetBEUI. IPX/SPX are the core of Novell's networking tools, which compete for file and printer sharing in DOS and Windows networks. IPX/SPX support is included in the Linux kernel, although it might not be compiled by default in all kernels. There are also file- and printer-sharing packages available that use the IPX/SPX stack.

AppleTalk Apple developed the *AppleTalk* stack for use with its LocalTalk network hardware. Its main use is with the AppleShare file-sharing protocols. Although initially tied to LocalTalk, AppleTalk can now be used over Ethernet (a combination that's sometimes called EtherTalk).

The Linux kernel includes support for AppleTalk, but this may not be compiled in all kernels. The Linux package that supports AppleTalk and AppleShare is Netatalk. Netatalk supports not just the old AppleTalk, but AppleShare IP, which uses TCP/IP as the protocol stack for file sharing. For the best functionality on a Macintosh network, you need both TCP/IP and AppleTalk support in Linux.

These alternatives to TCP/IP are all used on local networks, not on the Internet at large, which is a TCP/IP-based network. All of these alternatives are limited in ways that restrict their expansion. For instance, they lack the capacity to handle more than a couple of levels in their machine names. That is, as described shortly in "Hostnames," TCP/IP supports a hierarchical name structure that reduces the chance of conflicts in names, allowing every computer connected to the Internet to have a unique name. The naming schemes of these alternative stacks are much simpler, making it extremely impractical to maintain a worldwide naming system.

The Coming of IPv6

Another alternative protocol stack is actually an extension of TCP/IP. The current version of the IP portion of TCP/IP is 4. A major upgrade to this is in the works, however, and it goes by the name *IPv6*, for IP version 6. IPv6 adds several features and improvements to TCP/IP, including standard support for more secure connections and support for many more addresses. Check `http://playground.sun.com/pub/ipng/html/ipng-main.html` for detailed information on IPv6.

Although the four billion addresses allowed by TCP/IP sounds like a lot, there's been a lot of inefficiency in the allocation of those addresses. Therefore, as the Internet has expanded, the number of truly available addresses has been shrinking at a rapid rate. IPv6 raises the number of addresses to 2^{128}, or 3.4×10^{38}. This is enough to give every square millimeter of land surface on Earth 2.2×10^{18} addresses.

In 2001, IPv6 is still experimental, but some sites are beginning to use it. It's likely to be phased in over the next few years. The Linux kernel includes IPv6 support, so you can use it if you need to. Chances are that by the time the average office will need IPv6, it will be standard. Configuring a system

for IPv6 will be somewhat different than configuring it for IPv4, which is what this chapter describes.

Different protocol stacks are incompatible in the sense that they aren't completely interchangeable—for instance, you can't run an FTP client using AppleTalk. (A few protocols, like those used for Windows file sharing, can bind to multiple protocol stacks, though.) In another sense, these protocol stacks are *not* incompatible. Specifically, you can run multiple protocol stacks on one network or one computer. Many local networks today run two, three, or more protocol stacks. For instance, an office with both Macintoshes and Windows systems might run TCP/IP, NetBEUI, and AppleTalk.

Network Addressing

In order for one computer to communicate with another over a network, the computers need to have some way to refer to each other. The basic mechanism for doing this is provided by a network address, which can take several different forms, depending upon the type of network hardware, protocol stack, and so on. Large and routed networks pose additional challenges to network addressing, and TCP/IP provides answers to these challenges. Finally, to address a specific program on a remote computer, TCP/IP uses a *port number*, which identifies a specific running program, something like a telephone extension number identifies an individual in a large company. This section describes all these methods of addressing.

Types of Network Addresses

Consider an Ethernet network. When an Ethernet frame leaves one computer, it must be addressed to another Ethernet card. This addressing is done using low-level Ethernet features, independent of the protocol stack in question. Recall, however, that the Internet is composed of many different networks that use many different low-level hardware components. A user might have a dialup telephone connection (through a serial port) but connect to one server that uses Ethernet and another that uses Token Ring. Each of these devices uses a different type of low-level network address. TCP/IP requires something more to integrate across different types of network hardware. In total, there are three types of addresses that are important when you are trying to understand network addressing: network hardware addresses, numeric IP addresses, and text-based hostnames.

Hardware Addresses

At the lowest level of the OSI model is the Physical layer, which corresponds to network hardware. One of the characteristics of dedicated network hardware such as Ethernet or Token Ring cards is that they have unique *hardware addresses*, also known as *Media Access Control (MAC) addresses*, programmed into them. In the case of Ethernet, these addresses are six bytes in length, and they're generally expressed as hexadecimal (base 16) numbers separated by colons. You can discover the hardware address for an Ethernet card by using the `ifconfig` command. Type **`ifconfig eth`*`n`***, where *n* is the number of the interface (`0` for the first card, `1` for the second, and so on). You'll see several lines of output, including one like the following:

```
eth0      Link encap:Ethernet  HWaddr 00:A0:CC:24:BA:02
```

This line tells you that the device is an Ethernet card and that its hardware address is 00:A0:CC:24:BA:02. What use is this, though? Certain low-level network utilities and hardware use the hardware address. For instance, network switches use it to direct data packets. The switch learns that a particular address is connected to a particular wire, and so it sends data directed at that address *only* over the associated wire. The Dynamic Host Configuration Protocol (DHCP), which is discussed later in this chapter, is a means of automating the configuration of specific computers. It has an option that uses the hardware address to consistently assign the same IP address to a given computer. There are also advanced network diagnostic tools that let you examine packets that come from or are directed to specific hardware addresses.

For the most part, though, you don't need to be aware of a computer's hardware address. You don't enter it in most utilities or programs. It's important for what it does in general.

IP Addresses

Earlier, I said that TCP/IP allows for about 4 billion addresses. This figure is based on the size of the *IP address* used in TCP/IP: four bytes (32 bits). Specifically, 2^{32} = 4,294,967,296. Not all of these addresses are useable; some are overhead associated with network definitions, and some are reserved.

The 4-byte IP address and 6-byte Ethernet address are mathematically unrelated. Instead, the TCP/IP stack converts between the two using the *Address Resolution Protocol (ARP)*. This protocol allows a computer to

send a *broadcast* query—a message that goes out to all the computers on the local network. This query asks the computer with a given IP address to identify itself. When a reply comes in, it includes the hardware address, so the TCP/IP stack can direct traffic for a given IP address to the target computer's hardware address.

The procedure for computers that aren't on the local network is more complex. For such computers, a router must be involved, as described shortly, in "DNS and Routers: Linking It All Together."

IP addresses are usually expressed as four base-10 numbers (0–255) separated by periods, as in 192.168.29.39. If your Linux system's protocol stack is already up and running, you can discover its IP address by using `ifconfig`, as described earlier. The output includes a line like the following, which identifies the IP address (`inet addr`):

```
inet addr:192.168.29.39  Bcast:192.168.1.255
↳Mask:255.255.255.0
```

Although not obvious from the IP address alone, this address is broken down into two components: a network address and a computer address. The network address identifies a block of IP addresses that are used by one organization, and the computer address identifies one computer within that network. The reason for this breakdown is to make the job of routers easier—rather than record how to direct packets directed to each of the four billion IP addresses, routers can be programmed to direct traffic based on packets' network addresses, which is a much simpler job.

The *network mask* (aka the *subnet mask* or *netmask*) is a number that identifies the portion of the IP address that's a network address and the part that's a computer address. It's helpful to think of this in binary (base 2) because the netmask uses binary 1 values to represent the network portion of an address and binary 0 values to represent the computer address. The network portion always leads the computer portion. Expressed in base 10, these addresses usually consist of 255 or 0 values, 255 being a network byte and 0 being a computer byte. If a byte is part network and part computer address, it will have some other value. Another way of expressing a netmask is as a single number representing the number of network bits in the address. This number usually follows the IP address and a slash. For instance, 192.168.29.39/24 is equivalent to 192.168.29.39 with a netmask of

255.255.255.0—the last number shows the network portion to be three solid 8-bit bytes, hence 24 bits.

IP addresses and netmasks are extremely important for network configuration. If your network doesn't use DHCP or a similar protocol to assign IP addresses automatically, you must configure your system's IP address manually. A mistake in this configuration can cause a complete failure of networking or more subtle errors, such as an inability to communicate with just some computers.

Non-TCP/IP stacks have their own addressing methods. NetBEUI uses machine names; it has no separate numeric addressing method. AppleTalk uses two 16-bit numbers. These addressing schemes are independent from IP addresses.

Hostnames

Computers work with numbers, so it's not surprising that TCP/IP uses numbers as computer addresses. People, though, work better with names. For this reason, TCP/IP includes a way to link names for computers (known as *hostnames*) to IP addresses. In fact, there are *several* ways to do this, the most important of which is discussed in the next section, "DNS and Routers: Linking It All Together."

As with IP addresses, hostnames are composed of two parts: *machine names* and *domain names*. The former refers to a specific computer, and the latter to a collection of computers. Domain names are not equivalent to the network portion of an IP address, though; they're completely independent concepts. Domain names are registered for use by an individual or organization, which may assign machine names within the domain and link those machine names to any arbitrary IP address desired. Nonetheless, there is frequently some correspondence between domains and network addresses because an individual or organization that controls a domain is also likely to want a block of IP addresses for the computers in that domain.

Internet domains are structured hierarchically. At the top of the hierarchy are the top-level domains (TLDs), such as `.com`, `.edu`, and `.uk`. These TLD names appear at the *end* of an Internet address. Some correspond to nations (such as `.uk` and `.us`, for the United Kingdom and the United States, respectively), but others correspond to particular types of entities (such as `.com`

and `.edu`, which stand for commercial and educational organizations, respectively). Within each TLD are various domains that identify specific organizations, such as `sybex.com` for Sybex or `loc.gov` for the Library of Congress. These organizations may optionally break their domains into *subdomains*, such as `cis.upenn.edu` for the Computer and Information Science department at the University of Pennsylvania. Even subdomains may be further subdivided into their own subdomains; this structure can continue for many levels, but usually doesn't. Domains and subdomains include specific computers, such as `www.sybex.com`, Sybex's Web server.

When you configure your Linux computer, you may need to know its hostname. This will be assigned by your network administrator and will be a machine within your organization's domain. If your computer isn't part of an organizational network (say, if it's a system that doesn't connect to the Internet at all, or if it connects, it does so only via a dialup account), you'll have to make up a hostname. If your network uses DHCP, it may or may not assign your system a hostname automatically.

If you need to make up a hostname, choose an invalid TLD, such as `.invalid`. This will guarantee that you don't accidentally give your computer a name that legitimately belongs to somebody else. Such a name conflict could prevent you from contacting that system, and it could cause other problems as well, such as misdirected e-mail.

DNS and Routers: Linking It All Together

The Internet is big, and directing network traffic in a way that doesn't cause conflicts is a difficult task. There are two components that are particularly important in handling this task: the *Domain Name System (DNS)* and routers.

DNS is a distributed database of computers that convert between IP addresses and hostnames. Every domain must maintain at least two DNS servers that can either provide the names for every computer within the domain or that can redirect a DNS query to another DNS server that can better handle the request. Therefore, looking up a hostname involves querying a series of DNS servers, each of which redirects the search until the server that's responsible for the hostname is found. In practice, this process is hidden from you because most organizations maintain DNS servers that do all

the dirty work of chatting with other DNS servers. You need only point your computer to your organization's DNS servers. This detail may be handled through DHCP, or it may be information you need to configure manually, as described in "Basic Network Configuration."

As described earlier, routers pass traffic from one network to another. You configure your Linux system to directly contact systems on the local network. You also give the computer a router's address, which your system uses as a gateway to the Internet at large. Any traffic that's not destined for the local network is directed at this router, which passes it on to its destination. In practice, there are likely to be a dozen or more routers between you and most Internet sites. Each router has at least two network interfaces and keeps a table of rules concerning where to send data based on the destination IP address. Your own Linux computer has such a table, but it's likely to be very simple compared to those on major Internet routers. (Linux can function as a router, but such a configuration is beyond the scope of this book.)

Network Ports

Contacting a specific computer is important, but one additional type of addressing is still left: The sender must have an address for a specific program on the remote system. For instance, suppose you're using a Web browser. The Web server computer may be running more servers than just a Web server—it might also be running an e-mail server or an FTP server, to name just two of many possibilities. Another number beyond the IP address allows you to address traffic to a specific program. This number is a network port number, and every program that accesses a TCP/IP network does so through one or more ports.

When they start up, servers tie themselves to specific ports, which by convention are associated with specific server programs. For instance, port 25 is associated with e-mail servers, and port 80 is used by Web servers. Thus, a client can direct its request to a specific port and expect to contact an appropriate server. The client's own port number isn't fixed; it's assigned by the OS. Because the client initiates a transfer, it can include its own port number in the connection request, so clients don't need fixed port numbers. Assigning client port numbers dynamically also allows one computer to easily run several instances of a single client because they won't compete for access to a single port.

Fortunately, for basic functioning, you need to do nothing to configure ports on a Linux system. You may need to deal with this issue if you run unusual servers, though, because you may need to configure the system to link the servers to the correct ports.

Basic Network Configuration

Now that you know something about how networking functions, the question arises: How do you implement networking in Linux? Most Linux distributions provide you with the means to configure a network connection during system installation, as described in Chapter 2, " Installing Linux." Therefore, chances are good that networking already functions on your system. In case it doesn't, though, this section summarizes what you must do to get the job done. Chapter 9, "Troubleshooting," includes a few additional network troubleshooting tips, so if you have problems, you may want to consult Chapter 9.

Clients and Servers

Before proceeding further, one important distinction is the one between clients and servers. A *client* is a program that initiates a network connection to exchange data. A server listens for such connections and responds to them. For instance, a Web browser, such as Netscape or Opera, is a client program. You launch the program and direct it to a Web page, which means that the Web browser sends a request to the Web server at the specified address. The Web server sends back data in reply to the request. Clients can also send data, however, as when you enter information in a Web form and click a Submit or Send button.

The terms "client" and "server" can also be applied to entire computers that operate mostly in one or the other role. Thus, a phrase such as "Web server" is somewhat ambiguous—it can refer either to the Web server program or to the computer that runs that program. When this distinction is important and unclear from context, I clarify it (for instance, by referring to "the Web server program").

DHCP Configuration

One of the easiest ways to configure a computer to use a TCP/IP network is to use the *Dynamic Host Configuration Protocol (DHCP)*, which allows one computer on a network to manage the settings for many other computers. It works like this: When a computer running a DHCP client boots up, it sends a broadcast in search of a DHCP server. The server replies with the configuration information the client needs to allow it to communicate with other computers on the network—most importantly the client's IP address and netmask and the network's gateway and DNS server addresses. The DHCP server may also give the client a hostname. The client then configures itself with these parameters. From time to time, the client checks back with the DHCP server to "renew its lease" so that the DHCP server doesn't give the same IP address to another system.

There are three DHCP clients in common use on Linux: `pump`, `dhclient`, and `dhcpcd` (not to be confused with the DHCP server, `dhcpd`). Some Linux distributions ship with just one of these, but others ship with two or even all three. All distributions have a default DHCP client, though—the one that's installed when you tell the system you want to use DHCP at system installation time. Those that ship with multiple DHCP clients typically allow you to swap out one for another simply by removing the old package and installing the new one.

Ideally, the DHCP client runs at system bootup. This is usually handled either by a SysV startup file, as discussed in Chapter 6, "Managing Files and Services," or as part of the main network configuration startup file (typically a SysV startup file called `network` or `networking`). The system often uses a line in a configuration file to determine whether or not to run a DHCP client. For instance, Red Hat Linux sets this option in a file called `/etc/sysconfig/network-scripts/ifcfg-eth0` (this filename may differ if you use something other than a single Ethernet interface). The line in question looks like this:

```
BOOTPROTO="dhcp"
```

If the `BOOTPROTO` variable is set to something else, changing it as shown here will configure the system to use DHCP. It's usually easier to use a GUI configuration tool to set this option, however, as described shortly, in "Using GUI Configuration Tools."

Static IP Address Configuration

If a network lacks a DHCP server, you must provide basic network configuration options manually. There are several specific items that are required:

IP address You can set the IP address manually via the `ifconfig` command, but to set it automatically, you'll need to set it in a startup script such as `/etc/sysconfig/network-scripts/ifcfg-eth0` (as with DHCP configuration, the exact location of this information varies from one distribution to another). The `IPADDR` item contains the IP address.

Network mask The netmask can be set manually via the `ifconfig` command. To set it permanently, you must adjust a configuration file like `/etc/sysconfig/network-scripts/ifcfg-eth0`, in which it's set via the `NETMASK` item.

Gateway address You can manually set the gateway via the `route` command. To set it permanently, you need to adjust a configuration file, such as `/etc/sysconfig/network-scripts/ifcfg-eth0` (the `GATEWAY` item) or `/etc/sysconfig/network` (also the `GATEWAY` item). The gateway isn't necessary on a system that isn't connected to a wider network—that is, if the system works *only* on a local network that contains no routers.

DNS settings In order for Linux to use DNS to translate between IP addresses and hostnames, you need to specify at least one DNS server in the `/etc/resolv.conf` file. Precede the IP address of the DNS server by the keyword `nameserver`, as in `nameserver 192.168.29.1`. You can include up to three `nameserver` lines in this file. Adjusting this file is all you need to do to set the name server addresses; you don't need to do anything else to make the setting permanent.

If you're unsure of what to enter for these values, you should consult your network administrator. *Do not* enter random values or values you make up that are similar to those used by other systems on your network. Doing so is unlikely to work at all, and it could conceivably cause a great deal of trouble—say, if you mistakenly use an IP address that's reserved for another computer.

As just mentioned, the `ifconfig` program is critically important for setting both the IP address and netmask. This program can also display current settings. Basic use of `ifconfig` to bring up a network interface resembles the following:

```
ifconfig interface up addr netmask mask
```

For instance, the following command brings up `eth0` (the first Ethernet card) using the address 192.168.29.39 and the netmask 255.255.255.0:

```
# ifconfig eth0 up 192.168.29.39 netmask 255.255.255.0
```

This command links the specified IP address to the card so that the computer will respond to the address and claim to be that address when sending data. It doesn't, though, set up a route for traffic beyond your current network. For that, you need to use the `route` command, thus:

```
# route add default gw 192.168.29.1
```

Substitute your own gateway address for `192.168.29.1`. Both `ifconfig` and `route` can display information on the current network configuration. For `ifconfig`, omit `up` and everything that follows; for `route`, omit `add` and everything that follows. For instance, to view interface configuration, you might issue the following command:

```
# ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 00:A0:CC:24:BA:02
      inet addr:192.168.29.39  Bcast:192.168.29.255
↳Mask:255.255.255.0
      UP BROADCAST NOTRAILERS RUNNING  MTU:1500  Metric:1
      RX packets:16110258 errors:1 dropped:0 overruns:0
↳frame:1
      TX packets:14234890 errors:1 dropped:0 overruns:0
↳carrier:1
      collisions:0 txqueuelen:100
      Interrupt:10 Base address:0xc800
```

When configured properly, `ifconfig` should show a hardware address (`HWaddr`), an IP address (`inet addr`), and additional statistics. There should be few or no errors, dropped packets, or overruns for both received (`RX`) and transmitted (`TX`) packets. Ideally, there should be few collisions, but some are unavoidable if your network uses a hub rather than a switch. If collisions total more than a few percent of the total transmitted and received packets, you may want to consider replacing a hub with a switch. To use `route` for diagnostic purposes, you might use it as follows:

```
# route
Kernel IP routing table
```

```
Destination  Gateway      Genmask       Flags Metric Ref
↳Use Iface
192.168.29.0 *            255.255.255.0 U     0      0
↳0 eth0
127.0.0.0    *            255.0.0.0     U     0      0
↳0 lo
default      192.168.29.1 0.0.0.0       UG    0      0
↳0 eth0
```

This shows that data destined for 192.168.29.0 (that is, any computer with an IP address between 192.168.29.1 and 192.168.29.254) goes directly over `eth0`. The 127.0.0.0 network is a special interface that "loops back" to the originating computer. Linux uses this for some internal networking purposes. The last line shows the *default route*—everything that doesn't match any other entry in the routing table. This line specifies the default route's gateway system as 192.168.29.1. If it's missing or misconfigured, some or all traffic destined for external networks, such as the Internet, won't make it beyond your local network segment.

As with DHCP configuration, it's almost always easier to use a GUI configuration tool to set up static IP addresses, at least for new administrators. The exact locations of the configuration files differ from one distribution to another, so the examples listed earlier may not apply to your system.

Using GUI Configuration Tools

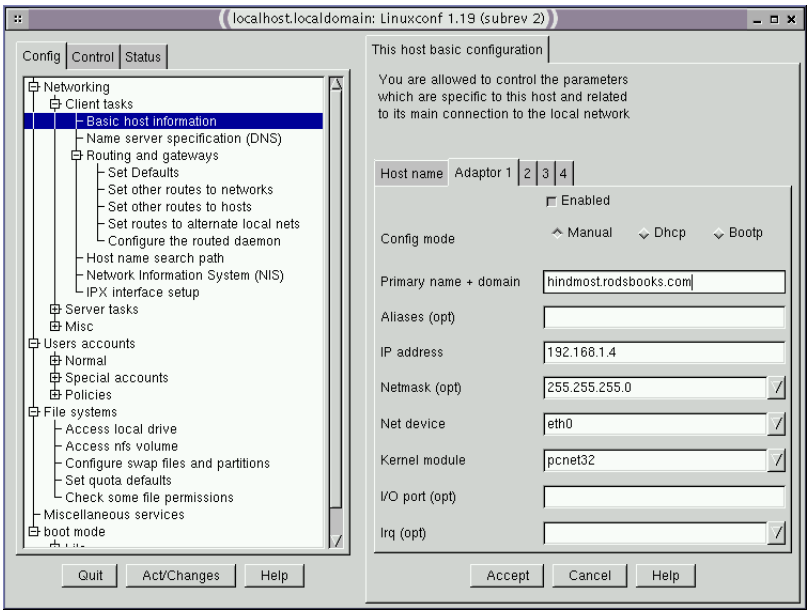
Most distributions include their own GUI configuration tools for network interfaces. For instance, Red Hat and many of its derivatives ship with `linuxconf` and `netconfig`; SuSE has YaST; and Caldera uses COAS. The details of operating these programs differ, but the GUI configuration tool provides a means to enter the information described earlier.

For instance, consider `linuxconf`. This program can be run from X, in text mode, or from a Web browser. The following procedure will configure a system using `linuxconf` and its X-based interface, but the other interfaces work in a similar way. (To use the Web interface, though, you must first configure `linuxconf` to operate in this way, which it doesn't do by default for security reasons.)

1. Start the utility by typing **`linuxconf`** in an `xterm` command prompt window or by selecting it from your desktop environment's menus. (You'll need to be `root` or an authorized administrative user to launch `linuxconf`.)

2. In the `linuxconf` window, select Networking ➢ Client Tasks ➢ Basic Host Information. A configuration panel entitled This Basic Host Configuration will appear to the right.
3. Click the Adaptor 1 tab. The display should now resemble that shown in Figure 5.3, although the entries will probably be different.

FIGURE 5.3 Network configuration using a GUI tool is a matter of entering information into fields that clearly identify the required information.

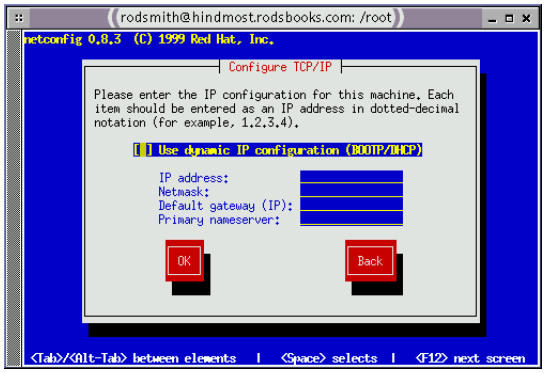


4. If your network uses DHCP, click the Dhcp item for the Config Mode and skip ahead to step 12.
5. Enter your computer's hostname in the Primary Name + Domain field.
6. Enter your IP address in the IP Address field.
7. After entering your IP address, a default netmask should appear in the Netmask (Opt) field. If yours differs, correct it.
8. Select Networking ➢ Client Tasks ➢ Name Server Specification (DNS) in the list to the left. You'll see a new configuration tab open to the right.

9. Enter your DNS servers in the fields provided on the right.
10. Select Networking ➢ Client Tasks ➢ Routing and Gateways ➢ Set Defaults in the list to the left. Again, a new configuration tab will appear on the right.
11. Enter your gateway address in the field provided for this purpose.
12. Click Accept in each of the open configuration tabs to the right, followed by Act/Changes below the list to the left. The result should be a small dialog box informing you that the system's status isn't synchronized with what you've configured.
13. Click Activate the Changes in the information dialog box to have it change your system's configuration and immediately enable the new network settings.
14. Click Quit in the remaining `linuxconf` window.

Not all network configuration tools are as elaborate as `linuxconf`. For instance, `netconfig` is a much simpler text-based tool. (Figure 5.4 shows it running in an `xterm` window.) You can enter the four required components in one place using `netconfig`, so it can be quicker than `linuxconf`—but it's also much less flexible.

FIGURE 5.4 `netconfig` is simpler than `linuxconf`, but it also does less.



Configuring Remote Client Access

Many Linux systems are connected directly to a network via a network card. Such systems are configured as just described—via DHCP or static IP address assignment. Other computers, though, use less direct connection methods. These computers are frequently located in homes or small businesses, and they need to use dial-up telephone connections, *Integrated Services Digital Network (ISDN)* links, or broadband connections. Such systems sometimes work much like direct connections, but they also sometimes require a different method of configuration.

Initiating a PPP Connection

A conventional telephone modem is the low end of network connectivity. This device turns the public telephone network into a computer networking medium, linking precisely two points together. In its simplest form, a modem can be used to initiate a text-mode connection using a *terminal program*—a program that allows for remote text-based logins but nothing else. Today, though, a modem is more often used in conjunction with PPP. PPP establishes a TCP/IP link between the two computers, so you can use any of the many TCP/IP-based tools, such as those discussed later in this chapter, in "Network Application Configuration." Most PPP accounts, though, are designed to be used for brief periods at a time, not continuously. Therefore, running servers on such systems is usually inadvisable because most servers require always-up Internet connections. Some ISPs do offer full-time PPP links, though.

To initiate a PPP connection, you must have PPP software installed on your Linux system. The most important PPP package is known as `pppd`, for "PPP daemon." This utility can both initiate PPP links and respond to attempts to initiate them. This section describes the former.

Using Text-Based PPP Utilities

In Linux, a PPP connection usually requires an entry in a file called `/etc/ppp/pap-secrets` or `/etc/ppp/chap-secrets`. Both files use the same format. They provide information that's passed between the PPP client and server for authentication, using the Password Authentication Protocol (PAP) or Challenge-Handshake Authentication Protocol (CHAP). Because PPP

was designed for use over public dial-up telephone lines, the caller must normally present a username and password to the other system; PAP and CHAP are merely protocols for doing this in a standard way. The format of lines in the secrets files are as follows:

```
username  server  password  IP_address
```

The *username* and *password* values are your username and password, respectively. Enter the values obtained from your Internet service provider (ISP). The *server* value is the name of the system to which you're connecting. Normally, it's an asterisk (*), signifying that `pppd` will connect to any computer. *IP_address* is the IP address that `pppd` expects to get. This will normally be blank, meaning that the system will accept any IP address.

Connecting from the command line requires modifying certain connection scripts. These are called `ppp-up`, `ppp-up-dialer`, and `ppp-down`. The first two start a connection, and the third breaks it. These scripts are often stored in a documentation directory, such as `/usr/share/doc/ppp-2.3.11/scripts`. Copy them to a convenient binary directory that's on your path, such as `/usr/local/bin`. You must then modify them with information relevant to your ISP:

- In `ppp-on`, locate the lines that begin `TELEPHONE=`, `ACCOUNT=`, and `PASSWORD=`, and modify them so that they're appropriate for your ISP and account. (The `ACCOUNT` and `PASSWORD` variables should contain dummy values if you use PAP or CHAP, as is almost always the case.)
- Check that the `DIALER_SCRIPT` variable in `ppp-on` points to the correct location of `ppp-on-dialer`. The default location is `/etc/ppp`.
- Check the call to `pppd` in the last lines of `ppp-on`. Most of the parameters to this call are quite cryptic, but you should at least be able to confirm that it's using the correct modem device filename and speed. Serial modems generally use `/dev/ttyS0` or `/dev/ttyS1` as the filename. `115200` is an appropriate speed in most cases, but the default is `38400`.
- Check the `ppp-on-dialer` script. This script includes a "chat" sequence—a series of strings the program expects to see from the modem or remote system in one column, and a series of responses in another column. You may need to log on using a terminal program like Seyon or `minicom` and then capture to disk the prompts your ISP uses to ask for your username and password; you'll then need to modify the last two lines of the script in order to make it work. Alternatively, you may need to comment out the last two lines by preceding

them with pound signs (#) and remove the backslash (\) from the CONNECT line if your ISP uses PAP or CHAP.

The chat program expects a single line; its input is only formatted in columns in ppp-on-dialer for the convenience of humans. The backslashes ending most lines signify line continuations so that chat interprets multiple input lines as a single line. Only the final line should lack a backslash.

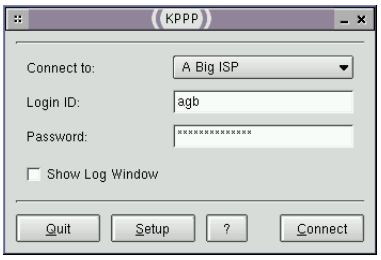
When you're done making these changes, type **ppp-on** (preceding it with a complete path, if necessary) as root to test the connection. If all goes well, your system should dial the modem, link up, and give you Internet access. If this fails to occur, check the last few lines of /var/log/messages with a command such as **tail -n 20 /var/log/messages**. There should be some sort of error messages, which may help you to diagnose the problem. To stop a connection, type **ppp-down**.

Using a GUI Dialer

Many people prefer to use GUI dialing utilities to control PPP connections. Many such programs are available. One that comes with most Linux systems is KPPP, which is part of the KDE system. You can use KPPP even from other environments, though, or you can use another GUI PPP dialer. Most PPP dialers offer similar features and functionality.

Figure 5.5 shows the main KPPP window. You can launch it by typing **kppp** in a terminal window or by selecting it from the main KDE menu (usually K ➢ Internet ➢ Kppp). Once it's configured, you need only select your ISP's name from the Connect To list, enter your username (in the Login ID field) and password, and click Connect to begin a connection. This button changes to allow you to disconnect once a connection is initiated.

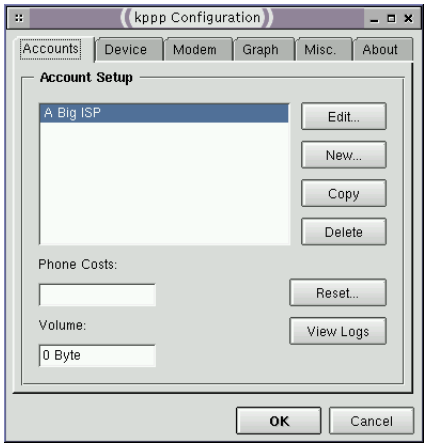
FIGURE 5.5 GUI dialers allow you to select from among several ISPs or dial-up numbers and connect by clicking a button.



To configure KPPP, follow these steps:

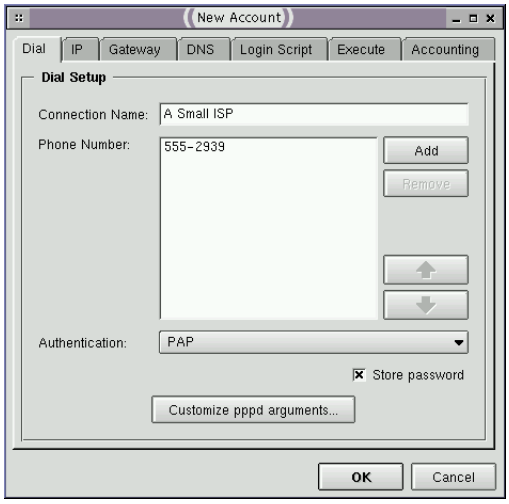
1. Click Setup. This produces the KPPP Configuration window shown in Figure 5.6. This window controls basic KPPP features and allows you to modify specific accounts.

FIGURE 5.6 The KPPP Configuration window controls accounts and overall KPPP settings.



2. Click New to create a new account. The system displays a dialog box asking if you want to use a wizard or set up using dialog boxes. The wizard doesn't support U.S. ISPs, so if you're in the U.S., you'll need to use the dialog box option. KPPP then displays the New Account dialog box (Figure 5.7).

FIGURE 5.7 The KPPP New Account dialog box lets you enter critical account-specific information.



3. Type an identifying name in the Connection Name field. This name exists so you can identify the configuration, so use whatever you like here.
4. Click Add, type in your ISP's phone number, and click OK. Repeat this step if your ISP provides multiple local access numbers.
5. Select the form of authentication used by your ISP (PAP is the most common, followed by CHAP and scripted logins).
6. Click OK in the New Account dialog box to close it.
7. In the KPPP Configuration window, check the settings on the Device and Modem tabs. You may need to adjust some of these, like the Modem Device and Connection Speed.

When you make a connection the first time, click Show Log Window in the main KPPP window (Figure 5.5). This produces a window that shows the interactions between your system and your ISP's, which can be helpful in case things don't go as you expect.

Using ISDN Services

ISDN is a digital alternative to conventional analog telephone lines. One advantage of ISDN is that it enables you to combine a pair of 64Kbps digital lines, for a net digital throughput of 128Kbps. This can be a substantial improvement over the 56Kbps maximum speed possible with analog telephone modems, but in today's world it's at best a stopgap measure; a better upgrade is to broadband, as described shortly, in "Using DSL or Cable Modem Services." Also, in the United States ISDN tends to be expensive, and availability is spotty. ISDN is much more popular in Europe, however.

ISDN service requires use of a device that's generally known as a terminal adapter. This fills a role that's similar to that of an analog telephone modem, and in fact, many of these adapters are external devices that interface through a serial port, just like normal telephone modems. When using such a terminal adapter, you can connect to your ISP just as you would when using an analog telephone modem—using PPP. Therefore, the preceding instructions on PPP connections apply equally well to such devices.

Some ISDN modems are internal cards, similar to internal analog telephone modems. These devices require special drivers in Linux. The Linux kernel includes drivers for several such cards, so there's a good chance yours is supported—but check on Linux support for any card before buying. The ISDN drivers allow the ISDN card to be accessed as if it were a modem, using commands similar to those used to dial ordinary modems. You must use the `/dev/ttyI0` (and subsequently numbered device filenames) rather than `/dev/ttyS0`, however.

Although ordinary PPP utilities can handle ISDN devices, many ISDN users prefer to use ISDN-specific utilities. These can help set up a system to dial up and transfer mail automatically, use the system as a firewall, and so on—tasks that are certainly possible with an analog telephone modem connection but that ISDN users are particularly likely to do. A set of utilities that can help with such configurations is available from `http://www.manna.nl/tipstekst_en.html`.

Using DSL or Cable Modem Services

Digital Subscriber Line (DSL) and cable modem services are the next step up in remote access connection methods. These methods are the most common form of *broadband* connectivity in 2001—high-speed remote access. (Broadband can also refer to media that allow the transmission of video and voice, as well as digital data.) Other forms of broadband include satellite, local radio, and fiber-optic connections, but these are still relatively uncommon. Like analog telephone and ISDN connections, broadband connections are typically used to link small numbers of computers in homes and businesses to the Internet via an ISP. Unlike typical PPP and ISDN connections, broadband connections can be operated 24 hours a day, so you may not need to run any commands to initiate a connection before using the service. (Two methods of connection, described shortly, are exceptions to this rule.)

DSL comes in several varieties, the most common of which are Asymmetric DSL (ADSL), Symmetric DSL (SDSL), and ISDN-based DSL (IDSL). Although the exact capabilities of these forms of DSL differ, as do the hardware requirements, configuration details are the same. Likewise, the basic principles of configuring a cable modem are the same as those of configuring DSL, although the hardware and underlying technologies differ.

Two factors affect broadband compatibility with Linux, and determine how you configure it:

Hardware compatibility All broadband connections require the use of a modem. A broadband modem isn't compatible with an ordinary analog telephone modem. Many external broadband modems interface to the host computer via an Ethernet port—the same type of interface that's used on many local networks. This type of interface is preferred in Linux. Other external modems use a USB interface. A few broadband modems come as internal cards. Both USB and internal broadband modems require special Linux drivers, which are rare.

IP address assignment method Broadband ISPs generally use one of four methods of IP address assignment: static IP addresses, DHCP, PPP over Ethernet (PPPoE), and PPP over ATM (PPPoA). The first two work exactly as they do in local networks, as described earlier, in "Basic Network Configuration." PPPoE and PPPoA are variants of ordinary PPP.

If your broadband provider gives you a Linux-incompatible modem, your only choices are to replace it with a compatible model (probably an external Ethernet-based device) or to write your own driver. The former is usually the easier course of action. In the IP address assignment arena, the main complication arises with ISPs that use PPPoE or PPPoA. PPPoE is the more common of these, and most Linux distributions now ship with at least one PPPoE package, usually Roaring Penguin (`http://www.roaringpenguin.com/pppoe`). Roaring Penguin comes with a text-based configuration script (`adsl-setup`) that asks you for critical information, such as your username and password. You can then initiate a connection by typing **`adsl-start`**, or you can stop it by typing **`adsl-stop`**.

Real World Scenario

Security and Broadband Connections

Many broadband users connect their computers directly to the broadband ISP's network, using only the broadband modem between the computer and the broadband network medium. This approach requires diligence in securing the connection, though; just like when you are connecting a system to a corporate network, the computer is readily accessible whenever it's powered on. Most corporate networks today include firewalls—devices that block unwanted access to internal computers. Few broadband ISPs provide firewalls, though, so a broadband-connected computer is unusually vulnerable. Furthermore, small business and home users frequently lack the knowledge to properly secure a computer, which makes for a dangerous combination.

One solution is to run a software firewall to protect the one connected computer (or that computer and any others connected indirectly through it). The "Controlling Access via a Firewall" section later in this chapter provides some pointers on firewall configuration in Linux.

Another option is to buy an external broadband router. These devices cost $50 to $500 (but most often $100–$150) and sit between your computer and an Ethernet broadband modem. They're programmed to function as firewalls, and they are usually very easy to configure. Most models also support network address translation (NAT), which allows you to share an Internet connection among several computers, even if your ISP provides just one IP address. Reviews of several models can be found at `http://www.practicallynetworked.com/reviews/index_router.htm`.

Network Application Configuration

Once you've got a network connection up and running, it's time to investigate what can be done with that connection. This section focuses primarily on popular network client programs—those programs that allow you as a user to initiate data transfers, such as retrieving e-mail or accessing filesystems on remote computers. The configuration of network servers (those

programs that respond to data transfer queries) is mostly beyond the scope of this book, although this section discusses a couple of them.

Using a Web Browser

The most popular Web browser in Linux is Netscape (`http://www.netscape.com`), but there are other choices. These include Mozilla (`http://www.mozilla.org`), a more fully open source offshoot of Netscape; Opera (`http://www.opera.com`), a commercial browser available on many platforms; Lynx (`http://lynx.browser.org`), a text-mode browser; and Konqueror (`http://www.kde.org`), a part of the K Desktop Environment (KDE).

Many desktop environments and window managers (discussed in "Basic GUI Use" in Chapter 6) include an easy way to launch a Web browser. In fact, Konqueror, KDE's Web browser, doubles as the environment's file manager and normally launches by default whenever you start KDE. If you prefer, you can launch a Web browser by typing its name in an xterm. (In the case of Lynx, you can launch it from a text-mode login, as well.)

Linux's GUI Web browsers work much as do their counterparts in other OSs. You can view pages linked to by the page you're viewing by clicking the link, which usually appears in another color. (Lynx uses arrow and Tab keys to highlight links; pressing the Enter key causes the program to load the linked-to page.)

Using an E-Mail Client

E-mail delivery on the Internet at large relies on a protocol known as the *Simple Mail Transfer Protocol (SMTP)*. This is known as a *push mail protocol* because the sender initiates a transfer. The mail server accepts the mail that's given to it and either stores it or forwards it to another system, depending upon the recipient address and the server's configuration. Linux computers almost invariably include SMTP servers and so can function as SMTP mail recipients.

Another type of mail protocol in common use today is known as a *pull mail protocol* because the receiver initiates the transfer. Typically, the pull mail client is a workstation, and the server handles mail for an entire organization. Two pull mail protocols are in common use: the *Post Office Protocol (POP)* and the *Internet Message Access Protocol (IMAP)*. Of the two,

IMAP is more sophisticated and allows the mail server to store and organize messages; POP is simpler and requires users to download and store mail locally in order to organize it into folders.

There are two main ways to handle e-mail in Linux, which mirror these two types of mail transfer protocols:

Read mail from the local mail queue. If you give correspondents your Linux system's name and your username on that system, you can let the Linux system function as an SMTP server and read mail directly on the Linux computer. For instance, if your system is `apollo.luna.edu` and your username is `hschmidt`, mail from other systems addressed to `hschmidt@apollo.luna.edu` will reach your system and be stored there for you to read.

Read mail from a remote system. If you don't want mail to be addressed directly to your own computer, you can use a pull mail protocol in conjunction with a separate mail server system. For instance, you might give your e-mail address as `hschmidt@mail.luna.edu`, then use your `apollo` workstation to retrieve mail from `mail.luna.edu`.

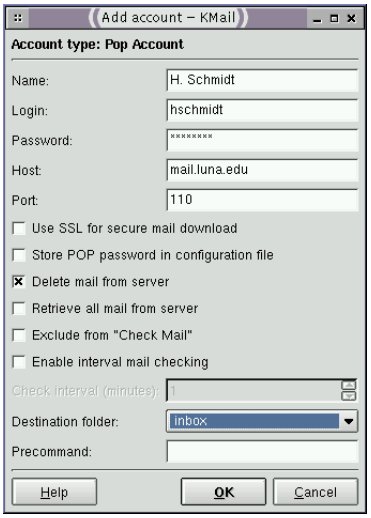
Each of these cases requires you to configure your mail reader appropriately, as described shortly. Using a local mail queue can make sense if the system has many users who don't have mail accounts on other systems. Reading mail from a separate mail server makes sense if your system's IP address changes frequently or if it's not online at all times, because SMTP mail delivery to your system will be unreliable in these cases.

SMTP servers can be misconfigured to function as *open mail relays*. These will forward mail from any address to any other address, and are beloved by those who send *spam*—unsolicited bulk e-mail. All Linux distributions released since 1999 or so are configured to *not* be open mail relays by default. If you're running an older distribution, or if you attempt to change your mail server's configuration, you should ensure that you aren't running an open mail relay. Consult `http://mail-abuse.org/tsi` for more information on this important topic.

Linux supports a wide variety of e-mail clients. These include Netscape Communicator (`http://www.netscape.com`), the mail portion of the Netscape Web browser; KMail (`http://www.kde.org`), KDE's mail client; Mutt (`http://www.mutt.org`), an advanced text-based mail reader; and

many others. As with Web browsers, you can launch mail clients by selecting them from desktop environment or window manager menus, or by typing the program's name at a shell prompt. You must usually configure the mail client to use either the local queue or a remote mail server. If the latter, you must enter the server's name, the protocols it uses, your username on that server (this may not be the same as your local username), and perhaps other information. Figure 5.8 shows the Add Account dialog box for KMail, in which most of this information is entered.

FIGURE 5.8 When using a pull mail protocol, your e-mail client uses your account on a mail server to retrieve your e-mail from that server.



You'll also have to choose how to send your mail. Because most Linux systems include a mail server (often sendmail, but there are other options), most mail programs give you the choice of using the local mail server to send outgoing mail or using an outside mail server. Chances are if you receive mail directly, you should send it using your local mail server; but if you receive mail through another mail server, you should send it in like manner.

Many organizations maintain separate incoming and outgoing mail servers. Therefore, you might not enter the same mail server's address as the outgoing mail server as you used when specifying the incoming mail server. Consult your ISP or network administrator for details.

The details of day-to-day mail client operation vary from one program to another, but as a general rule, these programs include functions to permit reading new mail, replying to such mail, sending new mail, deleting old messages, and organizing messages into mail folders. Many also let you save messages to files, spell-check your outgoing messages, and so on. When you use a remote mail server, you must either explicitly check for new mail (by clicking a button or selecting a menu option), or configure the program to do this automatically every once in a while.

Using X Programs Remotely

As noted in Chapter 2, Linux's GUI environment, the X Window System (or X for short), is unusual in that it's fully network-enabled. Using nothing but the normal X software and Linux network configuration, it's possible to run an X program on one computer while sitting at another computer, using the second computer's monitor, keyboard, and mouse. In fact, it's possible for one of these systems to be running a Unix OS that's not Linux. It's even possible to run an X server on a Windows, OS/2, or other completely non-Unix system, or on a system with a different class of CPU than the Linux system.

Although most people think of clients as running on the computers at which they sit and servers as running on remote systems, this isn't true of X. In X, the server runs on the system local to the user. To make sense of this, think of it from the program's point of view. To a word processor, the display and keyboard are services to be used, much like a network-accessible printer.

Suppose that your local network contains two machines. The computer called `zeus` is a powerful machine that hosts important programs, like a word processor and data analysis utilities. The computer called `apollo` is a much less powerful system, but it's got an adequate monitor and keyboard. Therefore, you want to sit at `apollo` and run programs that are located on `zeus`. Both systems run Linux. To accomplish this task, follow these steps:

1. Log into `apollo` and, if it's not already running X, start it.
2. Open a terminal (such as an xterm) on `apollo`.
3. Type **`xhost +zeus`** in `apollo`'s terminal. This command tells `apollo` to accept for display in its X server data that originates on `zeus`.

4. Log into `zeus` from `apollo`. You might use Telnet or Secure Shell (SSH), for instance. (See the upcoming sections, "Setting Up a Remote Access Server" and "Remote System Administration.") The result should be the ability to type commands in a shell on `zeus`.
5. On `zeus`, type **`export DISPLAY=apollo:0.0`**. (This assumes you're using bash; if you're using `tcsh`, the command would be **`setenv DISPLAY apollo:0.0`**.) This command tells `zeus` to use `apollo` for the display of X programs.
6. Type whatever you need to type to run programs at the `zeus` command prompt. For instance, you could type **`soffice`** to launch Star Office. You should see the programs open on `apollo`'s display, but they're running on `zeus`—their computations use `zeus`'s CPU, they can read files accessible on `zeus`, and so on.
7. After you're done, close the programs you've launched, log off of `zeus`, and type **`xhost -zeus`** on `apollo`. This will tighten security so that a miscreant on `zeus` won't be able to modify your display on `apollo`.

Sometimes, you can skip some of these steps. For instance, depending upon how it's configured, SSH can forward X connections, meaning that SSH intercepts attempts to display X information and passes those requests on to the system that initiated the connection. When this happens, you can skip steps 3 and 5, as well as the `xhost` command in step 7.

Another option for running X programs remotely is to use the Virtual Network Computing (VNC) system (`http://www.uk.research.att.com/vnc`). VNC runs a special X server on the computer that's to be used from a distance, and a special VNC client runs on the computer at which you sit. You use the client to directly contact the server. This reversal of client and server roles over the normal state of affairs with conventional X remote access is beneficial in some situations, such as when you are trying to access a distant system from behind certain types of firewall. VNC is also a cross-platform protocol; it's possible to control a Windows or MacOS system from Linux using VNC, but this is not possible with X. (X servers for Windows and MacOS are available, allowing you to control a Linux system from these non-Linux OSs, though.)

Using an FTP Client

The *File Transfer Protocol (FTP)* is one of the older Internet protocols for transferring files. Despite its age and deficiencies, FTP continues to be used because it's easy to configure and because FTP client programs are extremely common. On Linux, these programs include the original text-mode `ftp`, the updated text-mode NcFTP (`http://www.ncftpd.com/ncftp`), and GUI programs like gFTP (`http://gftp.seul.org`). In addition, Web pages may include links to files on FTP sites, so Web browsers can function as at least one-way FTP clients.

The purpose of FTP is, as the protocol's name implies, to transfer files between computers. When you run an FTP client, you can select files on the server or on your own system, then you click a button or issue a command to transfer the file. Text-mode FTP clients accept many commands, the most important of which are listed here:

`put` This command sends a single file from your system to the FTP server.

`get` You retrieve a single file from the server using the `get` command.

`mput` This command is a multifile variant of `put`; you can specify several files to send them all to the server.

`mget` This command is a multifile variant of `get`; you specify several files to retrieve them all.

`cd` Like the Linux shell command, `cd` in an FTP client changes to a new directory, but on the server.

`lcd` This command is similar to `cd`, but it changes your *local* directory—that is, the one on your own computer, not on the server.

`ls` or `dir` These two commands both produce directory listings on the server. Some servers respond differently to these two commands, typically producing more information (such as file sizes and dates) to `dir` than to `ls`.

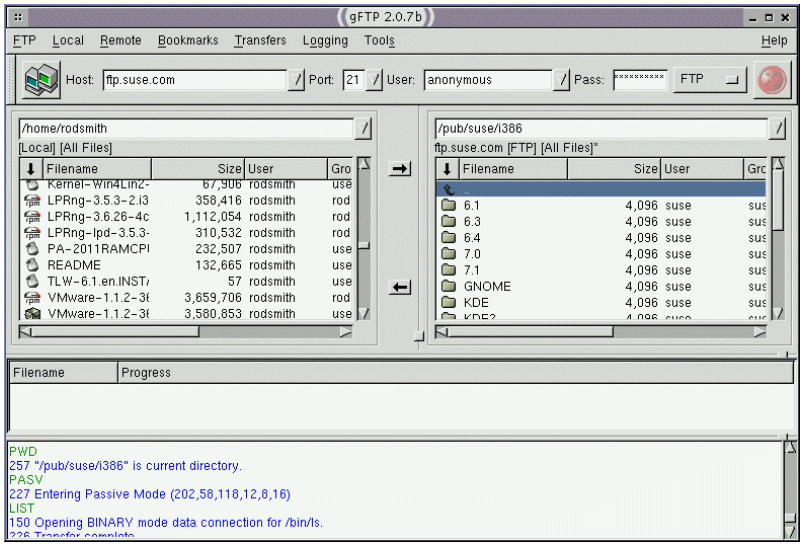
`binary` Most FTP clients default to binary mode, which transfers files without altering their contents. If you need to switch back to this mode, issue this command.

`ascii` Because different OSs store end-of-line characters in different ways, text files transferred via FTP sometimes don't display properly once retrieved. Issuing the `ascii` command causes a transfer that changes end-of-line characters to conform to the client's conventions. This is useful for text files, but should *not* be used on binary files.

`quit` To exit from an FTP session, you issue the `quit` command. This closes the session and exits from the program.

These commands are typically accessible from menu options or using buttons in GUI FTP programs. For instance, Figure 5.9 shows gFTP, with the local computer's filesystem displayed on the left and the FTP server's on the right. You can click files on one side and then click one of the arrows between the panes to move them between systems.

FIGURE 5.9 GUI FTP clients let you browse remote files using a point-and-click interface.



FTP requires that the user send a username and password to access the remote system. If the system you're contacting is one on which you have an account, you can use your ordinary login username and password on that system. This is often used, for instance, when uploading files to an ISP's Web server for your personal Web page. Many sites don't use usernames and passwords in quite this way, though; these are *anonymous FTP sites*. To use such a site, you send `anonymous` as the username and anything (conventionally your e-mail address) as a password. Anonymous FTP sites frequently hold publicly accessible files like Linux distributions and individual Linux programs. They frequently permit downloads but not uploads, although a few anonymous FTP sites accept uploads.

FTP sends its usernames and passwords in an unencrypted form. This means that the password is susceptible to interception if it passes over the Internet at large. Therefore, you should minimize your use of non-anonymous FTP whenever possible. (The insecure nature of FTP isn't a major problem for anonymous access.)

Although FTP access is often useful, it's not the most transparent or convenient way to access files on another computer. Other protocols, described in the next couple of sections, provide a more seamless integration of the server's files to the client's system.

Accessing SMB/CIFS Shares

Microsoft Windows uses a protocol that was originally called Server Message Block (SMB), but has been renamed Common Internet Filesystem (CIFS), for file and printer sharing. Using this protocol, it's possible to configure one Windows system to share a hard disk or directory with other computers. The client systems can mount the shared disk or directory as if it were a local drive. Printers can be shared in a similar manner. This type of configuration is very useful because it allows for easy file exchange between coworkers and because a network administrator can install software once on the file server rather than multiple times on each computer, saving disk space and administrative effort.

Linux includes tools that provide the ability to interact with Windows systems that use SMB/CIFS. The main package for this is called Samba, and it comes with all major Linux distributions. Samba includes two major client programs: `smbclient` and `smbmount`.

`smbclient` provides an interface to remote SMB/CIFS shares that's much like that of a text-mode FTP client. To use it, type **`smbclient //server/share`**, where *`server`* and *`share`* are the name of the server and the share you want to access, respectively. You'll be asked to provide a password. (By default, `smbclient` passes your login name as your username.) You can then use commands like `dir`, `get`, and `put` to obtain directory listings and transfer files. For instance, consider the following:

```
$ smbclient //apollo/hschmidt
Password:
```

```
smb: \> dir
  geology            D        0  Sun Apr  2 11:59:16 2000
  drrock.wpd             152576  Thu Mar 29 13:01:16 2001
         50355 blocks of size 65536. 8625 blocks available
smb: \> get drrock.wpd
smb: \> quit
```

This sequence of commands shows the retrieval of a file called `drrock.wpd` from the `hschmidt` directory on the `apollo` server. The FTP-like interface of `smbclient` is limiting, though. SMB/CIFS was intended for file sharing—the direct access to files by user programs. Therefore, Samba under Linux includes a utility called `smbmount` that will actually mount the remote share in Linux. This program is called much like `smbclient`, but you must add the Linux mount point to the call. Here's a sequence that accomplishes the same task as the preceding one:

```
$ smbmount //apollo/hschmidt /mnt/a17
Password:
$ ls -l /mnt/a17
total 152
-rwxr-xr-x 1 rodsmith users  152576 Mar 29 13:01
↳drrock.wpd
drwxr-xr-x 1 rodsmith users     512 Apr  2  2000 geology
$ cp /mnt/a17/drrock.wpd ./
$ smbumount /mnt/a17
```

The advantage of `smbmount` is that you can do much more than copy files. For instance, rather than copy `drrock.wpd` to your own computer, you can load it directly into a word processor and save the changes directly to the server.

One drawback to `smbmount` is that it assigns Linux ownership of all files on the remote server to the user who ran the command, unless you use the `-o uid=`*`UID`* option, which sets ownership to the user whose user ID is *`UID`*. You might also need to use the `-o username=`*`name`* option, to set the username used to access the shares.

For ordinary users to run `smbmount` and `smbumount`, the `smbmnt` and `smbumount` programs must have their SUID bits set, which allows ordinary users to run programs with `root` privileges. (`smbmnt` is a helper program to `smbmount`.) If this isn't the case when Samba is installed, type **`chmod a+s /usr/bin/smbmnt /usr/bin/smbumount`** as `root`. Thereafter, ordinary users will be able to use these programs, but they'll need to own the mount points they use, such as `/mnt/a17` in the preceding example.

Another way to mount SMB/CIFS shares is via the standard Linux `mount` command. This requires you to pass the filesystem type of `smbfs` with the `-t` parameter, thus:

```
# mount -t smbfs //apollo/hschmidt /mnt/a17
```

Samba is primarily a server system; the client utilities are just a small part of what Samba does. When the Samba server runs, a Linux system can function as a file server for Windows clients. The normal Samba server configuration is handled through the `smb.conf` file, which is normally stored in `/etc`, `/etc/samba`, `/etc/samba.d`, or someplace similar. Default configurations normally work to a limited extent, but need some customizations. You may need to set some of these options to use Samba as a client, as well. The two most common of these areas follows:

Workgroup or domain name You must set the workgroup or domain name to match your local network by using the `workgroup` parameter in `smb.conf`.

Encrypted passwords Samba defaults to using unencrypted passwords, but recent versions of Windows use encrypted passwords and won't fall back to unencrypted passwords. You must set the `encrypt passwords` parameter to `Yes`, and then use the separate `smbpasswd` command to add encrypted passwords for individual users, as in typing **`smbpasswd -a hschmidt`** to add a password for `hschmidt`.

You'll also need to define shares for your system. The default configuration usually includes some examples, as well as a `[homes]` share definition that gives users access to their home directories. For more information on Samba server configuration, consult the documentation at the main Samba Web site (`http://www.samba.org`) or a book on the subject, such as my *Linux Samba Server Administration* (Sybex, 2001).

Accessing NFS Shares

Like SMB/CIFS, Sun's *Network Filesystem (NFS)* is a file sharing protocol, but it was designed with the needs of Unix systems in mind. NFS includes Unix features, like support for owners, groups, and permission strings (see "File Permissions" in Chapter 4, "Users and Security," for more on these features) that aren't supported by SMB/CIFS. Because Linux hews closely to the Unix model, NFS is the preferred method for file sharing between Linux systems.

In Linux, client access to NFS exports is tightly integrated into normal Linux file-access utilities. Specifically, you use the `mount` command to mount the NFS exports, and you can then access files stored on the NFS server as if they were ordinary files. For instance, using the same example files as in the preceding SMB/CIFS example, you might issue commands like the following:

```
# mount apollo:/home/hschmidt /mnt/a17
# ls -l /mnt/a17
total 152
-rwxr-xr-x 1 rodsmith users  152576 Mar 29 13:01
↳drrock.wpd
drwxr-xr-x 1 rodsmith users     512 Apr  2  2000 geology
# cp /mnt/a17/drrock.wpd ./
# umount /mnt/a17
```

Ordinarily, only `root` may use the `mount` command. To get around this restriction, the system administrator can create an entry in `/etc/fstab` for the export. This entry should include the `user` option to give ordinary users the right to mount the filesystem, thus:

```
apollo:/home/hschmidt  /mnt/a17  nfs  user,noauto,exec  0 0
```

When the preceding line is present, ordinary users can mount the NFS export by typing **`mount /mnt/a17`**, and they can unmount it by typing **`umount /mnt/a17`**. Alternatively, omitting the `user` and `noauto` options causes the system to mount the remote filesystem automatically at system startup so that users don't need to explicitly mount the export at all; it will always be mounted (assuming the server is always up), just like local filesystems.

It's important to note that you're not required to enter a password when you access NFS exports. An NFS server allows a specified set of clients to access the exported directories in a more-or-less unrestricted manner; the server relies upon the client's security policies to prevent abuses.

On the server side, you export directories by adding entries to `/etc/exports`. For instance, the following line tells Linux to share its `/home` directory with the computers `taurus` and `littrow`, giving read/write access to `taurus` and read-only access to `littrow`:

```
/home taurus(rw) littrow(ro)
```

Using an SNMP Client

The *Simple Network Management Protocol (SNMP)* is designed to provide information about and control of a computer over a network. It's usually provided in a package called `snmpd` or `ucd-snmp`. There may be an accompanying package that provides the client features, which relies upon the main SNMP package. Linux's SNMP server is being renamed to NET-SNMP, however, so package names are likely to change as well.

SNMP is designed to allow an administrator to define a set of information that may be retrieved about a system. For instance, you might want to make the time since last reboot and the number of packets transmitted over a network interface accessible to at least some outside systems. If you run SNMP on many computers and then configure another to collect this information, you can quickly view critical information about many servers from one location.

Similarly, SNMP allows you to reconfigure a system from a remote location. SNMP can accept input values and then pass them on to other programs or place them in critical configuration files.

When you install the SNMP package, it is usually configured to run on the next reboot, but you may need to manually execute a SysV startup script to get it running before then. SNMP is configured through the `/etc/snmp/snmpd.conf` file. SNMP configuration is extremely complex, so if you would like to learn more about the subject, you should consult a book on the subject, or at least read the NET-SNMP documentation at `http://net-snmp.sourceforge.net`, before you attempt to use this package.

To use SNMP as a client to view or modify the configuration of other programs, you use commands such as `snmpget`, `snmpput`, `snmpstatus`, and `snmpwalk`. Alternatively, you can use a GUI tool like `tkmib` (included with some SNMP implementations) to browse your network for SNMP servers.

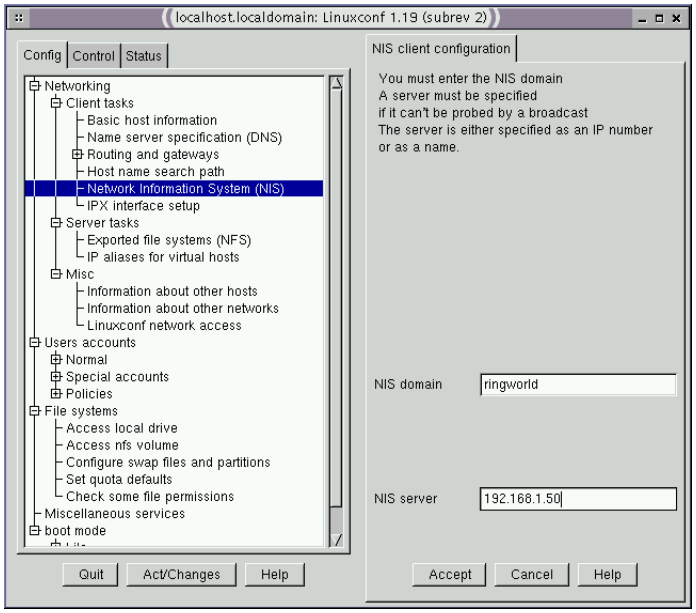
Using NIS

Network Information Service (NIS) is a protocol that's designed to simplify user authentication and related services on a network of multiple Unix or

Linux systems. There are several variants of NIS, such as NIS+, NIS YP and Switch (NYS), and Name Switch Service (NSS). The original NIS was once called Yellow Pages (YP), but that's a registered trademark in some areas, so the name was changed. Nonetheless, most NIS utilities still include `yp` in their names.

Some distributions let you configure NIS during system installation. You may be required to enter the name of the NIS domain name (which may be different from your DNS domain name), and the address of the NIS server. If you want to use NIS after installing the OS, you can configure it through a GUI configuration tool like `linuxconf`. Figure 5.10 shows the `linuxconf` module in which you configure an NIS client—Networking ➢ Client Tasks ➢ Network Information System (NIS). Enter the NIS domain and NIS server IP address in the fields that are provided. Thereafter, the standard login procedures should use the NIS username and password databases in addition to the local databases.

FIGURE 5.10 GUI configuration tools can help configure many network clients and servers, including NIS.



Setting Up a Remote Access Server

Remote access servers allow a user on one computer to run programs on another. One of the oldest remote access protocols around is Telnet, and all major Linux distributions ship with a Telnet server, which is typically called `telnetd` or `in.telnetd`. This file may be distributed in a package called `telnet`, `telnet-server`, or something else. Telnet servers are very simple, and therefore require no configuration beyond basic installation. They're normally launched from `inetd` or `xinetd`, which are programs that start other servers on an as-needed basis. The "Starting and Stopping Services" section of Chapter 6 discusses configuring these programs.

Unfortunately, Telnet suffers from the same problem as FTP—it sends passwords (and all other data) unencrypted across the network. Therefore, the SSH protocol has begun to emerge as a more secure replacement for Telnet. Until late in 2000, there were various legal barriers to the distribution of SSH, but these barriers have largely evaporated. Because of this, SSH is beginning to appear as a standard part of Linux distributions. The most popular SSH package in Linux is OpenSSH (`http://www.openssh.com`). SSH typically comes in at least two packages: a client and a server. There may also be a "common" package and support libraries.

Once all the required packages are installed and the server running, the default SSH configuration tends to work well. If necessary, though, you can fine-tune it. The normal SSH server configuration file is `/etc/ssh/sshd_config`. (There's also an `/etc/ssh/ssh_config` file that controls the SSH client.)

Some SSH packages come configured to allow `root` to log in directly. Even with the password encryption provided by SSH, this is inadvisable because it makes it too easy for somebody who has obtained the `root` password through other means to break into your system. To plug this security hole, change the `PermitRootLogin` option in `sshd_config` to `no`. Users who need to perform superuser tasks remotely can still log in as ordinary users and then use `su` to obtain the necessary privileges. This requires an outsider to have *two* passwords in order to do serious damage to the system.

Setting Remote Access Rights

One critical aspect of network operation is controlling access to the computer. The preceding discussions have touched on this matter—for

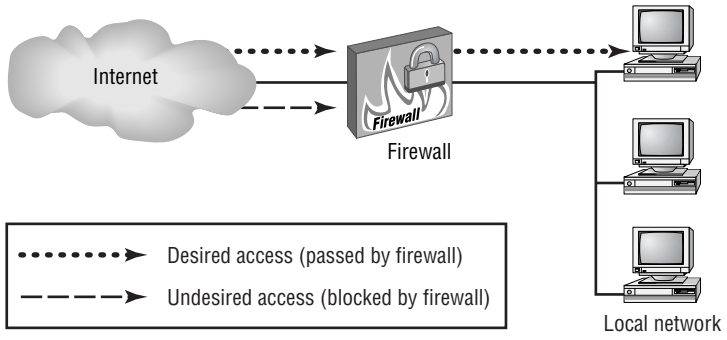
instance, the comments about preventing unauthorized mail relaying and blocking `root` logins via SSH. Nonetheless, this issue goes far beyond these few cases. Fortunately, Linux provides many means of controlling network access.

Whenever possible, apply redundant access controls. For instance, you can use both a firewall and TCP Wrappers or `xinetd` to block unwanted access to particular servers. Doing this helps protect against bugs and misconfiguration—if a problem emerges in the firewall configuration, for instance, the secondary block will probably halt the intruder. If you configure the system carefully, such an access will also leave a log file message that you'll see, so you'll be alerted to the fact that the firewall didn't do its job.

Controlling Access via a Firewall

Traditionally, firewalls have been routers that block undesired network transfers between two networks. Typically, one network is a small network under one management, and the other network is much larger, such as the Internet. Figure 5.11 illustrates this arrangement. (More complex firewalls that use multiple computers are also possible.) Dedicated external firewalls are available, and can be good investments in many cases. In fact, it's possible to turn an ordinary computer into such a device by using Linux—either with a special-purpose distribution like the Linux Router Project (`http://www.linuxrouter.org`) or by using an ordinary distribution and configuring it as a router with firewall features.

FIGURE 5.11 Firewalls can selectively pass some packets but not others, using assorted criteria.



As described earlier in this chapter, servers operate by associating themselves with particular network ports. Likewise, client programs bind to ports, but client port bindings aren't standardized. The most common type of firewall, a *packet filter*, blocks access by examining individual network packets and determining whether or not to let them pass based on the source and destination port number, the source and destination IP address, and possibly other low-level criteria, such as the network interface in a computer with more than one. For instance, in Figure 5.11, you might run a Samba file server internally, but outside computers have no business accessing that server. Therefore, you'd configure the firewall to block external packets directed at the ports used by Samba.

In addition to running a firewall on a router that serves an entire network, it's possible to run a firewall on an individual system. This approach can provide added protection to a sensitive computer, even if an external firewall protects that computer. It's also useful on computers that don't have the protection of a separate firewall, such as many broadband-connected systems.

Either way, Linux uses the `ipfwadm`, `ipchains`, and `iptables` tools to configure firewall functions. These tools are designed for the 2.0.*x*, 2.2.*x*, and 2.4.*x* kernels, respectively. (The 2.4.*x* kernel series includes the ability to use the older tools, but only as a compile-time option.) There are several ways you can configure a firewall:

Manually You can read up on the syntax of the tool used to configure your kernel and write your own script. Although this is possible, it's difficult for new Linux administrators because the options are both complex and subtle, making it easy to produce a firewall that doesn't do what you want. If you choose to pursue this path, I recommend that you read a book on the subject, such as Robert L. Ziegler's *Linux Firewalls* (New Riders, 1999).

With the help of a GUI configuration tool A few GUI configuration tools are available for Linux firewall configuration, such as Firestarter (`http://firestarter.sourceforge.net`) and Guarddog (`http://www.simonzone.com/software/guarddog`). A few distributions, such as Red Hat, are now shipping with such tools, as well. These tools let you specify certain basic information, such as the network port and the client and server protocols you wish to allow, and they generate firewall scripts that can run automatically when the system boots.

With the help of a Web site Robert Ziegler, the author of *Linux Firewalls*, has made a Web site available that functions rather like the GUI configuration tools but via the Web. You enter information on your system, and the Web site generates a firewall script. This tool is available at `http://linux-firewall-tools.com/linux`.

If you use a GUI tool or Web site, be sure it supports the firewall tool your kernel requires. As of May of 2001, only Firestarter supports `iptables`, but that could change by the time you read this. Also, you shouldn't consider a firewall to be perfect protection. You might create a configuration that actually contains flaws, or there might be flaws in the Linux kernel code that actually implements the firewall rules.

One of the advantages of a firewall, even to protect just one computer, is that it can block access attempts to *any* server. Most other measures are more limited. For instance, TCP Wrappers protects only servers configured to be run via TCP Wrappers from `inetd`; and passwords are good only to protect the servers that are coded to require them.

Controlling Access via TCP Wrappers

One popular means of running servers is via `inetd`, a server that listens for network connections on behalf of other servers and then launches the target servers as required. This approach can reduce the RAM requirements on a server computer when the server programs are seldom in use because only `inetd` need be running at all times. Chapter 6 covers `inetd` in more detail.

Not all Linux systems use `inetd`. Mandrake and Red Hat have switched to `xinetd`, which includes its own access control features. TCP Wrappers isn't normally used in conjunction with `xinetd`.

One further advantage of `inetd` is that it can be used in conjunction with another package, known as TCP Wrappers. This package uses a program known as `tcpd`. Instead of having `inetd` call a server directly, `inetd` calls `tcpd`, which does two things: It checks whether a client is authorized to access the server; and if the client has this authorization, `tcpd` calls the server program.

TCP Wrappers is configured through two files: `/etc/hosts.allow` and `/etc/hosts.deny`. The first of these specifies computers that are allowed access to the system in a particular way, the implication being that systems not listed are not allowed access. `hosts.deny`, by contrast, lists computers that are not allowed access; all others are given permission to use the system. If a system is listed in both files, `hosts.allow` takes precedence.

Both files use the same basic format. The files consist of lines of the following form:

```
daemon-list : client-list
```

The *`daemon-list`* is a list of servers, using the names for the servers that appear in `/etc/services`. There are also wildcards available, such as `ALL` for all servers.

The *`client-list`* is a list of computers to be granted or denied access to the specified daemons. You can specify computers by name or by IP address, and you can specify a network by using (respectively) a leading or trailing dot (`.`). For instance, `.luna.edu` blocks all computers in the `luna.edu` domain, and `192.168.7.` blocks all computers in the 192.168.7.0/24 network. You can also use wildcards in the *`client-list`*, such as `ALL` (all computers). `EXCEPT` causes an exception. For instance, when placed in `hosts.deny`, `192.168.7. EXCEPT 192.168.7.105` blocks all computers in the 192.168.7.0/24 network except for 192.168.7.105.

The `hosts.allow` and `hosts.deny` man pages (they're actually the same document) provide additional information on more advanced features. You should consult them as you build TCP Wrappers rules.

Remember that not all servers are protected by TCP Wrappers. Normally, only those servers that `inetd` runs via `tcpd` are so protected. Such servers typically include, but are not limited to, Telnet, FTP, TFTP, `rlogin`, finger, `linuxconf`, POP, and IMAP servers. A few servers can independently parse the TCP Wrappers configuration files, though; consult the server's documentation if in doubt.

Controlling Access via *xinetd*

In 2000 and 2001, the shift began to `xinetd` from `inetd`. Although `xinetd` *can* use TCP Wrappers, it normally doesn't because it incorporates similar functionality of its own. In 2001, the distributions that use `xinetd` use a

main configuration file called /etc/xinetd.conf; but this file is largely empty because it calls separate files in the /etc/xinetd.d directory to do the real work. This directory contains separate files for handling individual servers. Chapter 6 includes information on basic xinetd configuration. For now, know that security is handled on a server-by-server basis through the use of configuration parameters, some of which are similar to the function of hosts.allow and hosts.deny:

bind This option tells xinetd to listen on only one network interface for the service. For instance, you might specify bind = 192.168.23.7 on a router to have it listen only on the Ethernet card associated with that address. This feature is extremely useful in routers, but it is not very useful in computers with just one network interface. A synonym for this option is interface.

only_from You can specify IP addresses, networks (as in 192.168.78.0/24), or computer names in this line, separated by spaces. The result is that xinetd will accept connections only from these addresses, similar to TCP Wrappers' hosts.allow entries.

no_access This option is the opposite of only_from; you list computers or networks here that you want to blacklist. This is similar to the hosts.deny file of TCP Wrappers.

You should enter these options into the files in /etc/xinetd.d that correspond to the servers you want to protect. Place the lines between the opening brace ({) and closing brace (}) for the service. If you want to restrict *all* your xinetd-controlled servers, you can place the entries in the defaults section in /etc/xinetd.conf.

Some servers provide access control mechanisms similar to those of TCP Wrappers or xinetd by themselves. For instance, Samba provides hosts allow and hosts deny options that work much like the TCP Wrappers file entries, and NIS includes similar configuration options. These options are most common on servers that are awkward or impossible to run via inetd or xinetd.

Controlling Access via Passwords

Servers differ in who they'll serve. Some, such as Web servers and SMTP e-mail servers, are generally configured as publicly accessible. Anybody may access these servers, although the servers may not do all things for all people—for instance, a mail server might relay mail only for certain computers. Others, such as Telnet and FTP servers, require that users authenticate themselves in some way. This method is often a password, which usually corresponds to the user's normal login password.

Passwords can be a useful security tool, but they're not without their flaws. For one thing, as noted earlier, some protocols send passwords in an unencrypted form by default. These include, but are not limited to, Telnet, FTP, POP, and IMAP. There are secure variants of many of these protocols, which send the password, and sometimes other data, in an encrypted form. The encryption prevents the password from being used even if it's intercepted. Other protocols, such as SSH, are designed from the ground up to be secure.

Some protocols use a method of authentication other than a password. SSH, for instance, can be configured to use special keys on the client and server. When SSH is so configured, the user need not enter a password. Although this configuration provides for secure authentication, this configuration is risky should the user's normal client computer be compromised, because the encrypted key is stored on the hard disk, and so it can be stolen.

Still more risky is the method used by `rlogin`, `rsh`, and `rcp`—the so-called "r commands." These commands allow a user on one Unix or Linux system to log in to, run programs on, or copy files to or from another Unix or Linux system without being authenticated. Like NFS shares, the server system trusts the client system's authentication mechanism. Individual users can control their own r-command access. This is done by creating a file called `.rhosts` in the user's home directory. This file contains a list, one per line, of hosts that are trusted, optionally followed by a username on the host. (If no username is specified, the system requires the same login name on both computers.) It's generally unwise to leave the r-command servers running, if they're configured to run at all; they're just too much of a security risk. If you must use these servers, use a firewall and TCP Wrappers or `xinetd` options to restrict access to just those computers that require the access.

Controlling Access via File Permissions

Finally, remote access rights can be restricted by the use of file permissions. These are described more fully in Chapter 4. In brief, though, any server runs as if it were a particular user. Servers that require logins, such as FTP, Telnet, and SSH servers, typically give the user the access rights associated with the username used to gain access. Therefore, if an individual logs in as `ecernan`, that person may read, write, and modify files for which `ecernan` has access rights.

Some servers, though, don't use passwords. These typically run with the rights of one account that's specified by a startup script or super server. The `nobody` account is a common choice for this. `nobody` is a low-access account; normally, the user `nobody` can't read files unless they're given world read access. Many servers create their own low-privilege users so that these matters can be fine-tuned for a particular server. By carefully planning the users associated with servers and your file permission scheme, you can limit the access given to servers' users.

File permission restrictions are most useful in giving anonymous users partial access to your system. For instance, you might want an FTP or Web server to give clients access to some directories, but not others. You can use file permissions to ensure that the server can read only those files that are in the target directories, and no others. (There are often other ways to achieve similar effects, such as via server-specific configuration options or by using the `chroot` command to launch a server in an environment in which it cannot read outside of a specified set of directories. This second option is an advanced one that's beyond the scope of this book.)

Remote System Administration

One of the reasons it's so important to control remote access rights is that many different protocols can be used to provide administrative access to a Linux computer. Although using such protocols can pose a security risk, remote administration is often extremely convenient, or even necessary in some situations. You can use several types of tools to remotely administer your Linux system, including text-mode logins, GUI logins, file transfers, and dedicated remote administration protocols.

Text-Mode Logins

The earlier section, "Setting Up a Remote Access Server," described setting up a couple different types of servers that accept text-mode logins from distant systems: Telnet and SSH. You can use either of these to administer one system from another—even from a computer running another OS, like Windows or MacOS. Typically, you log in using a regular user account, and then you use `su` to enter the `root` password to acquire superuser privileges. Thereafter, you can do almost anything you could do from a text-mode login at the console.

Telnet passes all data in an unencrypted form. This means that both your ordinary user's login password and the `root` password you enter in conjunction with `su` might be intercepted by an unscrupulous individual on the source, destination, or any intervening network. For this reason, it's best not to use Telnet for remote administration. For that matter, if it's possible, you should totally avoid using Telnet. SSH encrypts all the data that pass between two systems, and so it is a much better choice for remote administration.

To use Telnet from Linux, you type **`telnet`** ***`hostname`***, where *`hostname`* is the DNS hostname of the computer you wish to contact. You'll then see the remote system's login prompt. The entire procedure looks like this:

```
$ telnet apollo.luna.edu
Trying 192.168.1.1...
Connected to apollo.luna.edu.
Escape character is '^]'.

Caldera OpenLinux(TM)
Version 2.4 eDesktop
Copyright 1996-2000 Caldera Systems, Inc.

login: ecernan
Password:
You have old mail in /var/spool/mail/ecernan.
Last login: Sat May  5 13:05:29 2001 from gemini on 4
[ecernan@apollo ecernan]$
```

At this point, anything you type (aside from Ctrl+], which is an "escape" character to let you enter commands into your local Telnet program) is processed by the remote system. You can use `su` to acquire `root` privileges, edit files with Vi, Emacs, or any other text-based editor, and so on.

SSH works in a similar way, except that you don't see the `login:` prompt; SSH passes your current username to the server, which attempts to use the same username to authenticate you. If you want to use a different username on the server than on your current system, you should include the `-l` *`username`* parameter on the command line, thus:

```
$ ssh apollo.luna.edu -l ecernan
ecernan@apollo.luna.edu's password:
Last login: Mon May  7 11:14:51 2001 from gemini.luna.edu
[ecernan@apollo ecernan]$
```

The first time you make a connection to a given server, you'll see a message informing you that the authenticity of the server can't be verified. The message goes on to display a code associated with the server. If you want to continue connecting, type **yes** in response to the query about this.

You may omit the `-l` *`username`* parameter if your username is the same on both systems. Once you've logged in with SSH, you can use the system much as you would from a Telnet login or from the console—by typing text-mode commands, editing files with text-mode editors, and so on. Because SSH encrypts all data, it's extremely unlikely that your original password, or the password you type when you use `su`, will be usable to anybody who intercepts the data stream.

There are other remote text-mode login tools besides Telnet and SSH. One particularly common tool is `rlogin`, which uses a trusted hosts security model, in which the server relies upon the client to authenticate users. As described earlier, `rlogin` is a potential security vulnerability. Because of this, it's best to either completely eliminate the `rlogin` server (typically called `/usr/sbin/in.rlogind`) using your package management tools (as described in Chapter 3, " Software Management") or stop the server from running (as described in Chapter 6).

The `rlogin` server is often included in a package along with other utilities that you may need. Don't remove the package to which this server belongs without first verifying that you don't need its other programs.

GUI Logins

If you want to use GUI administration tools remotely, you can do so, but you'll need appropriate software on the system you're using to access the Linux computer. Normally, this is an X server, as described earlier in this chapter, in "Using X Programs Remotely." Because all major Linux distributions include X servers, it's usually possible to use a Linux computer as a terminal for GUI configuration of another. (The main exception to this is if you've not installed X on the computer that you want to use to administer another.) Likewise, you can use a Windows system running an X server or VNC client (if you've installed the VNC server on Linux) to remotely control a Linux system with a GUI.

Once you've logged on, you can use the `su` command to acquire root privileges, just as you can when using a text-mode login. You can then run GUI administrative tools like Red Hat's `linuxconf`, SuSE's YaST2, or Caldera's COAS.

Neither X nor VNC encrypts most data transmitted over the network, although VNC encrypts its initial password. Therefore, when you issue the `su` command to acquire `root` privileges, you'll send the root password unencrypted. As a result, it's possible that it will be compromised. The simplest solution to this problem is usually to use SSH to make the initial connection. When properly configured, SSH will tunnel the X protocols through its own encrypted connection. This will slow down the display slightly, but it will protect the data (including passwords you type) from prying eyes.

File Transfers

Although generally not thought of as such, file transfer tools can be useful in remote administration. If you like, you can edit a configuration file on one

system and transfer it to another system. You might want to do this if one system has more sophisticated editors or configuration checking tools than does another system. For instance, if you're administering a print server on which you have only bare-bones tools, you might want to modify the configuration files in a more comfortable environment on some other computer and then transfer the configuration files to the print server. (This would require the print server to be running some file-transfer server like FTP, NFS, or Samba, of course.)

When using file transfers in this way, it's generally not a good idea to give direct access to the target directory for the configuration files. For instance, you probably shouldn't export a system's `/etc` directory using NFS or Samba. Although doing so makes it easy to read and write configuration files, it also makes it that much easier for an intruder to modify these files, especially if there is a flaw in the server or its configuration. Instead, you should transfer files to and from an ordinary user account and then use a remote login protocol, such as SSH, to enable the copying of files from that account to their ultimate destinations.

Remote Administration Protocols

There are several tools designed to allow you to administer a computer remotely. To do so, you'll need to run the server version of one of these tools on the computer you plan to administer, and you'll need to run a client on the system you intend to use to do the administration. (Many of these tools use ordinary Web browsers as clients, so you can administer a Linux computer from any system that supports a Web browser, even if it's not a Linux computer itself.) Examples of these tools include the following:

SNMP This protocol was described earlier in this chapter. It was designed as a remote administration protocol, but it requires fairly tedious configuration on the system that's to be administered. It also requires specialized client programs. For these reasons, it's never become a very popular Linux administration protocol.

SWAT The *Samba Web Administration Tool (SWAT)* is, as the name implies, a Web-based means of administering a Samba server. Once configured, SWAT can be accessed on port 901 using an ordinary Web browser. You specify the port number by adding a colon (`:`) and the number to the URL; so to administer `apollo.luna.edu`, you'd enter `http://apollo.luna.edu:901` in a Web browser. SWAT is limited to administering the Samba server functions of a computer, which limits the utility of this tool. SWAT provides unusually complete control of Samba, however.

Webmin Webmin is an ambitious Web-based administration tool. Its ambitiousness derives from the fact that it aims to allow Web-based administration of multiple Linux distributions (and other Unix-like systems) that use different configuration files. It accomplishes this goal by installing a series of configuration modules that are unique to each distribution. Once installed and running, Webmin binds to port 10000, so you'd enter `http://apollo.luna.edu:10000` in a Web browser to administer `apollo.luna.edu`. You can read more about Webmin on its Web page, `http://www.webmin.com/webmin`.

Remote administration tools frequently send passwords in an unencrypted form, so they're potentially dangerous tools to use except on well-protected local networks. Webmin supports using Secure Sockets Layer (SSL) to encrypt transmissions, but doing so requires extra configuration (consult the Webmin Web page for more details).

Web administration tools may be started using either standalone configurations or a super server. Both options are discussed in the "Starting and Stopping Services" section of Chapter 6.

Network Diagnostic Tools

Network configuration is a complex topic, and unfortunately, things don't always work as planned. Fortunately, there are a few commands you can use to help diagnose a problem. The simplest of these is `ping`. This command sends a simple packet to the system you name (via IP address or hostname) and waits for a reply. In Linux, `ping` continues sending packets once every second or so until you interrupt it with a Ctrl+C keystroke. Here's an example of its output:

```
$ ping speaker
PING speaker.rodsbooks.com (192.168.1.1) from 192.168.1.3
↳: 56(84) bytes of data.
64 bytes from speaker.rodsbooks.com (192.168.1.1): icmp_
↳seq=0 ttl=255 time=149 usec
```

```
64 bytes from speaker.rodsbooks.com (192.168.1.1): icmp_
↳seq=1 ttl=255 time=136 usec
64 bytes from speaker.rodsbooks.com (192.168.1.1): icmp_
↳seq=2 ttl=255 time=147 usec
64 bytes from speaker.rodsbooks.com (192.168.1.1): icmp_
↳seq=3 ttl=255 time=128 usec

--- speaker.rodsbooks.com ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.128/0.140/0.149/0.008 ms
```

This command sent four packets and waited for their return, which occurred quite quickly (in an average of 0.140ms) because the target system was on the local network. By pinging systems on both local and remote networks, you can isolate where a network problem occurs. For instance, if you can ping local systems but not remote systems, the problem is most probably in your router configuration. If you can ping by IP address but not by name, the problem is with your DNS configuration.

Another useful diagnostic tool is `netstat`. This is something of a Swiss Army knife of network tools because it can be used in place of several others, depending upon the parameters it is passed. It can also return information that's not easily obtained in other ways. Some examples include the following:

Interface information Pass `netstat` the `--interface` or `-i` parameter to obtain information on your network interfaces similar to what `ifconfig` returns. (Some versions of `netstat` return information in the same format, but others display the information differently.)

Routing information You can use the `--route` or `-r` parameter to obtain a routing table listing similar to what the `route` command displays.

Masquerade information Pass `netstat` the `--masquerade` or `-M` parameter to obtain information on connections mediated by the `ipchains` or `iptables` tools. (These tools allow a Linux router to perform IP masquerading, in which a Linux system "hides" a network behind a single IP address. This can be a good way to stretch limited IP addresses.)

Program use Some versions of `netstat` support the `--program` or `-p` parameters, which attempt to provide information on the programs that are using network connections. This attempt isn't always successful, but it often is, so you can see what programs are making outside connections.

Open ports When used with various other parameters, or without any parameters at all, `netstat` returns information on open ports and the systems to which they connect.

`netstat` is a very powerful tool, and its options and output aren't entirely consistent from one distribution to another. You may want to peruse its man page and experiment with it to learn what it can do.

Summary

Networking is very important to many modern Linux systems, which frequently function as servers or workstations on local networks. Networks operate by breaking data into individual packets in a manner that's dictated by the particular protocol stack in use by the system. Linux includes support for several protocol stacks, the most important of which is TCP/IP, the protocol stack upon which the Internet is built. You can configure Linux for TCP/IP networking by using DHCP to automatically obtain an address, by entering the information manually, or by establishing a PPP link. You can do any of these things using text-mode or GUI tools, although the GUI tools aren't standardized across different distributions.

Once Linux is connected to a network, you can use any of many client programs to access resources on the network (either the local network or the Internet). File transfer and sharing, Web browsing, e-mail, and remote access are just some of the applications of networking possible in Linux.

Because most Linux distributions ship with a large number of network servers, it's important that you know how to restrict access to your system. If you don't, chances are that sooner or later somebody undesirable will gain unauthorized access to your system. Firewalls, TCP Wrappers, `xinetd`, passwords, and file permissions can all be used to protect your computer.

Exam Essentials

Determine appropriate network hardware for a Linux computer. If the computer is to be used on an existing network, you must obtain a network card of a type that's compatible with that network, such as Ethernet, Token Ring, and so on. If you're building a new local network, Ethernet is the most common choice, although more exotic alternatives are also available and may be suitable in some specific situations.

Describe the information needed to configure a computer on a static IP network. Four pieces of information are critical: the IP address, the netmask (aka network mask or subnet mask), the network's gateway address, and the address of at least one DNS server.

Summarize the function of PPP. The Point-to-Point Protocol negotiates a TCP/IP connection, typically acquiring requisite information from the PPP server. It's used to connect computers via telephone lines, and is used in modified form for some broadband links.

Explain the nature of X clients and servers. An X server controls a screen display and handles input from the user's mouse and keyboard. Therefore, the X server is used directly by the user, and X clients are the programs that rely upon the X server's services.

Summarize common access control mechanisms. Firewalls, TCP Wrappers, and `xinetd` can all control access to particular ports, either by blocking them entirely or by controlling access to the servers that run on those ports. Passwords and file permissions can control access to individuals, by requiring authentication or restricting access to specific files once a user has gained entry to the system.

Describe how a Linux system may be administered remotely. Remote administration may be achieved through text-mode login protocols like Telnet or SSH, through remote GUI sessions (X or VNC), or through specialized remote administration tools like SWAT or Webmin.

Commands in This Chapter

Command	Description
`ifconfig`	Configures a network interface, or displays information on that configuration
`route`	Configures a routing table entry, or displays information on the routing table
`smbclient`	Transfers files to and from SMB/CIFS shares using an FTP-like user interface
`smbmount`	Mounts an SMB/CIFS share in the Linux filesystem tree
`smbumount`	Unmounts an SMB/CIFS share from the Linux filesystem tree
`netstat`	Displays information on a Linux computer's network configuration or the processes that use network resources

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

Address Resolution Protocol (ARP)

anonymous FTP site

AppleTalk

broadband

broadcast

client

default route

Domain Name System (DNS)

domain name

Dynamic Host Configuration Protocol (DHCP)

envelope

Fiber Distributed Data Interface (FDDI)

Fibre Channel

File Transfer Protocol (FTP)

frame

full-duplex

gateway

gigabit Ethernet

half-duplex

hardware address

High-Performance Parallel Interface (HIPPI)

hostname

hub

Integrated Services Digital Network (ISDN)

internet

Internet

Internet Message Access Protocol (IMAP)

Internet Packet Exchange (IPX)

IP address

IPv6

LocalTalk

machine name

Media Access Control (MAC) address

NetBEUI

NetBIOS

netmask

Network Filesystem (NFS)

Network Information Service (NIS)

network mask

open mail relay

Open System Interconnection (OSI) model

packet filter

packet

payload

Point-to-Point Protocol (PPP)

port number

Post Office Protocol (POP)

protocol stack

pull mail protocol

push mail protocol

Requests for Comment (RFC)

router

Samba Web Administration Tool (SWAT)

Sequenced Packet Exchange (SPX)

Simple Mail Transfer Protocol (SMTP)	switch
Simple Network Management Protocol (SNMP)	terminal program
spam	Token Ring
subdomain	Transmission Control Protocol/ Internet Protocol (TCP/IP)
subnet mask	

Review Questions

1. Which types of network hardware does Linux support? (Choose all that apply.)
   - **A.** Token Ring
   - **B.** Ethernet
   - **C.** DHCP
   - **D.** Fibre Channel

2. Which of the following is a valid IP address on a TCP/IP network?
   - **A.** 202.9.257.33
   - **B.** 63.63.63.63
   - **C.** 107.29.5.3.2
   - **D.** 98.7.104.0/24

3. Which of the following is *not* a Linux DHCP client?
   - **A.** `pump`
   - **B.** `dhcpcd`
   - **C.** `dhcpd`
   - **D.** `dhclient`

4. You try to set up a computer on a local network via a static TCP/IP configuration, but you lack a gateway address. Which of the following is true?
   - **A.** Because the gateway address is necessary, no TCP/IP networking functions will work.
   - **B.** TCP/IP networking will function, but you'll be unable to convert hostnames to IP addresses or vice-versa.
   - **C.** You'll be able to communicate with machines on your local network segment but not with other systems.
   - **D.** The computer won't be able to tell which other computers are local and which are remote.

5. Which of the following types of information is returned by typing **`ifconfig eth0`**? (Choose all that apply.)

   A. The names of programs that are using `eth0`

   B. The IP address assigned to `eth0`

   C. The hardware address of `eth0`

   D. The hostname associated with `eth0`

6. In what way do GUI network configuration tools simplify the network configuration process?

   A. They're the only way to configure a computer using DHCP, which is an easier way to set networking options than static IP addresses.

   B. They provide the means to configure PPPoE or PPPoA, which are easier to configure than DHCP or static IP addresses.

   C. Once running, they provide easy-to-find labels for options, obviating the need to locate appropriate configuration files.

   D. They're consistent across distributions, making it easier to find appropriate options on an unfamiliar distribution.

7. Which of the following pieces of information is usually required to initiate a PPP connection over an analog telephone line? (Choose all that apply.)

   A. The ISP's telephone number

   B. The client IP address

   C. An account name (username)

   D. A password

8. What is the usual full speed of ISDN service?

   A. 56Kbps

   B. 128Kbps

   C. 608Kbps

   D. 1.5Mbps

9. Which is the ideal interface type for a broadband modem under Linux?

   A. Ethernet

   B. USB

   C. RS-232 serial

   D. Internal PCI card

10. You want to use an X server on an old Pentium computer to run X clients on a modern Alpha CPU system, with the goal of performing computationally intensive spreadsheet calculations. Which of the following is true?

    A. The spreadsheet will compute slowly because of the slow speed of the Pentium server.

    B. You won't be able to run the spreadsheet because the Alpha and Pentium CPUs need different executables.

    C. The computation will run swiftly, but graphics displays may be slowed by the Pentium's limited speed.

    D. Computations will run swiftly only if the Alpha computer makes its filesystem available via NFS.

11. You want to allow Linux users running StarOffice to directly edit files stored on a Windows 2000 SMB/CIFS file server. Which of the following would you use to enable this?

    A. Linux's standard NFS file sharing support

    B. An FTP server running on the Windows system

    C. The Linux `smbclient` program

    D. The Linux `smbmount` program

12. How does an NFS server determine who may access files it's exporting?

A. It uses the local file ownership and permission in conjunction with the client's user authentication and a list of trusted client computers.

B. It uses a password that's sent in unencrypted form across the network.

C. It uses a password that's sent in encrypted form across the network.

D. It uses the contents of individual users' `.rlogin` files to determine which client computers may access a share.

13. Why might you configure a Linux computer to function as an NIS client?

A. To mount remote filesystems as if they were local

B. To defer to a network's central authority concerning user authentication

C. To set the system's clock according to a central time server

D. To automatically obtain IP address and other basic network configuration information

14. What function does SNMP fill?

A. It allows remote systems to send mail to users of the computer.

B. It allows for remote monitoring and configuration of a computer.

C. It monitors several network ports and runs other servers as required.

D. It retrieves mail from a remote system using the POP protocol.

15. Why is it unwise to allow `root` to log on directly using SSH?

A. Somebody with the `root` password but no other password could then break into the computer.

B. The `root` password should never be sent over a network connection; allowing `root` logins in this way is inviting disaster.

C. SSH stores all login information, including passwords, in a publicly readable file.

D. When logged on using SSH, `root`'s commands can be easily intercepted and duplicated by undesirable elements.

16. A server/computer combination appears in both `hosts.allow` and `hosts.deny`. What's the result of this configuration when TCP Wrappers runs?

    A. TCP Wrappers refuses to run and logs an error in `/var/log/messages`.

    B. The system's administrator is paged to decide whether to allow access.

    C. `hosts.deny` takes precedence; the client is denied access to the server.

    D. `hosts.allow` takes precedence; the client is granted access to the server.

17. When is the `bind` option of `xinetd` most useful?

    A. When you want to run two servers on one port

    B. When you want to specify computers by name rather than IP address

    C. When `xinetd` is running on a system with two network interfaces

    D. When resolving conflicts between different servers

18. How do you change the password used by `rlogin`?

    A. Use the `rpasswd` command.

    B. Change the normal user account password.

    C. Change the Samba encrypted password.

    D. You can't; `rlogin` doesn't use passwords.

19. Which of the following types of access do servers that require usernames and passwords normally provide?

    A. Access with the permissions of the `nobody` account

    B. Access with the permissions of the account under which name they're run

    C. Access with `root` permissions

    D. Access with the permissions of the username that's entered

20. Which of the following tools may you run on a Linux computer to allow you to administer it remotely? (Choose all that apply.)

A. Netscape

B. TCP Wrappers

C. An SSH server

D. Webmin

Answers to Review Questions

1. A, B, D. Ethernet is the most common type of network hardware for local networks in 2001. Linux supports it very well, and Linux also includes support for Token Ring and Fibre Channel network hardware. DHCP is a protocol used to obtain a TCP/IP configuration over a TCP/IP network. It's not a type of network hardware, but it can be used over hardware that supports TCP/IP.

2. B. IP addresses consist of four 1-byte numbers (0-255). They're normally expressed in base 10 and separated by periods. 63.63.63.63 meets these criteria. 202.9.257.33 includes one value (257) that's not a 1-byte value. 107.29.5.3.2 includes five 1-byte numbers. 98.7.104.0/24 is a network address—the trailing /24 indicates that the final byte is a machine identifier and the first three bytes specify the network.

3. C. `dhcpd` is the Linux DHCP *server*. The others are all DHCP clients. Most distributions ship with just one or two of the DHCP clients.

4. C. The gateway computer is a router that transfers data between two or more network segments. As such, if a computer isn't configured to use a gateway, it won't be able to communicate beyond its local network segment. (If your DNS server is on a different network segment, name resolution via DNS won't work, although other types of name resolution, such as `/etc/hosts` file entries, will still work.)

5. B, C. When used to display information on an interface, `ifconfig` shows the hardware and IP addresses of the interface, the protocols (such as TCP/IP) bound to the interface, and statistics on transmitted and received packets. This command does *not* return information on programs using the interface or the hostname associated with the interface.

6. C. Once you know what tool to run in a distribution, it's usually not difficult to find the label for any given network configuration option in a GUI tool. You can configure DHCP, PPPoA, and PPPoE in text mode (and the latter two are arguably more complex than DHCP). GUI configuration tools, although they provide similar functionality, are not entirely consistent from one distribution to another.

7. A, C, D. You need a telephone number to dial the call (although this is *not* needed for a PPPoE or PPPoA broadband connection). Most ISPs use a username and password to authenticate access. Although you can specify an IP address, this option is only used in specialized circumstances.

8. B. ISDN usually operates at 128Kbps, although it can drop back to 64Kbps if one data channel is required for voice service. 608Kbps and 1.5Mbps are both well above ISDN's capabilities; these are broadband speeds.

9. A. Ethernet broadband interfaces allow you to use any supported Ethernet card. USB and internal broadband modems require special Linux drivers, which are rare. RS-232 serial broadband interfaces are extremely rare, and they don't permit true broadband speeds (most RS-232 serial ports top out at about 115Kbps).

10. C. The X server handles the display and user input only, so its speed will influence graphics displays. Computations occur on the fast Alpha-based X client system.

11. D. `smbmount` allows you to mount a remote SMB/CIFS share as if it were a local disk. Linux's NFS support would work if the Windows system was running an NFS server, but the question specifies that it's using SMB/CIFS, not NFS. An FTP server on the Windows system would allow file transfers, but not direct file access. The same would be true for the Linux `smbclient` program.

12. A. NFS uses a "trusted host" policy to let clients police their own users, including access to the NFS server's files. NFS does not use a password, nor does it use the `.rlogin` file in users' home directories.

13. B. NIS functions as a means of distributing database information across a network, most notably including user authentication information. It's not used for file sharing, clock setting, or distributing basic TCP/IP configuration information.

14. B. SNMP is a network management protocol. Option A describes SMTP. Option C describes the function of `inetd` or `xinetd`. Option D describes a program called fetchmail, which isn't a server at all.

15. A. Allowing only normal users to log in via SSH effectively requires two passwords for any remote `root` maintenance, improving security. SSH encrypts all connections, so it's unlikely that the password, or commands issued during an SSH session, will be intercepted. (Nonetheless, some administrators prefer not to take even this small risk.) SSH doesn't store passwords in a file.

16. D. TCP Wrappers uses this feature to allow you to override broad denials by adding more specific explicit access permissions to `hosts.allow`, as when setting a default deny policy (`ALL : ALL`) in `hosts.deny`.

17. C. The `bind` option of `xinetd` lets you tie a server to just one network interface, rather than link to them all. It has nothing to do with running multiple servers on one port, specifying computers by hostname, or resolving conflicts between servers.

18. D. `rlogin` relies on the client system to authenticate users. To control access, you specify trusted clients in the user's `.rhosts` file.

19. D. One of the reasons for requiring a username is to determine whose permissions to use when granting access to the filesystem.

20. C, D. An SSH server enables you to log in and use normal text-based configuration utilities to administer a system. Webmin is a specialized remote administration tool that lets you administer a system from any computer with a Web browser. Although Netscape is such a Web browser, its installation on the computer you intend to administer remotely won't enable remote administration, because it's only a client. TCP Wrappers can be an important security tool in preventing unauthorized administrative access, but it doesn't enable remote administration by itself.

Managing Files and Services

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **3.3 Set environment variables (e.g., `PATH`, `DISPLAY`, `TERM`).**
- ✓ **3.11 Identify the purpose and characteristics of configuration files (e.g., BASH, `inittab`, `fstab`, `/etc/*`).**
- ✓ **3.12 Edit basic configuration files (e.g., BASH files, `inittab`, `fstab`).**
- ✓ **3.14 Document the installation of the operating system, including configuration.**
- ✓ **4.13 Manage runlevels using `init` and `shutdown`.**
- ✓ **4.14 Stop, start, and restart services (daemons) as needed (e.g., init files).**
- ✓ **4.17 Manage and navigate the Graphical User Interface (e.g., menus, xterm).**
- ✓ **4.18 Program basic shell scripts using common shell commands (e.g., `grep`, `find`, `cut`, `if`).**
- ✓ **5.9 Document work performed on a system.**

Much of what it takes to manage a Linux computer is handling specific configuration files, setting environment variables, and managing services and runlevels. These tasks control much of what a Linux computer does, from its startup process to the default information that is available to various programs. Therefore, this chapter covers these tasks, beginning with configuration files and moving on to environment variables, services, and runlevels.

This chapter also covers a few additional administrative and user tasks. These include handling Linux's GUI environment, shell scripting, and documenting your system's configuration. GUI use and possibly shell scripting can be important even to non-administrators, although shell scripts are particularly handy in dealing with administrative tasks.

Basic Configuration File Locations

Before you can edit configuration files, you must be able to locate them. As a multiuser OS, Linux stores two types of configuration files: user and system. User configuration files store settings that apply to individual users. These may include things like users' desktop icon placements, window manager preferences, and scripts that run automatically when starting a command prompt shell. In theory, individual users are usually responsible for controlling their own configuration files, but a system administrator must be able to handle this task as well. For one thing, a system administrator is a user as well as the administrator. For another thing, users may come to the system administrator when they damage their configurations. `root` can edit any file on the system, including users' own configuration files, and so `root` can bail them out of any trouble they've created for themselves.

System configuration files, by contrast, control the entire system. Some of these provide defaults for users' configurations, but others control the system as a whole. This second class of configuration file is particularly critical because if there is a problem with some of these files, the system may misbehave quite seriously, or possibly even fail to boot.

User Configuration Files

User configuration files are stored in the users' home directories. These are usually in the `/home` directory tree, in subdirectories named after the account names. For instance, `/home/theo` is the likely home directory for a user called `theo`. These directory locations may be changed, however. To see how, refer to Chapter 4,"Users and Security," which describes configuring accounts and home directory locations.

Within the user's home directory, configuration files are usually (but not always) *dot files*. These are files whose names begin with dots (`.`). Most Linux tools ignore such files unless instructed explicitly not to do so. (For instance, typing **`ls`** alone won't show these files, but **`ls -a`** will.) The result is that dot files are hidden, much as are files on a Windows system with the hidden bit set. This feature makes dot files unobtrusive, which is ideal for configuration files that users don't want appearing in every directory listing.

Most user accounts start with a default set of configuration files. The most important of these include the following:

`.bashrc` This file is a script that's run whenever a bash shell runs. Because bash provides the most common Linux command prompt, this file controls many aspects of a text-based login, such as default environment variables (described later in this chapter, in "Setting Environment Variables").

`.kderc` and `.kde` The `.kderc` file sets many global parameters for the K Desktop Environment (KDE). The `.kde` directory holds configuration files for specific KDE programs.

`.gnome` The `.gnome` directory is the GNU Network Object Model Environment (GNOME) equivalent of the `.kde` directory; it holds configuration files for many specific GNOME component programs.

`.netscape` This directory holds configuration files for the Netscape Web browser.

X configuration files Depending upon how you start X, you'll probably have an X configuration file called `.xinitrc`, `.xsession`, or `.Xclients`. This file, like `.bashrc`, is actually a script. It's often used to launch programs you regularly use.

Window manager files Window managers control window placement and provide decorative borders and program-launch facilities. Most window managers have configuration files or directories named after themselves, such as `.icewm` or `.fvwmrc`.

In addition to these files, you'll probably find many more dot files and directories accumulating in users' directories, particularly if your users run a wide array of programs. Any user program can create a configuration file if it wants to. As a result, you'll see configuration files for mail programs, news readers, word processors, text editors, and more. Most of these are named after the programs that created them, such as `.gedit` for the gEdit editor.

System Configuration Files

Most system configuration files are stored in the `/etc` directory tree. Some files reside directly in this directory, but some hide out in subdirectories. Indeed, a trend in the last few years has been to move multiple configuration files associated with a single program into subdirectories named for the program, such as `/etc/samba` for Samba configuration files. These subdirectory names are not always consistent from one distribution to another, though; for instance, some distributions use `/etc/samba.d` or `/etc/smb` rather than `/etc/samba`.

These configuration files actually fall into several different classes. Some are startup scripts that control the system startup process. Some files control the system once it's booted, and others control specific servers.

System Startup Scripts

System startup scripts control the Linux startup process. This process is not entirely consistent from one distribution to another. As a general rule, Linux systems use *SysV startup scripts* (SysV is short for *System V*, an influential version of Unix from years gone by), but many distributions use variants of this process.

In Unix as a whole, two startup processes are common. One is the SysV method used by Linux. The other is the Berkeley Standard Distribution (BSD) startup process. The BSD process involves running fewer but larger startup scripts than the SysV system. Some early Linux distributions used BSD startup scripts, and the open source BSD OSs (FreeBSD, OpenBSD, and NetBSD) that compete with Linux use them as well. All the major Linux distributions in 2001 use SysV startup scripts.

The SysV startup process begins when the kernel runs the `init` program. This program uses its configuration file, `/etc/inittab`, to begin the system startup process. This file's contents are described in more detail shortly, in the section entitled "`/etc/inittab`." For now, know that one of this file's earlier lines specifies what to do during system initialization (`sysinit` is the keyword used for this). In many Linux systems (such as Red Hat and its closest relatives), the `sysinit` process runs a script called `/etc/rc.d/rc.sysinit`. In Debian and its derivatives, the system runs a script called `/etc/init.d/rcS`.

In either case, this initialization script runs a number of other scripts according to the system's *runlevel*—a number from 0–6 that indicates how the system is to boot. "Setting the Runlevel," later in this chapter, includes more extensive discussion of the runlevel. In most distributions, runlevel 3 corresponds to a normal text-only boot, and runlevel 5 is a normal boot that also automatically starts X. In any event, startup scripts for specific servers and system utilities reside in directories corresponding to particular runlevels. For instance, in Red Hat and most of its derivatives, these files go in `/etc/rc.d/rc?.d`, where *?* is the runlevel number. In Debian and its derivatives, the equivalent directories are `/etc/rc?.d`.

The scripts in each runlevel directory have filenames of the form `S??name` or `K??name`, where *name* is a name that corresponds to a program and *??* is a sequence number. Filenames that begin with `S` represent programs that are to be started, and `K` indicates a program that's to be killed. For instance, a startup directory might contain the following scripts:

```
K35smb
S10network
S80postfix
```

This collection of files means that the `smb` service (Samba) should be stopped, while the `network` and `postfix` services should be started. The scripts are executed in numerical order, so `network` is started before `postfix`.

In reality, these startup scripts are usually symbolic links to the real scripts, which are located in other directories, such as `/etc/init.d` or `/etc/rc.d/init.d`. The scripts in these directories don't have `S` or `K` prefixes or sequence numbers because these are characteristics that are associated with specific runlevels.

The end result of this configuration is that you can control what services run in any given runlevel by moving or renaming links in the appropriate runlevel control directories. For instance, renaming `K35smb` to `S35smb` will start the Samba server rather than kill it. There are also tools that may be used to activate or deactivate servers in particular runlevels, as described later in this chapter.

There are variants on this SysV startup script scheme. For instance, in SuSE Linux, the startup files are controlled through `/etc/rc.config` rather than by their `S` or `K` filename prefixes. Many Linux distributions support a startup script that runs after most or all of the others, in which you may enter any system-specific customizations you might need. This script is usually called `rc.local` or `boot.local`, and it exists in `/etc` or a startup script subdirectory such as `/etc/rc.d`. You can use this script to start servers that don't come with SysV startup scripts, or to set system-specific features such as the keyboard repeat rate (set via the `kbdrate` program).

It's important to remember that these files are scripts. This means that they're plain text files, but they contain commands that are executed by the system as it boots. Most of the SysV startup scripts start or stop just one program, or a closely related cluster of programs, such as the `smbd` and `nmbd` programs that collectively are Samba. Some of these scripts, though, may perform more complex operations. The `network` startup script, for instance, is fairly complex because it must check for the presence of an unknown number of network interfaces and take actions for each of these—actions that may vary from one interface to another.

Files Defining System Functions

Another class of configuration files consists of those that define how the system functions once it's booted. These files usually aren't scripts. Instead, they're used by other programs to control their operation. These files are

located in the /etc directory tree, just like other system-wide Linux configuration files. Some of the more important examples are listed here:

lilo.conf This file controls the configuration of LILO, the most popular Linux boot loader. By editing this file and running the lilo program, you can change what kernel your system boots, add new OSs to your boot menu, or change important boot options. LILO configuration was discussed in Chapter 3, "Software Management."

fstab This file determines what disk partitions and devices Linux mounts automatically, and in what ways. For instance, you can specify that a Windows partition on a dual-boot system be accessible as /windows, /mnt/win, or just about anything else you'd like. The /etc/fstab file isn't restricted to controlling Windows partitions, though; it includes entries for all Linux partitions. Its file format is discussed in more detail shortly, in the section called "/etc/fstab."

X11 The /etc/X11 directory houses several configuration files and subdirectories related to the configuration of X. The most important of these is XF86Config, which defines the screen resolution, color depth, mouse type, and other critical XFree86 details. (Some distributions place this file directly in /etc, though, not in /etc/X11.)

modules.conf or conf.modules These files contain information that helps Linux determine what kernel modules to load, and when. These modules contain drivers that don't reside in the kernel file proper but that may nonetheless be necessary. The format of these files is discussed shortly, in "/etc/modules.conf."

passwd, shadow, group, and gshadow These files contain information on user accounts and groups, including usernames, group names, home directory locations, passwords, and group membership. They're described in Chapter 4.

resolv.conf This file contains information on the locations of name servers, which are computers that translate between numeric IP addresses and hostnames for network interactions.

This is only a sampling of the system function control files. Many of the files in /etc handle comparatively esoteric aspects of system operation or are things you're not likely to need to adjust. Other files exist on some distributions but not others. For instance, some distributions place the computer's

network hostname in a file called /etc/hostname or /etc/HOSTNAME, but others don't do this. In addition, many files in /etc control servers, rather than general system operation.

Files Controlling Important Servers

Servers normally run in the background without human supervision, so they're configured through files stored in /etc. As an administrator, you can adjust a server's configuration file and restart it to change its behavior. Some of the more important server configuration files include these:

inetd.conf or xinetd.conf These files control the network servers that are run through the inetd or xinetd programs, respectively. These programs can look for network requests for several different servers and then launch the appropriate servers only when needed. Doing this saves memory when a server is seldom used. Most distributions ship with either inetd or xinetd, but it's possible to use both. These files are described in more detail in the "Starting and Stopping Services" section later in this chapter.

sendmail.cf This file controls the sendmail mail server, which is used on most Linux distributions. A few distributions use alternative mail servers, such as Exim (controlled through /etc/exim.conf) or Postfix (controlled through files in /etc/postfix).

smb.conf This is the Samba configuration file, which handles file sharing with Windows systems. This file is often located in a subdirectory of /etc, such as /etc/samba or /etc/samba.d.

exports A Network Filesystem (NFS) server handles file sharing with other Linux and Unix systems. Such a server is configured through the /etc/exports file, which contains lists of directories that are to be exported and information on what systems may access these exported directories.

httpd.conf This file controls the Apache Web server. It's normally located in a subdirectory of /etc, such as /etc/httpd/conf.

Many other servers place their configuration files in /etc or a subdirectory thereof. If you've just installed a server and want to configure it, check in /etc for a file or subdirectory with a name that's related to the server you've installed. The name may be related to the server name (as in sendmail.cf) or to the protocol it serves (as in httpd.conf), so check both.

It's possible for server configuration files to be located elsewhere, and this practice is not uncommon for servers that are compiled locally. Such configuration files frequently go in the server's installation directory or in `/usr/local/etc`. If you know where the documentation for the server is located, you can check it—but this documentation sometimes refers to default locations, which may change when a server is compiled as an RPM or Debian package.

Format of Common Configuration Files

Knowing where a configuration file is located is only the start of any attempt to change a system's configuration. To do any good, you must understand something about the format of a configuration file's contents. Many chapters of this book discuss specific configuration file formats. This section covers a few of the more important general-purpose configuration files.

In all cases, you can edit the configuration files with any editor you like. Chapter 7, "Managing Partitions and Processes," includes an introduction to the Vi editor, but you can use another editor, provided it's installed and working.

Don't edit any configuration file unless you understand its function and the file format. Changing some files, including those discussed here, can cause your system to become unbootable if you make a mistake. In such cases, you may need to boot from an emergency system and restore a backup of the original file or reverse your edits.

In most cases, you can learn more about a configuration file by using the man (short for manual) help system, which displays a man page about a file or command. For instance, typing **`man fstab`** displays information about the `/etc/fstab` configuration file.

Startup Scripts

The *startup scripts* described earlier are just that—scripts. As such, they can be edited like other scripts, as described briefly later in this chapter. As a general rule, it's best not to edit SysV startup scripts unless you have a good idea

of what you're doing—these scripts typically include checks for various conditions and launch the program in a way that the distribution developer knew would work.

One exception to this rule is if you've installed a server that was designed for another distribution. In such a case, the startup script may have been customized for an incompatible startup system. You have three options in this situation:

Edit the provided script. You may be able to tweak the provided startup script to get it to work on your system.

Build a new script. You can often design a new startup script by modeling it after the script used to start up another server. Remember to edit the main script and change all the symbolic links to it in all the runlevel directories.

Start the server in the `rc.local` script. You can start the server by entering an appropriate startup command in the `rc.local` or `boot.local` startup script. Not all distributions support this option, though—in particular, Debian and its derivatives don't include this feature.

Real World Scenario

Startup Methods That Work

In my experience, starting a server you build from source or install from an incompatible distribution is usually handled best by creating an entry in your system's `rc.local` or `boot.local` startup script. Modifying scripts provided for an incompatible distribution seldom works unless you're intimately familiar with your distribution's startup process because these scripts tend to be riddled with distribution-specific assumptions. Building a new script can be a good option, but it often fails because of server-specific features in your model script or because your new server needs options that might not be easily handled in the model script. Local startup scripts, by contrast, are simple, and so they can usually support simple one-line calls to new programs. The problem with this is that this method provides no easy process control mechanisms—you can't have the server automatically stop and start when you switch runlevels. It's also inappropriate if the new server has to run before other servers that are launched through the usual SysV scripts.

/etc/inittab

As described earlier, the /etc/inittab file controls the init process, which is responsible for running startup scripts. This file consists of multiple lines, which may be comments, indicated by a leading pound sign (#); or control lines. (Comments are lines that the program that reads the file ignores.) Control lines have the following format:

id:*runlevel*:*action*:*process*

The meanings of each of these components is as follows:

id This is a 1–4 character identification string for the runlevel process. Sometimes it must have a specific form, such as a number for login terminals. The l*n* strings, where *n* is a number from 0–6, represent the action to be taken when switching to a new runlevel.

runlevel The runlevel was briefly described earlier. At any given time, the computer is in one runlevel, which determines what system programs and processes it runs. For most entries, the *runlevel* variable identifies the runlevels at which a process will run, specified without intervening spaces—for instance, 2345 for runlevels 2–5.

action This field determines how init handles the process. The following are some common *action* values:

initdefault This is used to specify the default runlevel.

sysinit This indicates the process that's to be run during system boot. Normally, this process runs the SysV initialization scripts.

respawn The system starts the process whenever it quits. This is used for login processes.

wait The system runs the process and waits for its termination. This is used for running the SysV initialization scripts.

ctrlaltdel This identifies the process that init is to run when somebody presses the Ctrl+Alt+Del key sequence. This normally invokes a reboot process.

process The command to be executed for the given condition.

One of the more common reasons to edit /etc/inittab is to change the system's default runlevel. Doing this is described later in this chapter, in "Setting the Runlevel."

/etc/inittab also contains lines that start various getty programs. A *getty* is a program that displays a text-based login prompt on a text-mode device. Most Linux distributions start six gettys on six virtual consoles, which you can switch to by typing Alt+F*n*, where *n* is the number of the getty. (You must type Ctrl+Alt+F*n* if you're running X.) Linux supports several different types of getty programs, such as the original getty, mgetty, and mingetty. Each supports somewhat different features. For instance, getty and mgetty support logins over serial ports, but mingetty handles only logins from the console (the computer and monitor attached directly to the computer). By adding new getty lines that monitor serial ports, you can support logins from serial terminals or remote logins from modems.

/etc/fstab

The /etc/fstab file controls how Linux provides access to disk partitions and removable media devices. Linux supports a unified directory structure in which every disk device (partition or removable disk) is mounted at a particular point in the directory tree. For instance, you might access a floppy disk at /mnt/floppy. The root of this tree is accessed from /. Directories off of this root may be other partitions or disks, or they may be ordinary directories. For instance, /etc should be on the same partition as /, but many other directories, such as /home, may correspond to separate partitions. /etc/fstab describes how these filesystems are laid out. (fstab is an abbreviation of "filesystem table.")

/etc/fstab consists of a series of lines, each of which contains six fields that are separated by one or more spaces or tabs. A line that begins with a pound sign (#) is a comment, and is ignored. Listing 6.1 shows a sample /etc/fstab file.

Listing 6.1: Sample /etc/fstab File

```
#device       mount point  filesystem options       dump fsck
/dev/hda1     /            ext2       defaults         1 1
/dev/hdb7     /home        reiserfs   defaults         0 2
/dev/hdb5     /windows     vfat       uid=500,umask=0 0 0
/dev/hdc      /mnt/cdrom   iso9660    user,noauto      0 0
/dev/fd0      /mnt/floppy  auto       user,noauto      0 0
server:/home  /other/home  nfs        user,exec        0 0
/dev/hda4     swap         swap       defaults         0 0
```

The meaning of each field in this file is as follows:

Device The first column specifies the mount device. These are usually device filenames that reference hard disks, floppy drives, and so on. For instance, in Listing 6.1, `/dev/hda1` is the first partition on the first EIDE hard disk. It's also possible to list a network drive, as in `server:/home`, which is the `/home` export on the computer called `server`.

Mount point The second column specifies the *mount point*; in the unified Linux filesystem, this is where the partition or disk will be mounted. This should usually be an empty directory in another filesystem. The root (`/`) filesystem is an exception. So is swap space, which is indicated by an entry of `swap`. (The "Adding Swap Space" section in Chapter 8, "Hardware Issues," covers the swap partition in more detail.)

Filesystem type The filesystem type code is the same as the type code used to mount a filesystem with the `mount` command. Common type codes in Linux include `ext2` (for Linux's native ext2 filesystem), `nfs` (for networked NFS exports), `minix` (for the old Minix filesystem, which is sometimes used on floppy disks), `iso9660` (for CD-ROMs), and `vfat` (for Windows partitions and floppies). Listing 6.1 also shows `reiserfs`, which is used for the new Reiser filesystem. A filesystem type code of `auto` lets the kernel auto-detect the filesystem type, which can be a convenient option for removable media devices. Auto-detection doesn't work with all filesystems, though.

Mount options Most filesystems support several mount options, which modify how the kernel treats the filesystem. You may specify multiple mount options, separated by commas. For instance, `uid=500,umask=0` for `/windows` in Listing 6.1 sets the user ID (owner) of all files to 500, and sets the umask to 0. (Chapter 4 includes a discussion of the meaning of the user ID and umask.) `defaults` specifies that the default values are to be used. `user` lets ordinary users mount and unmount filesystems, which can be useful for removable media. (`owner` has a similar meaning, but it requires that the user own the device file.) `noauto` (which has no relationship to the filesystem type of `auto`) means that the filesystem won't be mounted at boot time. Again, this is useful for removable-media devices. Type **`man mount`** to learn about mount options for specific filesystems.

dump operation The next-to-last field contains a `1` if the `dump` utility should back up a partition, or a `0` if it should not. If you never use the `dump` backup program, this option is essentially meaningless.

fsck order At boot time, Linux uses the `fsck` program to check filesystem integrity. This column specifies the order in which this check occurs. A `0` means that `fsck` should *not* check a filesystem. Higher numbers represent the check order. The root partition should have a value of `1`, and all others that should be checked should have a value of `2`.

If you add a new hard disk or need to repartition the one you've got, you'll probably need to modify `/etc/fstab`. You might also need to edit it to alter some of its options. For instance, setting the user ID or umask on Windows partitions mounted in Linux may be necessary to let ordinary users write to the partition. Chapter 7 covers filesystem handling in greater detail.

/etc/modules.conf

The Linux kernel contains drivers for hardware and filesystems, as well as other features. Some of these drivers and features can be compiled into the main kernel file or as *modules*, which are separate driver files. Linux permits dynamic loading and unloading of modules, which allows you to reduce system memory requirements by keeping unused modules unloaded until they're needed. For instance, you might leave a floppy driver unloaded most of the time. This feature can also be useful when dealing with hardware you can attach and detach while the computer is running, such as USB and PC Card devices.

In order to help manage these drivers, Linux supports a kernel module autoloader. This tool detects when a device is needed and attempts to load the appropriate driver. For this to work, though, a configuration file must keep track of these devices. For instance, there must be some way to link `eth1` (an Ethernet interface) to the `3c501` module, if that's the driver for your card. This is the task of the `/etc/modules.conf` file (which is called `/etc/conf.modules` on some distributions).

It's possible to load kernel modules manually, without using `/etc/modules.conf`. Doing so requires manual intervention, though, and eliminates some options possible with the file.

The section entitled "Managing Kernel Modules" in Chapter 8 covers kernel modules in more detail. The `modules.conf` file contains information about modules, such as modules that Linux should load when a particular type of device is accessed and hardware configuration options like what interrupt request (IRQ) to use with a hardware device.

`modules.conf` supports an unusually complex syntax, including conditional statements (to do things if certain conditions are met), inclusion of secondary files, and more. Mostly, though, lines consist of single commands that operate on module names, allowing you to specify certain operations. For instance, take a look at the following:

```
alias snd-card-0 snd-card-interwave
options snd-card-interwave snd_irq=15
post-install snd-card-interwave alsactl restore
alias eth1 3c501
options 3c501 io=0x300 irq=5
```

This example demonstrates some of the most frequently used `/etc/modules.conf` features:

- The `alias` commands tell the system what modules to load (`snd-card-interwave` or `3c501`) when a particular type of device is needed (`snd-card-0` or `eth1`).
- The `options` line specifies options to be passed to the modules. Options are very specific to each module. For instance, notice that the IRQ is set through the `snd_irq` option for `snd-card-interwave`, but through the `irq` option for `3c501`. You can pass multiple options on one line, as illustrated by the `3c501` example.
- The `post-install` command tells Linux to run a program after loading a module. In the preceding example, the `alsactl restore` command runs after the system loads the `snd-card-interwave` module. There's a similar `pre-install` command for running a command before loading a module.

Fortunately, Linux generally handles loading appropriate modules with little or no configuration of `/etc/modules.conf`. If you have problems with a particular module, you may need to consult its documentation to determine precisely what commands to use in this file. Chapter 8 also contains additional advice on module management.

Like many other configuration files, /etc/modules.conf may include comment lines that begin with a pound sign (#).

/etc/profile

/etc/profile contains environment variables (described shortly, in "Setting Environment Variables") and programs that should be run whenever the bash shell is run. Because this is the default shell on most Linux systems, /etc/profile is run whenever most users of the system log in. Environment variables commonly set in /etc/profile include PATH (a colon-separated list of directories to be searched for programs), PS1 (the default shell prompt), MAIL (the location of the user's incoming mail file), USER (the user's account name), and HOSTNAME (the computer's network hostname). In addition, /etc/profile commonly includes a call to the umask program, which sets the umask. This is a number that represents bits to be removed from file permissions when a user creates a file, as described in Chapter 4. The exact details of what /etc/profile does depend on the distribution and, of course, any changes you make to it.

Setting an environment variable or running a program in /etc/profile does *not* guarantee that this action will go unchanged. User configuration files can set or change environment variables, run umask, or generally undo whatever was done in /etc/profile. /etc/profile exists as a convenient way for you to adjust important features such as the PATH environment variable. For instance, if you add binaries to some unusual location, such as /opt/wp8/wpbin, you might add this directory to PATH, as specified in /etc/profile, in order to add it to all users' PATH environment variables.

For users whose default shell is not bash, you'll need to set default environment variables in some other way. The /etc/csh.cshrc and /etc/csh.login files serve a function similar to that of /etc/profile for users of tcsh and some other shells. The details of the contents of these files differ, though, so you shouldn't simply copy information from one to the other. Again, "Setting Environment Variables" includes information on commands you can add to these files.

Additional Files

Some configuration files use formats that are easy to figure out because they're quite simple. Others are moderately complex but well documented.

Some configuration files, though, are quite cryptic. If you're faced with an unknown configuration file, by all means, examine it in a text editor, but be cautious about modifying it. Many, but not all, configuration files have man pages, so typing **man** followed by the file's name (without the path to the file) may produce help on the file. Other times, you can find help on the configuration file format in the associated program's documentation.

Many of this book's chapters cover the use of specific programs, and this often involves editing configuration files. Some programs are complex enough that entire books have been written about them, and in some cases huge parts of these books are devoted to explaining the package's configuration file formats. This shouldn't intimidate you, however; in most cases, knowing just a few features is enough to get you started configuring a program. Modeling your system after an existing one can also take you far down the path to creating an appropriate configuration.

Setting Environment Variables

People exist in certain environments. These can be office buildings, homes, streets, forests, airplanes, and so on. These environments provide us with, among other things, certain information. For instance, we can tell by reading a street sign that we're at the intersection of State and Main streets, or that there's an old mill a short distance away. Just as we humans live in an environment, so do the programs we run on computers. The features that are salient to people, though, aren't the same as the ones that are important to computer programs. Computer programs must be concerned with issues such as the amount of free disk space or the availability of memory. Linux also provides programs with a set of supplementary information known as *environment variables*. Like street signs, environment variables convey information about the resources available to the program. Therefore, understanding how to set and use environment variables is important for both system administrators and users.

The Role of Environment Variables

Programs query environment variables to learn about the state of the computer as a whole, or what resources are available. These variables contain information such as the location of the user's home directory, the computer's Internet hostname, and the name of the command shell that's in use. Individual programs may also use program-specific environment variables to tell them where their configuration files are located, how to display information, or how to use other program-specific options. As a general rule, though, environment variables provide information that's useful to multiple programs. Program-specific information is more often found in program configuration files.

Where to Set Environment Variables

If you're using the bash shell, you can set an environment variable from a command prompt for a specific login by typing the variable name followed by an equal sign (=) and the variable's value, then typing **export** and the variable name on the next line. For instance, you could type the following:

```
$ NNTPSERVER=news.abigisp.com
$ export NNTPSERVER
```

You can shorten this syntax to a single line by typing **export** at the start of the first line:

```
$ export NNTPSERVER=news.abigisp.com
```

The former syntax is sometimes preferable when setting multiple environment variables because you can type each variable on a line and then use a single `export` command to make them all available. This can make shorter line lengths than you would get if you tried to export multiple variables along with their values on a single line. For instance, you could type the following:

```
$ NNTPSERVER=news.abigisp.com
$ YACLPATH=/usr/src/yacl
$ export NNTPSERVER,YACLPATH
```

This syntax is the same as that used for setting environment variables in `/etc/profile`. This system-wide configuration file is called from a bash shell script, which means it contains commands that could be typed at a command prompt.

When setting environment variables in a shell script such as `/etc/profile`, you should ignore the command prompts (`$`) shown in these examples.

Users of the `tcsh` shell don't use `/etc/profile` for setting environment variables, and in fact, the syntax just described doesn't work for this shell. For `tcsh`, the appropriate command to set an environment variable is `setenv`. It's used much like `export` in its single-line form, but without an equal sign:

```
$ setenv NNTPSERVER news.abigisp.com
```

Instead of using `/etc/profile`, `tcsh` uses the `/etc/csh.cshrc` and `/etc/csh.login` files for its system-wide configuration. Therefore, if your system has both bash and `tcsh` users, you'll need to modify both files, using the appropriate syntax for each file.

The preceding examples assigned values to environment variables. In other contexts, though, the environment variable is preceded by a dollar sign (`$`). You can use this fact to refer to an environment variable when setting another. For instance, in bash, the following command adds `:/opt/bin` to the `PATH` environment variable:

```
$ export PATH=$PATH:/opt/bin
```

This syntax is somewhat more complicated for `tcsh`. In this shell, you must add quotes around the new value and use curly braces around the `PATH` variable reference:

```
$ setenv PATH "${PATH}:/opt/bin"
```

In addition to the system-wide files, individual users may set environment variables by editing their local configuration files: `.bashrc` for bash, and `.tcshrc`, `.cshrc`, or `.login` for `tcsh` (`tcsh` tries each of these files in turn until it finds one that exists).

The Meanings of Common Environment Variables

There are many common environment variables you may encounter on your system. You can find out how environment variables are configured by typing **env**. This command is used to run a program with a changed set of environment variables, but when it is typed alone, it returns all the environment variables that are currently set. Variables you may see in this output include the following:

USER This is your current username. It's a variable that's maintained by the system.

PWD This is the present working directory. This environment variable is maintained by the system. Programs may use it to search for files when you don't provide a complete pathname.

HOSTNAME This is the current TCP/IP hostname of the computer.

PATH This is an unusually important environment variable. It sets the *path* for a session, which is a colon-delimited list of directories in which Linux searches for executable programs when you type a program name. For instance, if PATH is `/bin:/usr/bin` and you type **`ls`**, Linux looks for an executable program called `ls` in `/bin` and `/usr/bin`. If the command you type isn't on the path, Linux responds with a `command not found` error. The PATH variable is typically built up in several configuration files, such as `/etc/profile` and the `.bashrc` file in the user's home directory.

The path often includes the current directory indicator (`.`) so that programs in the current directory can be run. This practice poses a security risk, though, because a miscreant could create a program with the name of some other program (such as `ls`) and trick another user into running it by simply leaving it in a directory the victim frequents. Even the `root` user may be victimized in this way. For this reason, it's best to omit the current directory from the PATH variable, especially for the superuser. If it's really needed for ordinary users, put it at the *end* of the path.

LD_LIBRARY_PATH A few programs use this environment variable to indicate directories in which library files may be found. It works much like PATH.

PS1 This is the default prompt in bash. It generally includes variables of its own, such as `\u` (for the username), `\h` (for the hostname), and `\W` (for the current working directory). This value is frequently set in `/etc/profile`, but it is often overridden by users.

NNTPSERVER Some Usenet news reader programs use this environment variable to specify the name of the news server system. This value might be set in `/etc/profile` or in the user's configuration files.

TERM This variable is a name of the current terminal type. In order to move a text-mode cursor and display text effects for programs like text-mode editors, Linux has to know what commands the terminal supports. The `TERM` environment variable contains this information. It's normally set automatically at login, but in some cases you may need to change it.

DISPLAY This variable identifies the display used by X. It's usually `:0.0`, which means the first (numbered from 0) display on the current computer. When you use X in a networked environment, though, this value may be preceded by the name of the computer at which you're sitting, as in `machine4.threeroomco.com:0.0`. This value is set automatically when you log in, but you may change it if necessary. You can run multiple X sessions on one computer, in which case each one gets a different `DISPLAY` number—for instance, `:0.0` for the first session and `:1.0` for the second.

Any given system is likely to have several other environment variables set, but these are fairly esoteric or relate to specific programs. If a program's documentation says that it needs certain environment variables set, you can set them system-wide in `/etc/profile` or some other suitable file, or you can set them in user configuration files, as you deem appropriate.

Starting and Stopping Services

A typical Linux system can be thought of as a collection of running programs. Even just after the computer has booted, when you see nothing but a login prompt on the screen and haven't touched the keyboard, the computer is running several programs. Some of these programs handle routine local services, such as the text-based login prompts. Others are servers that make the computer available to outside systems. All of these services need to

be started in some way, and Linux provides several different means to accomplish this task.

Linux normally runs any given server using just one of the methods discussed here, and most distributions provide a single default method of launching a server. This is particularly important for the discussion of SysV startup scripts and `xinetd`, because these methods both rely on the presence of configuration files that won't be present if the package maintainer intended that the server be run in some other way.

Starting and Stopping via SysV Scripts

The "Startup Scripts" section earlier in this chapter discussed the SysV startup process in general. In brief, when Linux starts, it enters one of several runlevels. Runlevel 0 shuts down the computer, runlevel 1 configures it to run in a single-user maintenance mode, and runlevel 6 reboots the system. On most Linux systems, runlevel 3 corresponds to a multiuser text-mode boot, and runlevel 5 adds X to the mix (for a GUI login prompt). SuSE uses runlevels 2 and 3 instead of 3 and 5, though, and Slackware uses 3 and 4 for these functions. By default, Debian attempts to start X in all its runlevels. In any event, a couple of runlevels are unused by default. The upcoming section, "Setting the Runlevel," covers temporarily or permanently switching runlevels.

Temporarily Enabling or Disabling a Service

SysV startup scripts reside in particular directories—normally `/etc/rc.d/init.d` or `/etc/init.d`. You may run one of these scripts, followed by an option like `start`, `stop`, or `restart`, to affect the server's run status. (Some startup scripts support additional options, like `status`. Type the script name without any parameters to see a list of its options.) For instance, the following command starts the Samba server on a Mandrake 7.2 system:

```
# /etc/rc.d/init.d/smb start
```

You'll usually see some indication that the server is starting up. If the script responds with a `FAILED` message, it typically means that something about the configuration is incorrect, or the server may already be running.

You should keep a few things in mind when manually starting or stopping a service in this way:

- The name of the startup script is usually related to the package in question, but it's not fully standardized. For instance, some Samba servers call their startup scripts `smb`, but others use `samba`. A few startup scripts perform fairly complex operations and start several programs. For instance, many distributions include a `network` or `networking` script that initializes many network functions.
- SysV startup scripts are designed for specific distributions, and may not work if you install a package on another distribution. For instance, a Red Hat SysV startup script is unlikely to work properly on a SuSE system.
- Startup scripts occasionally appear to work, when in fact the service doesn't operate correctly. You can often find clues to failure in the `/var/log/messages` file (type **`tail /var/log/messages`** to see the last few entries).
- One way to reinitialize a server so it rereads its configuration files is to use the `restart` startup script command. Some startup scripts don't include a `restart` command, though. With these, you may need to manually issue the `stop` command followed by the `start` command when you change configuration options. Some servers provide commands you can issue directly to have them reread their configuration options without explicitly restarting them, though; consult the server's documentation for details.

Temporarily starting or stopping a service is useful when you need to adjust a configuration, or when you first install a server. It's almost always possible to reconfigure a running Linux system without rebooting it by reconfiguring and restarting its services.

Permanently Adding or Removing a Service

If you want to permanently change the mix of services your system runs, you may need to adjust which SysV startup scripts the computer runs. As described earlier, Linux determines which services to run by using the runlevel. In addition to the `/etc/rc.d/init.d` or `/etc/init.d` directory in which the SysV startup scripts reside, Linux systems host several directories that contain symbolic links to these scripts. These directories are typically

named /etc/rc.d/rc*n*.d or /etc/rc*n*.d, where *n* is a runlevel number. For instance, /etc/rc.d/rc3.d is the directory associated with runlevel 3. The links in these directories use filenames of the form K*nnservice* or S*nnservice*, where *nn* is a two-digit number and *service* is the name of a service. When the computer enters a given runlevel, it executes the K* and S* scripts in the associated directory. The system passes the `start` command to the scripts that begin with S, and it sends the `stop` command to the scripts that begin with K. Thus, the key to controlling the starting and stopping of services is in the naming of the files in these SysV script directories—if you rename a script that starts with S so that it starts with K, it will stop running the next time the system enters the affected runlevel.

SuSE Linux uses a file called /etc/rc.config to veto startup via the normal startup scripts. A number of variables in /etc/rc.config correspond to specific servers (such as START_HTTPD for Apache, the HTTP server), and the script gives these variables yes or no values. The normal startup scripts check these values and won't start the server if the corresponding value is no. Thus, to start a server via SysV scripts in SuSE, you must enable the script as described here *and* edit its entry in /etc/rc.config. SuSE's configuration tools, YaST and YaST2, do both automatically.

The numbers that come after the S and K codes control the order in which various services are started and stopped. The system executes these scripts from the lowest-numbered to the highest-numbered. This factor can be quite important. For instance, you'll normally want to start servers like Samba or Apache *after* basic networking is brought up.

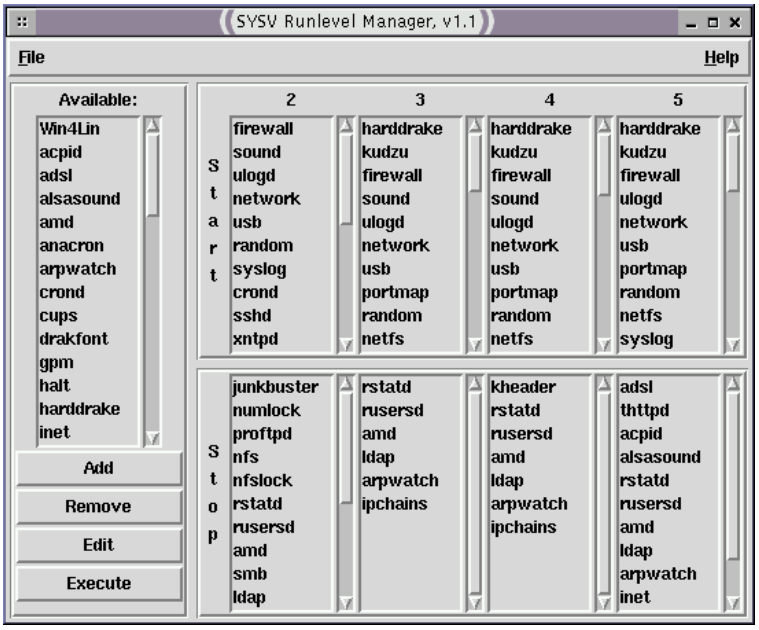
Various tools exist to help you adjust what services run in various runlevels. Not all distributions include all these tools, though. Here are some of the tools for adjusting services:

`chkconfig` This is a strictly command-line utility. Pass it the `--list` parameter to see a summary of services and whether or not they're enabled in each runlevel. You can add or delete a service in a given runlevel by using the `--level` parameter, as in **`chkconfig --level 5 smb on`**, which enables Samba in runlevel 5. (Pass it `off` rather than `on` to disable a service.)

`ntsysv` This is a text-mode utility, but it presents a menu of services run at the runlevel specified with the `--level` parameter. You can enable or disable a service by moving the cursor to the runlevel and pressing the space bar.

`tksysv` Figure 6.1 shows this GUI utility. It allows you to enable or disable services in any runlevel from 2 through 5. Locate and select the service in the given runlevel, then click Remove. This removes its start or stop entry. Then click the service in the Available list and click Add. You'll be prompted to enter a runlevel and whether to start or stop it. You may also need to enter a sequence number.

FIGURE 6.1 The `tksysv` program provides a GUI interface to runlevel service management.



Distribution-specific tools Many distributions' general system administration tools, such as `linuxconf`, YaST, and COAS, provide a means to start and stop SysV services in specific runlevels. Details vary from one distribution to another, so consult your distribution's documentation to learn more.

Once you've modified the SysV startup script listings, that service will run (or not run, if you've disabled it) the next time you restart the computer or change runlevels, as described later, in "Setting the Runlevel." Setting the startup script runlevel information, though, does not change the run status of a service. For that, you'll need to manually enable or disable the service, as described earlier.

One additional method for permanently disabling a service deserves mention: removing it completely from the computer. You can use a package management system (as described in Chapter 3), or you can track down the program's binary files and delete them, to ensure that a service never runs. This is certainly the best way to accomplish the task if the computer never needs to run a program, because it saves on disk space and makes it impossible to misconfigure the computer to run an unwanted server—at least, short of reinstalling the server.

Editing *inetd.conf*

One of the problems with running servers through SysV startup scripts is that the running servers constantly consume memory, even if they're not used much. This is one of the primary motivating factors behind *super servers*, which are servers that listen for network connections intended for any of several other servers. When the super server detects such a connection, it launches the appropriate conventional server. Prior to that time, the conventional server was not running, and so it did not consume any memory. The drawback to this arrangement is that it may take some time for the conventional server to start up, particularly if it's a large one like Samba or Apache. This can result in delays in initiating connections. Nonetheless, this approach is common for many smaller and infrequently used servers. Two super servers are common on Linux: `inetd`, which is described here, and `xinetd`, which is described in the next section.

Be sure you edit the appropriate configuration file! Administrators used to one tool are often confused when they work on a system that uses the other super server. The administrator may edit the wrong configuration file and find that changes have no effect. Ideally, a system won't have a configuration file for an uninstalled super server, but sometimes these do exist, particularly when upgrading a distribution to a new version that changes the super server.

You control servers that launch via `inetd` through the `/etc/inetd.conf` file. This file consists of a series of lines, one for each server. A typical line resembles the following:

```
ftp stream tcp nowait root /usr/sbin/tcpd
↳/usr/sbin/in.ftpd -l -a
```

Each line consists of several fields separated by one or more spaces. The meanings of these fields are listed here:

Service name This is the name of the service as it appears in the `/etc/services` file.

Socket type This may be any of several values, the most common of which are `stream`, `raw`, and `dgram`.

Protocol This is the type of TCP/IP protocol used, usually `tcp` or `udp`.

Wait/Nowait For `datagram` socket types, this entry specifies whether the server connects to its client and frees the socket (`nowait`) or processes all its packets and then times out (`wait`). Servers that use other socket types should specify `nowait` in this field.

User This is the username used to run the server. `root` and `nobody` are common choices, but others are possible, as well.

Server name This is the filename of the server. In the preceding example, the server is specified as `/usr/sbin/tcpd`, which is the TCP Wrappers binary. This program provides some security checks, allowing you to restrict access to a server based on the origin and other factors. Chapter 5, "Networking," discusses TCP Wrappers in more detail.

Parameters Everything after the server name consists of parameters that can be passed to the server. If you use TCP Wrappers, you pass the name of the true target server (such as `/usr/sbin/in.ftpd`) in this field, along with its parameters.

The pound sign (#) is a comment symbol for `/etc/inetd.conf`. Therefore, if a server is running via `inetd` and you want to disable it, you can place a pound sign at the start of the line. If you want to add a server to `inetd.conf`, you'll need to create an entry for it. Most servers that can be run from `inetd` include sample entries in their documentation. Many distributions ship with `inetd.conf` files that include entries for common servers, as well, although many of them are commented out; remove the pound sign at the start of the line to activate the server.

After modifying `inetd.conf`, you must restart the `inetd` super server itself. This super server normally runs as a standard SysV server, so you can restart it by typing something similar to the following:

```
# /etc/rc.d/init.d/inetd restart
```

It's generally wise to disable as many servers as possible in `inetd.conf` (or the `xinetd` configuration files, if you use `xinetd`). As a general rule, if you don't understand what a server does, disable it. This will improve the security of your system by eliminating potentially buggy or misconfigured servers from the equation.

Editing *xinetd.conf* or *xinetd.d* Files

`xinetd` (pronounced "zi-net-dee") is an extended super server. It provides the functionality of `inetd`, plus security options that are similar to those of TCP Wrappers. Mandrake began using `xinetd` with its version 7.1, and Red Hat with its version 7.0. Other distributions may use it in the future. If you like, you can replace `inetd` with `xinetd` on any distribution.

The `/etc/xinetd.conf` file controls `xinetd`. On both Red Hat and Mandrake, though, this file contains only global default options and a directive to include files stored in `/etc/xinetd.d`. Each server that should run via `xinetd` then installs a file in `/etc/xinetd.d` with its own configuration options.

Whether the entry for a service goes in `/etc/xinetd.conf` or a file in `/etc/xinetd.d`, it contains information similar to that in the `inetd.conf` file. `xinetd`, though, spreads the information across multiple lines and labels it more explicitly. Listing 6.2 shows an example that's equivalent to the earlier `inetd.conf` entry. This entry provides precisely the same information as

the `inetd.conf` entry except that it doesn't include a reference to `/usr/sbin/tcpd`, the TCP Wrappers binary. Because `xinetd` includes similar functionality, it's generally not used with TCP Wrappers. Chapter 5 includes information on `xinetd` security features.

Listing 6.2: Sample `xinetd` Configuration Entry

```
service ftp
{
        socket_type     = stream
        protocol        = tcp
        wait            = no
        user            = root
        server          = /usr/sbin/in.ftpd
        server_args     = -l -a
}
```

One additional `xinetd.conf` parameter is important: `disable`. If you include the line `disable = yes` in a service definition, `xinetd` ignores the entry. Some servers install startup files in `/etc/xinetd.d` that have this option set by default; you must edit the file and change the entry to read `disable = no` to enable the server. You can also disable a set of servers by listing their names in the `defaults` section of the main `xinetd.conf` file on a line called `disabled`, as in `disabled = ftp shell`.

As with `inetd`, after you make changes to `xinetd`'s configuration, you must restart the super server. You do this by typing a command similar to the following:

```
# /etc/rc.d/init.d/xinetd restart
```

Custom Startup Files

Occasionally it's desirable to start a server through some means other than a SysV script or super server. This is most frequently the case when you've compiled a server yourself or installed it from a package file intended for a distribution other than the one you're using, and when you don't want to run it through a super server for performance reasons. In such cases, the program may not come with a SysV startup script, or it may not work correctly on your system.

Many Linux distributions include a startup script that runs after the other SysV startup scripts. This script is generally called `/etc/rc.d/rc.local`, `/etc/rc.d/boot.local`, or something similar. You can launch a server from this script by typing the command you would use to launch the server manually, as described in the server's documentation. For instance, you might include the following line to launch an FTP server:

```
/usr/sbin/in.ftpd -l -a
```

Some programs must have an ampersand (&) added to the end of the line to have them execute in the background. If you fail to do this, subsequent lines in the startup script may not run.

One thing to keep in mind when running a server via the custom startup script is that this method provides no means to shut down a server, as you can do by passing the `stop` parameter to a SysV startup script. If you want to stop such a server, you'll need to use the Linux `kill` or `killall` command, possibly after locating the server's process ID number via `ps`. For instance, take a look at the following:

```
# ps ax | grep ftp
 6382 ?        S      0:00 in.ftpd -l -a
# kill 6382
```

`ps` is covered in Chapter 7, "Managing Partitions and Processes." `grep` and the pipe (`|`) are covered in Chapter 9, "Troubleshooting."

Rather than provide a single custom local startup script, Debian and its derivatives provide a directory, `/etc/rc.boot`, in which you can add your own startup scripts. This approach allows you to create separate scripts for each program you want to start up, or you can create a single script to do them all, as you would with other distributions. Call your script whatever you like within `/etc/rc.boot`; the `/etc/init.d/rcS` startup script runs all the scripts in `/etc/rc.boot`, no matter what they're called.

Setting the Runlevel

One way to change the services a system offers en masse is to change the computer's runlevel. As with individual services, you can change the runlevel either temporarily or permanently. Both can be useful. Temporary

changes are useful in testing changes to a system, and permanent changes are useful in implementing changes on a permanent basis.

Understanding the Role of the Runlevel

As described earlier in this chapter, Linux enters a specific runlevel when it boots in order to run some predetermined subset of the programs installed on the computer. For instance, you might want to have two configurations for a computer: one that provides all the computer's usual array of network servers, and another that provides a more limited set, which you use when performing maintenance on the computer. By defining appropriate runlevels and switching between them, you can easily enable or disable a large number of servers.

On most Linux systems, the runlevel also controls whether or not the computer provides a text-mode or GUI login prompt. The latter is the preferable default state for most workstations, but the former is better for many servers or in cases when the X configuration is suspect.

Using *init* or *telinit* to Change the Runlevel

The `init` program is critical to Linux's boot process because it reads the `/etc/inittab` file that controls the boot process and implements the settings found in that file. Among other things, `init` sets the system's initial runlevel.

Once the computer has booted, you can use the `telinit` program to alter the runlevel. (In practice, calling `init` directly also usually works because `telinit` is usually just a symbolic link to `init`.) When using `telinit`, the syntax is as follows:

```
telinit [-t time] runlevel
```

You can discover what runlevel your computer is in with the `runlevel` command. This command displays the previous and current runlevels as output.

In most cases, *`runlevel`* is the runlevel to which you want the system to change. There are, however, a few special codes you can pass as well. Most importantly, `S` or `s` brings the system into a single-user mode; and `Q` or `q` tells the system to reexamine the `/etc/inittab` file and implement any changes in that file.

It's possible to misconfigure X so that it doesn't start. If you do this and your system is set to start X automatically, with some distributions, one consequence is that the system will try to start X, fail, try again, fail, and so on ad infinitum. If the computer has network connections, one way to stop this cycle is to log in remotely and change the runlevel to one that doesn't start X. This will stop the annoying screen flickering that results as X tries to start and fails. You can then correct the problem from the remote login or from the console, test X, and restore the default runlevel.

When switching runlevels, `init` must sometimes kill processes. It does so politely at first by sending a `SIGTERM` signal, which is a way to ask a program to manage its own shutdown. If that doesn't work, though, `init` becomes imperious and sends a `SIGKILL` signal, which is more likely to work but can be more disruptive because the program may leave temporary files lying about and be unable to save changes to open files. The `-t` *`time`* parameter tells `telinit` how long to wait between sending these two signals to a process. The default is five seconds, which is normally plenty of time.

One special case of runlevel change happens when you are shutting down the computer. Runlevel 0 shuts down the computer and halts it—depending upon kernel options and hardware capabilities, this may shut off power to the computer, or it may simply place the computer in a state from which it's safe to turn off system power. Runlevel 6 reboots the computer. You can enter these runlevels using `telinit`, but it's better to use a separate command called `shutdown` to accomplish this task because it offers additional options. The syntax for this command is as follows:

```
shutdown [-t sec] [-arkhcfF] time [warning-message]
```

The meanings of the parameters are as follows:

`-t` *`sec`* This is the delay, in seconds, between `shutdown` telling processes to stop via `SIGTERM` and actually initiating the shutdown process. The default is 5 seconds. This gives programs the chance to shut down cleanly (closing open files, for instance).

`-a` The `/etc/inittab` file contains an invocation of `shutdown` that's called whenever the Ctrl+Alt+Del keystroke is pressed. This allows anybody with physical access to the computer to restart it. If this is undesirable, add the `-a` parameter, and the system will check the `/etc/shutdown.allow` file for a list of users authorized to shut down the

system. Only if one of those users is logged in at the console will `shutdown` proceed.

`-r` This parameter causes a reboot after a shutdown. Essentially, it invokes a change to runlevel 6.

`-k` This parameter "fakes" a shutdown—it sends a shutdown warning message to users, but it doesn't shut down the computer.

`-h` This parameter causes the system to halt after a shutdown. Essentially, it invokes a change to runlevel 0.

`-c` If you initiate a shutdown but then change your mind, issuing `shutdown` again with this parameter cancels it. This is most likely to be useful when the shutdown command was scheduled for some time in the future, rather than immediately.

`-f` This option causes the system to skip its disk check (`fsck`) when it reboots.

`-F` This option forces a disk check (`fsck`) when it reboots.

`time` Shutdowns may be scheduled with this parameter, which can take many different formats. `now` is a common value, which causes an immediate shutdown. You can also specify a time in 24-hour *`hh`*`:`*`mm`* format, as in `13:15` for a shutdown at 1:15 P.M. A *`time`* in the format `+`*`m`* causes a shutdown in *`m`* minutes.

`warning-message` When many people use a system for remote logins, it's generally a good idea to give these users advance warning of a shutdown. You can include a message explaining why the system is going down or how long you expect it to be down.

On a single-user system, **`shutdown -h now`** and **`shutdown -r now`** are perfectly reasonable uses of `shutdown`. When the system has many users, you might be better off scheduling a shutdown for 5, 10, or more minutes in the future and giving information on the expected downtime, as in the following:

```
# shutdown -h +10 "adding new hard disk; up again in 30
↳minutes"
```

A few distributions include commands called `halt` and `reboot` that are equivalent to `shutdown -h now` and `shutdown -r now`, respectively.

Permanently Changing the Runlevel

You can permanently change the computer's runlevel by editing the `/etc/inittab` file. This file contains a line like the following:

```
id:3:initdefault:
```

This example shows a system configured for runlevel 3. To change it, you'd change the 3 to whatever value is appropriate. After making this change, you can cause the system to switch immediately to the new runlevel by running `telinit`, as described earlier. **`telinit Q`** will cause the system to read your changes directly, or you can use the runlevel in place of `Q`.

Do not set the default runlevel to 0 or 6 since this will cause the system to immediately shut down or reboot.

Basic GUI Use

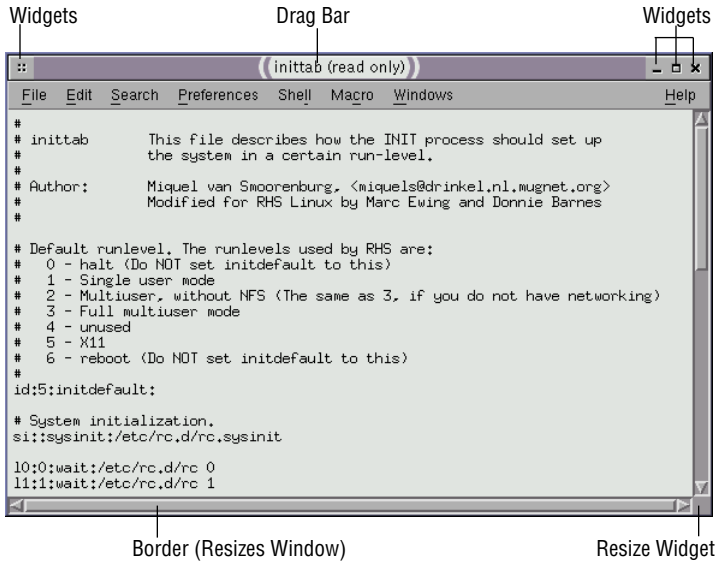
As described in Chapter 2, "Installing Linux," Linux's GUI environment is not a monolithic one as is common on some other OSs. Instead, X is a very modular system, allowing system administrators and even individual users to radically alter the way the system works. Furthermore, different distributions customize their default environments, and offer different GUI administration tools. Therefore, it's difficult to provide detailed information on "the" Linux GUI because one person's GUI environment will be different from the next one's. Nonetheless, there are certain commonalities and general principles, and this section covers these.

Although Linux GUI system administration tools are becoming more common, they tend not to apply across distributions. Red Hat's `linuxconf` is different from SuSE's YaST2, for instance. For this reason, discussions of "generic" Linux tend to emphasize text-based configuration methods.

Features Offered by Window Managers

Window managers provide both decorative and functional features atop the "raw" windows provided by X. Understanding these features, and how they differ between window managers, will help you to navigate a Linux system and work in a variety of window managers, should the need arise. Figure 6.2 shows a typical window as displayed by the IceWM window manager (`http://icewm.sourceforge.net`) using its Helix theme. This configuration is visually similar to most other common Linux window managers, although the details of coloration and patterning in the drag bar at the top of the window often vary substantially, as do the details of widget availability and placement.

FIGURE 6.2 Window manager controls vary in appearance and location, but these are typical.



Window manager controls are frequently referred to as *widgets*. They're commonly found on either or both ends of the drag bar and sometimes elsewhere as well. Features usually offered by a window manager that directly relate to window manipulations include the following:

Window options Most window managers include a drop-down menu accessed by clicking the widget in the upper-left corner. The exact contents of this menu differ from one window manager to another, but typical options include the ability to hide, resize, move, maximize (grow to fill the screen), and close a window. Some of these options, such as the movement options, cause a new control to appear that will allow you to perform the action.

Window movement Most window managers let you move the window about the screen by clicking in the drag bar, keeping the mouse button held down, and moving the mouse. Some also allow you to move a window by selecting an option from the window's menu, as just described.

Resizing You can usually resize a window by clicking a resize widget in the lower-right corner of the window. Many window managers allow resizing in one direction by clicking the border of the window or by selecting a resize option from the main window option menu. Another common resizing option maximizes the window. A few include a "roll up" option, which eliminates all of the window but the drag bar. Minimizing the window makes it completely invisible; you must select it from a desktop menu to retrieve it. Most window managers provide one-click widget shortcuts for at least some of these options.

Closing Most window managers provide a widget that you can click or double-click to close the window entirely. If you do this to the program's main window, this should cause the program running in the window to terminate.

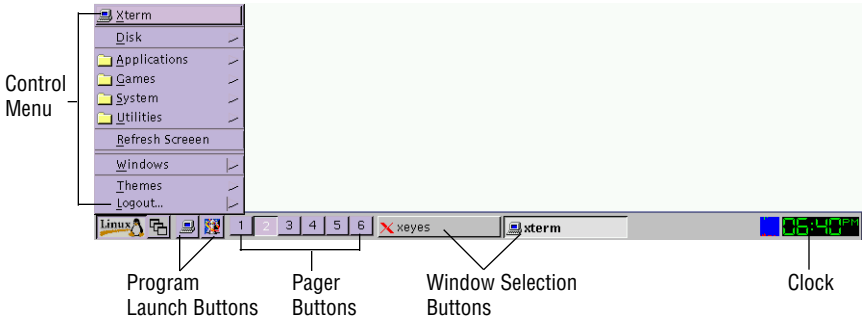
Some of these options may not be available on all windows, even when the window manager supports the feature. For instance, some windows can't be resized. Programmers set the features that a window supports; window managers provide the user interface to let you use these features.

Window managers also offer global features that affect the entire screen. These are illustrated in Figure 6.3. These features include the following:

Control menu Window managers invariably provide some sort of menu that's used to launch programs and control overall window manager function. In some cases (such as IceWM, as illustrated in Figure 6.3), this

control menu can be accessed by clicking a button in a corner of the screen (typically the lower-left corner). In other cases, the menu pops up when you right-click anywhere on the screen that's not occupied by a window or other window manager element. Control menus can usually be customized through window manager configuration files.

FIGURE 6.3 Window managers frequently provide access to global features in a bar on the top or bottom of the screen.



Program launch features As just stated, you can usually launch programs through the control menu. Some window managers provide additional program launch features, such as the program launch buttons of IceWM shown in Figure 6.3.

Pager A pager is a feature in which the window manager maintains multiple virtual desktops, each of which can host its own set of windows. This feature can reduce desktop clutter, but it requires some sort of interface, such as the six pager buttons shown in Figure 6.3. Click on a button to bring up a new desktop.

Window selection When you have many windows open, it's easy for one to completely obscure another. Window selection buttons like those shown in Figure 6.3 allow you to make a window fully visible even if it's completely obscured.

Information displays Many window managers provide clocks, system activity meters, indicators of the presence of new e-mail, and other system information displays.

Desktop environments also provide many of these global features. The desktop environment's features generally override those of the window manager when they're in conflict. It's important to remember that these features vary from one window manager or desktop environment to another.

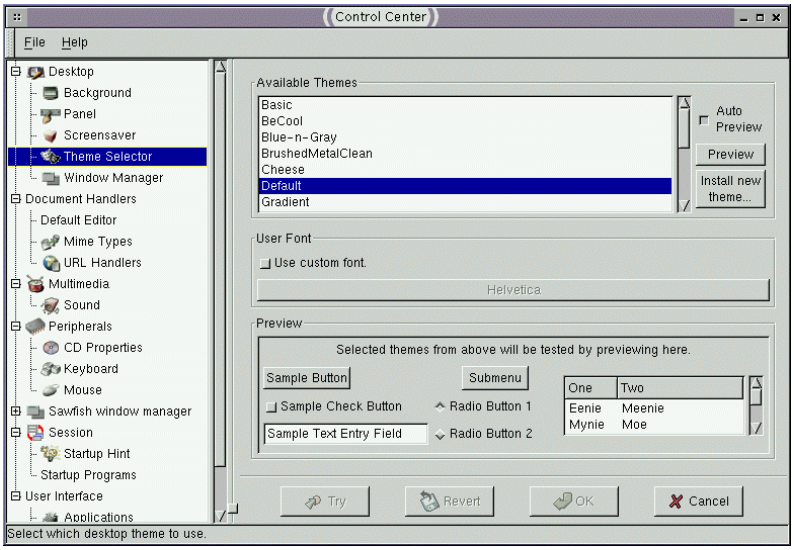
Features Offered by Desktop Environments

One of the roles filled by desktop environments is that of the global features provided by many window managers, as just described. Desktop environments usually have program-launch facilities, pagers, system information displays, and so on. These features can all be used just as in window managers. Desktop environment control menus include ways to launch additional desktop environment tools, some of which may be run automatically when you log in or start X. These tools include the following:

File managers A file manager allows you to copy files using your mouse, launch the programs that created documents, and so on. They're very convenient for users, and they can also be used to administer a Linux system, although many system administrators prefer to use a text-based shell. Many file managers, including those offered by the popular KDE and GNOME packages, provide icons on the desktop that, when clicked or double-clicked, open file manager windows on important directories, such as the user's home directory or the floppy disk mount point. File managers also frequently include "trash cans" to which users may drag files to be deleted. Such files aren't *actually* deleted until some time later, when the trash is "emptied."

Desktop environment controls Traditionally, Unix systems have used disparate programs and controls to adjust many aspects of a user's GUI environment, such as keyboard repeat rate, mouse tracking rate, desktop background images, and so on. Desktop environments typically provide a unified control utility, though, such as GNOME's Control Center, shown in Figure 6.4. From this program, you can adjust many aspects of the GUI environment. Programs designed for the desktop environment will take certain defaults (such as for fonts used in menus) from settings you adjust in the control utility, further unifying the GUI experience. These settings affect only one user's defaults; the desktop environment stores these values in each user's home directory, so one user's preferences won't annoy another user who likes other settings.

FIGURE 6.4 Desktop environment control utilities let you adjust environment features on an individualized basis.



Support utilities Desktop environments include support utilities, such as calculators, text editors, mail programs, audio CD players, and even a few simple games. Although you can use these programs from outside the desktop environment, they are officially part of the environment, and can be easily launched from the environment's menus.

Overall, the best way to learn about a desktop environment's capabilities is to use it. Try investigating the programs available from the desktop environment's main menu. If you're not sure what a program does, check to see if it has a Help button or menu, or simply exit from it until you have time to come back to it.

Unless you're logged in as `root` or provide a `root` password if a program asks for one, you can't do serious damage to the Linux installation as a whole by investigating programs in a desktop environment. There's an outside chance you could render an account unusable, though. If that happens, log in using text mode (by pressing Ctrl+Alt+F1 and logging in) and delete the desktop environment's directory—`.kde` for KDE or `.gnome` for GNOME. The next time the desktop environment starts up, it will restore default settings, so the account should be usable again, although you'll have lost your customizations.

Launching an Xterm

One particularly important X program is known as the *xterm*. This is a program that provides a text-based shell within X's GUI environment. Most window managers and desktop environments provide a very easy way to launch an xterm window. For instance, the left program launch button in Figure 6.3 starts an xterm. The icon for an xterm usually looks like a computer terminal or monitor. Most desktop environments and window managers also include one or more xterm options on their main menus.

In fact, there are several different programs that provide similar functionality. These include the original xterm; the similar `rxvt` program; and the much flashier Konsole and GNOME Terminal of KDE and GNOME, respectively. Throughout this book, I refer to xterm windows, but you can use any xterm or xterm-like program you like. All xterms let you run text-based programs under X. Most include a scroll bar so that you can view old commands and long output listings. The more sophisticated versions include additional features, like menus that allow you to quickly change color schemes and font selections. (In the original xterm, these can be set by options when launching the program. This requires modifying the window manager or desktop environment's xterm-launching command.)

Once an xterm is running, you can use it just like a text-mode login. You'll see a prompt for your shell, and then you can type commands at that prompt. You can also cut-and-paste text between the xterm and other programs by selecting the text and then clicking the middle mouse button in the destination program. (This procedure works between other X-based programs, as well.) One exception to the rule of xterms working like text-mode logins is that you don't log out of an xterm by typing **`logout`**; you type **`exit`** instead. There are also a few programs that can't run in xterms—mainly programs that create graphics displays using low-level calls to the graphics card, like games and some graphics utilities. Many of these programs have X-based equivalents, though, so you may be able to run one of these when in X.

GUI Administrative Tools

Most Linux distributions include some type of GUI administrative tools. These provide point-and-click interfaces to a wide array of configuration options, such as partition mount points, network configuration, network server configuration, runlevels, user administration, and so on. Linux distribution maintainers have not settled on a standard GUI administration

tool. Some of the more common GUI administrative tools include the following:

`linuxconf` Red Hat uses `linuxconf` as its GUI administrative tool, and several distributions derived from Red Hat, such as Mandrake, LinuxPPC, and Yellow Dog, use `linuxconf` as well. The tool operates in X using GUI controls, in text mode or remote logins using text-based menus, or via a Web browser. You can launch `linuxconf` by typing **`linuxconf`** at an xterm or text-mode login, or by selecting it from a window manager or desktop environment menu, if it's on such a menu. To use `linuxconf` from a Web browser, you must first configure `linuxconf` to start in that way, typically by using `linuxconf` itself or by editing the `/etc/xinetd.d/linuxconf-web` file as you would other `xinetd`-based servers. You would then direct a Web browser (potentially running on any OS) to use `linuxconf` by entering the computer's IP address or hostname followed by `:98`, as in `http://norbert.pangaea.edu:98`.

YaST and YaST2 SuSE created Yet another Setup Tool (YaST) as a configuration tool that uses text-based menus. YaST2 is the next-generation, GUI-based configuration tool for SuSE systems. Like `linuxconf`, these tools can be launched from menus on standard SuSE desktop environments, or by typing their names (`yast` or `yast2`) in an xterm or text-mode login.

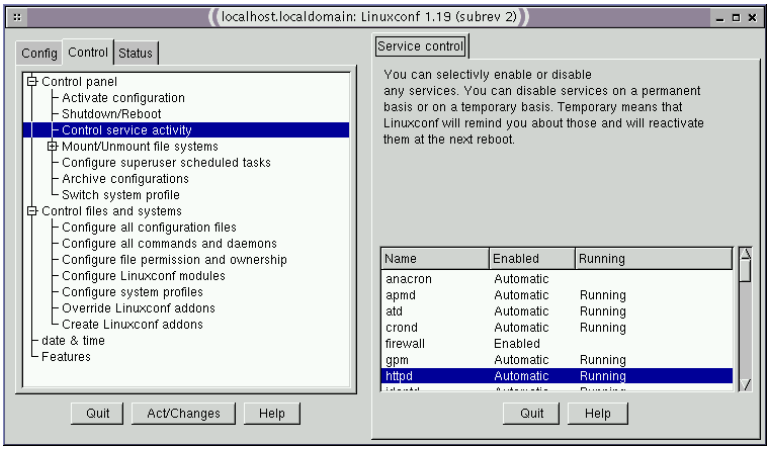
COAS The Caldera Open Administration System (COAS) is Caldera's entry to the GUI administration tool sweepstakes. It can be launched from menus on the standard Caldera desktop, or by typing **`coastool`** at an xterm prompt.

SAS Stormix developed the Storm Administration System (SAS) for its Storm Linux distribution. SAS supports both GUI and text-based menu operation. It can be launched from menus on the standard Storm desktop, or by typing **`sas`** at a command prompt.

Typically, GUI administrative tools organize administrative tasks into groups, such as networking, user administration, filesystems, and so on. Each of these groups has subgroups, which may in turn be further subdivided. Eventually, you get to a configuration control for a specific feature, where you can enter necessary configuration information in a window. Figure 6.5 illustrates this organization in Red Hat's `linuxconf`, in which the menu on the left lists the topic areas and the right portion of the window

shows the ultimate data entry area. Some GUI administration tools (including the version of `linuxconf` shipped with Mandrake) use a series of separate windows for each topic area, rather than the menu on the left in Figure 6.5.

FIGURE 6.5 GUI administration tools categorize administration tasks and provide point-and-click interfaces to the underlying utilities.



GUI administrative tools are essentially very specialized editors; they make changes to the underlying text-based configuration files. Therefore, you can do anything with a text editor and a few commands that you can do with a GUI configuration tool.

Basic Shell Scripting

You'll do much of your work on a Linux system by typing commands at a shell prompt. As you use Linux, though, you're likely to find some of these tasks to be quite repetitive. If you need to add a hundred new users to the system, for instance, typing **useradd** a hundred times can be tedious. Fortunately, Linux includes a way to cut through the tedium: *shell scripts*. These are simple programs written in an interpreted computer language

that's embedded in the Linux shell you use to type commands. Most Linux systems use the bash shell by default, so shell scripts are often written in the bash shell scripting language; but the `tcsh` and other shell scripting languages are quite similar. In fact, it's not uncommon to see shell scripts that run in any common Linux shell.

Many Linux startup scripts, including SysV startup scripts, are in fact shell scripts. Therefore, understanding shell scripting is necessary if you want to modify a Linux startup script.

Like any programming task, shell scripting can be quite complex. Consequently, this chapter barely scratches the surface of what can be accomplished through shell scripting. Consult a book on the topic, such as *Learning the Bash Shell* by Cameron Newham and Bill Rosenblatt (O'Reilly, 1998), for more information.

Beginning a Shell Script

Shell scripts are plain text files, so you create them in text editors. A shell script begins with a line that identifies the shell that's used to run it, such as the following:

```
#!/bin/sh
```

The first two characters are a special code that tells the Linux kernel that this is a script and to use the rest of the line as a pathname to the program that's to interpret the script. Shell scripting languages use a pound sign (#) as a comment character, so the script utility itself ignores this line, although the kernel doesn't. On most systems, `/bin/sh` is a symbolic link that points to `/bin/bash`, but it could point to some other shell. Specifying the script as using `/bin/sh` guarantees that an appropriate shell program will be present on any Linux system that runs the script, but if the script uses any features specific to a particular shell, you should specify that shell instead—for instance, use `/bin/bash` or `/bin/tcsh` instead of `/bin/sh`.

When you're done writing the shell script, you should modify it so that it's executable. You do this with the `chmod` command, as described in Chapter 4. Specifically, you use the `+x` option to add execute permissions, probably in

conjunction with `a` to add these permissions for all users. For instance, to make a file called `my-script` executable, you'd issue the following command:

```
$ chmod a+x my-script
```

You'll then be able to execute the script by typing its name, possibly preceded by `./` to tell Linux to search in the current directory for the script. If you fail to do this, you can still run the script by running the shell program followed by the script name (as in **`bash my-script`**); but it's generally better to make the script executable. If the script is one you run regularly, you may want to move it to a location on your path, such as `/usr/local/bin`. When you do that, you won't have to type the complete path or move to the script's directory to execute it; you can just type **`my-script`**.

Using External Commands

One of the most basic features of shell scripts is the ability to run external commands. Almost all the commands you type in a shell prompt are in fact external commands—they're programs located in `/bin`, `/usr/bin`, and other directories on your path. You can run such programs by including their names in the script. You can also specify parameters to such programs in a script. For instance, suppose you want to start a script that launches two xterms and the KMail mail reader program. (Listing 6.3 presents a shell script that accomplishes this goal.) Aside from the first line that identifies it as a script, the script looks just like the commands you might type to accomplish the task manually, aside from one fact: The script lists the complete paths to each program. This is usually not strictly necessary, but listing the complete path ensures that the script will find the programs even if the `PATH` environment variable changes. Also, each program-launch line in Listing 6.3 ends in an ampersand (`&`). This character tells the shell to go on to the next line without waiting for the first to finish. If you omit the ampersands in Listing 6.3, the effect will be that the first xterm will open, but the second won't open until the first is closed. Likewise, KMail won't start until the second xterm is stopped.

Listing 6.3: A Simple Script That Launches Three Programs

```
#!/bin/bash
/usr/X11R6/bin/xterm &
/usr/X11R6/bin/xterm &
/usr/bin/kmail &
```

Although launching several programs from one script can save time in startup scripts and some other situations, scripts are more frequently used to run a series of programs that manipulate data in some way. Such scripts typically do *not* include the ampersands at the ends of the commands, because one command must run after another or may even rely upon output from the first. A comprehensive list of such commands is impossible because you can run any program you can install in Linux as a command—even another script. A few commands that are commonly used in scripts include the following:

Normal file manipulation commands The file manipulation commands discussed in Chapter 7, such as `ls`, `mv`, `cp`, and `rm`, are often used in scripts. You can use these commands to help automate repetitive file maintenance tasks.

`grep` This command locates files that contain specific strings. In its most basic form, you type **`grep`** ***`pattern files`*** to search for *`pattern`* within the specified *`files`*. For instance, **`grep happy *`** searches for the string `happy` within all files in the current directory.

`find` Where `grep` searches for patterns within the contents of files, `find` does so based on filenames, ownership, and similar characteristics. `find` also searches recursively by default, meaning that it searches all of the subdirectories within any specified directory. To use `find`, you pass it one or more directory names and an expression that includes a search criterion. For instance, **`find /home -name "budget*"`** finds all files whose names begin with `budget` in the `/home` directory and its subdirectories.

`cut` This command extracts text from fields in a file. It's frequently used to extract variable information from a file whose contents are highly patterned. To use it, you pass it one or more options that control what it cuts followed by one or more filenames. For instance, users' home directories appear in the sixth colon-delimited field of the `/etc/passwd` file. You could therefore type **`cut -f 6 -d ":" /etc/passwd`** to extract this information.

`sed` This program's name stands for Stream Editor. It provides many of the capabilities of a conventional text editor but via commands that can be typed at a command prompt or entered in a script. For instance, if you want to copy the file `budget-01.txt` to `budget-02.txt` but replace every occurrence of the string `2001` with `2002`, you could type **`sed s/2001/2002/ budget-01.txt > budget-02.txt`**.

echo Sometimes a script must provide a message to the user. `echo` is the tool to accomplish this goal. You can pass various options to echo or just a string to be shown to the user. For instance, `echo "Press the Enter key"` causes a script to display the specified string.

Many of these commands are extremely complex; completely describing them is beyond the scope of this chapter. You can consult these commands' man pages for more information.

Even if you have a full grasp of how to use some key external commands, simply executing commands you might type at a command prompt is of limited utility. Many administrative tasks require you to modify what you type at a command, or even what commands you enter, depending upon information from other commands. For this reason, scripts include additional features to help you make your scripts useful.

Using Variables

Variables can help you expand the utility of scripts. A variable is a placeholder in a script for a value that will be determined when the script runs. Variables' values can be passed as parameters to scripts or generated internally to the scripts.

Variables that are passed to the script are frequently called parameters. They're represented by a dollar sign (`$`) followed by a number from 0 up—`$0` stands for the name of the script, `$1` is the first parameter to the script, `$2` is the second parameter, and so on. To understand how this might be useful, consider the task of adding a user. As described in Chapter 4, creating an account for a new user typically involves running at least two commands—`useradd` and `passwd`. You might also need to run additional site-specific commands, such as commands to create unusual user-owned directories aside from the user's home directory. As an example of how a script with a parameter variable can help in such situations, consider Listing 6.4. This script creates an account and changes the account's password (you'll be prompted to enter the password when you run the script). It creates a directory in the `/shared` directory tree corresponding to the account, and it sets a symbolic link to that directory from the new user's home directory. It also adjusts ownership and permissions in a way that may be useful, depending upon your system's ownership and permissions policies.

Listing 6.4: A Script to Reduce Account Creation Tedium

```
#!/bin/sh
useradd $1
passwd $1
mkdir /shared/$1
chown $1.users /shared/$1
chmod 775 /shared/$1
ln -s /shared/$1 /home/$1/shared
chown $1.users /home/$1/shared
```

If you use Listing 6.4, you need type only three things: the script name with the desired username, and the password (twice). For instance, if the script is called `mkuser`, you might use it like this:

```
# mkuser ajones
Changing password for user ajones
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

Most of the scripts' programs operate silently unless they encounter problems, so the interaction (including typing the passwords, which don't echo to the screen) is a result of just the `passwd` command. In effect, Listing 6.4's script replaces seven lines of commands with one. Every one of those lines uses the username, so by using this script, you also reduce the chance of an error.

Another type of variable is assigned within the script itself—for instance, it can be set from the output of a command. These variables are also identified by leading dollar signs, but they're typically given names that at least begin with a letter, such as `$ADDR` or `$NAME`. (When assigning values to variables, the dollar sign is omitted, as illustrated shortly.) You can then use these variables in conjunction with normal commands as if they were command parameters, but the value of the variable is passed to the command.

For instance, consider Listing 6.5, which implements simple firewall rules using the `ipchains` utility. This script uses two variables. The first is `$IP`, which is extracted from the output of `ifconfig` using `grep` and `cut` commands. (The trailing backslash on the second line of the script indicates that the following line is a continuation of the previous line.) The pipe operator

(|) used in this assignment is discussed in Chapter 9. When assigning a value to a variable from the output of a command, that command should be enclosed in back-quote characters (`), which appear on the same key as the tilde (~) on most keyboards. These are *not* ordinary single quotes, which appear on the same key as the regular quote character (") on most keyboards. The second variable, `$IPCHAINS`, simply points to the `ipchains` program. It could as easily be omitted, with subsequent uses of `$IPCHAINS` replaced by the full path to the program. Variables like this are sometimes used to make it easier to modify the script in the future. For instance, if you move the `ipchains` program, you need only modify one line of the script. They can also be used in conjunction with conditionals to ensure that the script works on more systems—for instance, if `ipchains` were called something else on some systems.

Listing 6.5: Script Demonstrating Assignment and Use of Variables

```
#!/bin/sh
IP=`ifconfig eth0 | grep inet | cut -f 2 -d ":" | \
    cut -f 1 -d " "`
IPCHAINS="/sbin/ipchains"
echo "Restricting access to $IP"
$IPCHAINS -A input -p tcp -s 0/0 -d $IP 25 -j REJECT
$IPCHAINS -A input -p tcp -s 0/0 -d $IP 80 -j REJECT
```

Listing 6.5 is a poor firewall. It blocks only two ports, and omits many other features useful in a firewall. It is, however, an accessible demonstration of the use of variables in a script.

Scripts like Listing 6.5, which obtain information from running one or more commands, are useful in configuring features that rely on system-specific information or information that varies with time. You might use a similar approach to obtain the current hostname (using the `hostname` command), the current time (using `date`), the total time the computer's been running (using `uptime`), free disk space (using `df`), and so on. When combined with conditional expressions (described shortly), variables become even more powerful because then your script can perform one action when one condition is met, and another in some other case. For instance, a script that installs software could check free disk space and abort the installation if there's not enough disk space available.

Using Conditional Expressions

Scripting languages support several types of *conditional expressions*. These allow a script to perform one of several different actions depending upon some condition—typically the value of a variable. One common command that uses conditional expressions is `if`, which allows the system to take one of two actions depending upon whether or not some condition is true. `if`'s conditional expression appears in brackets after the `if` keyword and can take many forms. For instance, `-f` *`file`* is true if *`file`* exists and is a regular file; `-s` *`file`* is true if *`file`* exists and has a size greater than 0; and *`string1`* `=` *`string2`* is true if the two strings have the same values.

To better understand the use of conditionals, consider the following code fragment:

```
if [ -s /tmp/tempstuff ]
   then
      echo "/tmp/tempstuff found; aborting!"
      exit
fi
```

This fragment causes the script to exit if the file `/tmp/tempstuff` is present. Such code might be useful if the script creates and then later deletes this file, since its presence indicates that a previous run of the script didn't succeed.

Conditional expressions are sometimes used in *loops*, as well. Loops are structures that tell the script to perform the same task repeatedly until some condition is met (or until some condition is no longer met). For instance, Listing 6.6 shows a loop that plays all the `.wav` audio files in a directory.

Listing 6.6: A Script That Executes a Command on Every Matching File in a Directory

```
#!/bin/bash
for d in `ls *.wav` ;
   do play $d ;
done
```

The `for` loop as used here executes once for every item in the list generated by `ls *.wav`. Each of those items (filenames) is assigned in turn to the `$d` variable and so is passed to the `play` command.

Documenting System Configuration

One very important system administration task that's easy to overlook is that of documenting your configuration. Even a lightly used Linux system is likely to collect dozens of changes to configuration files, program installations, and other modifications over the course of its lifetime. It's easy to forget these changes, which can cause problems down the line. For instance, if you alter a system startup script to launch a new server but then replace or upgrade that server, a failure to modify that startup script can cause error messages or result in the updated server not starting. Also, if the system is seriously damaged or if you need to reproduce the system's configuration on another computer, a good set of notes on the first system's configuration can be invaluable.

Maintaining an Administrator's Log

Many administrators keep a written log of all system maintenance. By *written*, I mean just that: recorded in a paper notebook. This format has an advantage in that it's not susceptible to many of the problems that can plague an electronic notebook. For instance, if you keep a log on the computer, that log will most probably be lost if your hard disk dies. A paper notebook is also easily transported to another system, even one without network connectivity, so that you can use your notes to reproduce a configuration on another system.

What should you write in this computer diary? Important information to record includes the following:

Initial configuration Record information on how the computer was configured at system installation. This may include major hardware components, disk partitioning information (including start and end points for each partition—invaluable information should the partition table become corrupt), the Linux distribution and version number, and major installation options (package sets, TCP/IP configuration options, and so on).

Package installations When you install a software package, as described in Chapter 3, record this information. This is particularly important if you compile a package yourself or install it from a tarball since these installation methods leave no record in a package management database, as RPM and Debian package installations do.

Configuration file edits Whenever you edit a configuration file, summarize your changes in the notebook. For small changes, you may want to include precise descriptions of the change—for instance, give the exact environment variable settings you add. For larger changes, you may want to give an overview and leave the details to a backup file.

Filesystem changes Sometimes you must move programs around, or resize your filesystems. When this happens, record what changes you made. Again, when resizing partitions, record the precise sizes of the new partitions.

Kernel recompilations If you recompile or upgrade your Linux kernel, record the details of the changes, including the kernel version number, the major features you added or omitted, and the name of the new kernel.

Hardware changes When adding, deleting, or reconfiguring hardware, make note of those changes. Some of these will also be reflected in configuration file changes. For instance, adding a hard disk will almost certainly entail changing the `/etc/fstab` file.

Correcting earlier entries If you make a change that invalidates information in earlier entries, you may want to track them down and note the change so that you don't accidentally use the wrong information if you ever need it in an emergency.

Ideally, the log book should be stored somewhere that's readily available whenever you administer the computer. A desk drawer next to the computer may work well, for instance. The log won't normally contain sensitive information, but if it does, keep it locked away from prying eyes when it's not in use.

Do not record any passwords in the log book, and *especially* not the `root` password. Only authorized administrators should know the `root` password, and writing it or any other password down is an invitation to disaster. There's no need for any system administrator to know other users' passwords because `root` can do anything to other users' accounts, or even assume other users' identities, as described in Chapter 9.

Backing Up Important Configuration Files

One way to document your system's configuration is to back up important configuration files. The easiest way to do this is to back up the entire `/etc` directory. This can be done with the `tar` command, described more fully in Chapters 3 and 7:

```
# mount /dev/fd0 /mnt/floppy
# tar cvfz /mnt/floppy/etc.tgz /etc
```

These commands create a compressed backup of the entire `/etc` directory's contents on a floppy disk mounted to `/mnt/floppy`. Some distributions, unfortunately, place more data in `/etc` than will fit on a single floppy disk, even with compression, so you may need to use multiple floppies or store the information on a higher-capacity disk like a Zip or LS-120 disk.

Of course, you should perform regular full backups of your computer, which will store all your configuration files along with everything else. Keeping a separate backup of `/etc` is most useful when you've made some extensive change that's causing problems; this way, you can recover a single file from a smallish tarball on disk, which is usually much faster and safer than recovering that file from a tape backup.

Backups of the `/etc` directory tree are *not* a substitute for a written administrator's log. The administrator's log includes information on what files you've altered, which can help lead you directly to a change, rather than fumble around in various files looking for a change. Likewise, a log isn't a substitute for configuration file backups; a log isn't likely to contain the entire contents of all the configuration files, any one of which might be necessary on short notice in an emergency.

The `/etc` directory contains some data that should not be made readily available. In particular, the `/etc/shadow` file (or `/etc/passwd` on systems that don't use shadow passwords) contains encrypted passwords. Although these passwords are encrypted, weak passwords can be extracted via brute-force attacks. Therefore, you should keep your `/etc` directory backups in a secure location.

Summary

There are two basic classes of configuration files: user configuration files and system configuration files. User configuration files are stored in users' home directories, and they allow users to customize the function of various user-level programs. System configuration files reside in the `/etc` directory tree and control various system-wide parameters.

Environment variables can be set on a system-wide basis to control certain aspects of a user's Linux experience, such as the default prompt. Users can adjust their environment variables by typing appropriate commands or by editing their personal startup files.

Starting and stopping specific services and setting the system's runlevel are how you control many aspects of a system's overall "personality"—what network servers the system runs, which set of locally defined services are running by default, and so on. Knowing how to adjust these features allows you to customize your system, and can help you improve security by removing unnecessary servers.

Although many administrative tasks can be done in text mode, many users and even system administrators prefer to use a GUI environment. Linux's GUI environments have matured substantially since the mid-1990s, and today they offer many of the comforts users more familiar with Windows or MacOS have come to expect.

Many system administration tasks involve repetitive actions. For this reason, most administrators learn to write at least basic shell scripts, which can combine many commands in one, frequently using variables and conditional expressions to improve the flexibility of the scripts.

When you edit configuration files or perform other system maintenance, it's important to document your changes. One useful technique for doing this is to keep an administrator's log book. Readily accessible backups of important configuration files, such as the entire `/etc` directory tree, can also help should you need to restore a configuration file to an earlier state.

Exam Essentials

Describe the SysV startup procedure. The `init` process reads the `/etc/inittab` file, which controls programs that run when changing from one runlevel to another. Scripts in directories corresponding to each runlevel start and stop services when the runlevel changes.

Know some of the most critical system configuration files. Important system configuration files include`/etc/inittab`, which directs the boot process; `/etc/fstab`, which determines how and where partitions are mounted; `/etc/modules.conf`, which controls the loading of kernel modules; and `/etc/profile`, which sets defaults for environment variables.

Summarize the purpose of environment variables. Environment variables provide information that should be invariant across programs, such as the user's name and the path to be searched for program files.

Explain the differences between SysV startup scripts and super servers for running servers. SysV startup scripts start servers running on a computer at startup or when changing runlevels so that the servers are always running and can respond quickly to requests, but servers run in this way consume RAM at all times. Super servers run the target servers only in response to requests from clients, thus reducing the memory burden for infrequently used servers but at the cost of slower responses to incoming requests.

Describe the function of the runlevel. Sometimes you may want to run a Linux system with a different set of services than you run at other times. The runlevel lets you define several sets of services and switch quickly between them.

Explain the purpose of GUI administrative tools. Traditional Unix or Linux system administration involves using a large number of text-based commands and editing a number of configuration files. GUI administration tools can ease a new administrator's task by providing a common user interface to these separate commands and files.

Describe how a shell script can be useful. A shell script combines several commands, possibly including conditional expressions, variables, and other programming features to make the script respond dynamically to a system. Therefore, a shell script can reduce administrative effort by performing a series of repetitive tasks at one command.

Commands in This Chapter

Command	Description
`telinit`	Changes the current runlevel.
`man`	Displays help information on a command, configuration file, or other system feature.
`export`	Makes an environment variable available from the bash shell.
`setenv`	Sets an environment variable in `tcsh` and related shells.
`env`	Displays the current environment variables, or temporarily changes them.
`init`	Sets the initial runlevel of the computer.
`telinit`	Changes the runlevel of the computer. (In reality, it's a symbolic link to `init`.)
`shutdown`	Shuts down (halts) or restarts the computer.
`xterm`	Provides a text-mode prompt when in X.

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

conditional expression
dot file
environment variable
`getty`
loop
module
path
runlevel
shell script
startup script

super server
System V
SysV startup script
variable
widget
xterm

Review Questions

1. Which of the following describes most user configuration files?
   A. They control which users may access a server.
   B. Their filenames begin with dots (`.`).
   C. They may not be edited by `root`.
   D. They exist in the `/etc` directory tree.
2. Which of the following configuration files controls where Linux mounts partitions and disks?
   A. `/etc/inittab`
   B. `/etc/fstab`
   C. `/etc/modules.conf`
   D. `/etc/profile`
3. Which of the following is a standard feature of Linux startup scripts?
   A. They are stored in a binary format that must be edited with the `bedit` program.
   B. They are created from meta-configuration scripts using the `m4` utility.
   C. They are shell scripts that may be edited much like other shell scripts.
   D. They are standardized so that they may be transported across distributions.
4. To alter a Linux system's default runlevel, what would you do?
   A. Issue the **`telinit`** ***`x`*** command, where *`x`* is the desired runlevel.
   B. Edit `/etc/modules.conf` and enter the runlevel as an option to the `runlevel` module.
   C. Issue the **`telinit Q`** command to have the system query you for a new runlevel.
   D. Edit `/etc/inittab` and enter the correct runlevel in the `initdefault` line.

5. What is wrong with the following `/etc/fstab` file entry? (Choose all that apply.)

```
/dev/hda8  nfs  default  0 0
```

   A. The entry is missing a mount point specification.

   B. `/etc/fstab` fields should be separated by commas.

   C. The `default` option may only be used with ext2 filesystems.

   D. `/dev/hda8` is a disk partition, but `nfs` indicates a network filesystem.

6. Which of the following will add `/usr/local/bigprog/bin` to the end of the `PATH` environment variable, if placed in `/etc/profile`?

   A. `export PATH=/usr/local/bigprog/bin`

   B. `setenv PATH=$PATH:/usr/local/bigprog/bin`

   C. `export PATH=$PATH:/usr/local/bigprog/bin`

   D. `setenv PATH "${PATH}:/usr/local/bigprog/bin"`

7. Who may set default environment variables for an ordinary user?

   A. Either `root` or the user, with the user's settings taking precedence

   B. Either `root` or the user, with `root`'s settings taking precedence

   C. `root` only

   D. The user only

8. Where is the best location for the current directory indicator (`.`) to reside in `root`'s `PATH` environment variable?

   A. Before all other directories.

   B. After all other directories.

   C. Nowhere; it shouldn't be in `root`'s path.

   D. Wherever is convenient.

9. What types of information should you record in an administrator's log? (Choose all that apply.)

A. The exact contents of all configuration files

B. The locations of major utility programs, like `e2fsck`

C. Major options selected during system installation

D. Descriptions of changes to important configuration files

10. Which of the following methods is the best way to back up the `/etc` directory?

A. Use `tar` to copy the contents to a floppy or other removable disk.

B. Print each file and keep the printed record in a binder or notebook.

C. Copy all the files to another system on the Internet.

D. Use `diff` to record the differences between the current files and their original state.

11. A Linux system keeps its SysV startup scripts in the `/etc/init.d` directory. Which of the following commands will temporarily stop the ProFTPd server on that computer, if it's started from these startup scripts?

A. **`/etc/init.d/proftpd stop`**

B. **`sysvstop /etc/init.d/proftpd`**

C. **`sysvstop proftpd`**

D. **`/etc/init.d/proftpd stop5m`**

12. A new Linux system administrator edits `/etc/inetd.conf` to add a server. After making this change, the administrator tests the new server, but a remote system can't access the new server. Why might this be? (Choose all that apply.)

A. The administrator may have forgotten to restart `inetd`.

B. The system might be using `xinetd` rather than `inetd`.

C. The administrator may have forgotten to edit the `/etc/rc.d/init.d` script for the new server.

D. The administrator may have forgotten to start the new server manually for the first time.

13. You've installed a server by compiling it from source code. The source code included no SysV startup script, and you don't want to run it from a super server, so you start it in a local startup script (`/etc/rc.d/rc.local`). You need to temporarily shut down the server. How might you do this?

    A. Type **`/etc/rc.d/rc.local stop`**.

    B. Edit the startup script to remove the server and rerun the script.

    C. Remove the server's entry from `/etc/inetd.conf` and type **`/etc/rc.d/init.d/inetd restart`**.

    D. Find the server's process ID number (*`pid`*) with `ps` and then type **`kill`** ***`pid`***.

14. Which of the following commands switches a running system into runlevel 3?

    A. **`telnet 3`**

    B. **`runlevel 3`**

    C. **`telinit 3`**

    D. **`switch-runlevel 3`**

15. What does the following command, when typed by a system administrator at noon, accomplish?

    ```
    # shutdown -r 01:00 "Up again soon."
    ```

    A. Reboots the computer at 1:00 P.M. (in 1 hour) and displays the message `Up again soon` as a warning to users

    B. Shuts down (halts) the computer at 1:00 P.M. (in 1 hour) and displays the message `Up again soon` as a warning to users

    C. Shuts down (halts) the computer at 1:00 A.M. (in 13 hours) and displays the message `Up again soon` as a warning to users

    D. Reboots the computer at 1:00 A.M. (in 13 hours) and displays the message `Up again soon` as a warning to users

16. Which of the following are common methods of resizing a window? (Choose all that apply.)

    A. Typing **`enlarge`** ***`win-num`*** at an xterm, where *`win-num`* is the window's X ID number

    B. Clicking the resize widget in the lower-right corner and dragging to a new width and height

    C. Clicking the maximize widget in the drag bar to have the window fill the screen

    D. Selecting a resize option from the window's main menu widget and dragging the resulting tool

17. Once you've started an xterm with default parameters, how can you remove it from your screen? (Choose all that apply.)

    A. By typing **`logout`** in the xterm window

    B. By typing **`exit`** in the xterm window

    C. By clicking a close widget in the window border, if one is present

    D. By dragging the xterm window to the desktop environment's trash icon

18. After using a text editor to create a shell script, what step should you take before trying to use the script?

    A. Set one or more executable bits using `chmod`.

    B. Copy the script to the `/usr/bin/scripts` directory.

    C. Compile the script by typing **`bash`** ***`scriptname`***, where *`scriptname`* is the script's name.

    D. Run a virus checker on the script to be sure it contains no viruses.

19. Describe the effect of the following short script, `cp1`, if it's called as **`cp1 big.c big.cc`**:

```
#!/bin/sh
cp $2 $1
```

A. It has the same effect as the `cp` command, copying the contents of `big.c` to `big.cc`.

B. It compiles the C program `big.c` and calls the result `big.cc`.

C. It copies the contents of `big.cc` to `big.c`, eliminating the old `big.c`.

D. It converts the C program `big.c` into a C++ program called `big.cc`.

20. What is the purpose of conditional expressions in shell scripts?

A. They prevent scripts from executing if license conditions aren't met.

B. They display information on the script's computer environment.

C. They allow the script to take different actions in response to variable data.

D. They allow scripts to learn in a manner reminiscent of Pavlovian conditioning.

Answers to Review Questions

1. B. Most user configuration files are "hidden" dot files that reside in the user's home directory. These seldom have any effect on server operations. `root` may edit *any* normal file on the computer, including user configuration files. System configuration files, not user configuration files, reside in `/etc`.

2. B. `/etc/inittab` controls the boot process. `/etc/modules.conf` contains information on kernel modules. `/etc/profile` sets default environment variables for users of the bash shell.

3. C. Linux controls its startup process through shell scripts, which makes the relevant files easy to edit once you understand shell scripting. These files are not stored in a binary format or created from meta-configuration files. Standardization of startup scripts is poor across distributions, so using a package that requires a startup script from one distribution on another frequently requires you to edit or replace the startup script.

4. D. `/etc/inittab` controls the default runlevel. Although `telinit` can be used to *temporarily* change the runlevel, this change will not be permanent. **`telinit Q`** tells the system to reread `/etc/inittab`, so it could be used to implement a changed default after you've edited the file, but it will have no effect before editing this file. `/etc/modules.conf` has nothing to do with runlevels, and there is no standard `runlevel` module.

5. A, D. A mount directory must be specified between the device entry (`/dev/hda8`) and the filesystem type code (`nfs`). The `nfs` filesystem type code may only be used with an NFS export specification of the form *`server:/export`*, as the device specification. `/etc/fstab` fields are separated by spaces or tabs, not commas (but commas are used between individual options if several options are specified in the options column). The `default` option may be used with *any* filesystem type.

6. C. Option A sets the path to contain *only* the `/usr/local/bigprog/bin` directory, rather than *adding* that directory to the existing path. Options B and D use the `tcsh` syntax for setting the path, and option B uses it incorrectly (`/etc/profile` is used for setting environment variables in bash, not `tcsh`).

7. A. `root` may set environment variables in `/etc/profile` or other system-wide configuration files, and users may set their own environment variables in `.bashrc`, other user-level configuration files, or by typing them in manually. Because the user's settings come later, they override system defaults, if in conflict.

8. C. The current directory indicator is particularly dangerous in `root`'s `PATH` environment variable because it can be used by unscrupulous local users to trick `root` into running programs of the unscrupulous user's design.

9. C, D. The administrator's log should contain information to help you recover a nearly identical system should the need arise, or to help you back out of configuration changes that don't work. Options C and D are useful to one or both of these goals. On the other hand, the exact contents of all configuration files would be far too tedious to enter in a paper log, and the locations of major utility programs are standardized and easy to discover. (If you move a program from its standard location, though, recording this fact may be in order.)

10. A. Floppies are reasonably quick and can usually hold all of the `/etc` directory's contents, when compressed. When floppies are too small, Zip disks or similar media do well. Printouts are impractical when you need to quickly recover an entire file. Some files in `/etc` are sensitive, and so should not be transferred over the Internet. Also, an Internet link could go down at an awkward time, preventing recovery of the data. Although `diff` could produce a compact file of changes, keeping this up-to-date could be difficult, and recovery after changes that were *not* recorded through `diff` could be impossible.

11. A. There is no standard `sysvstop` command, so options B and C can't be correct. Option D uses a parameter (`stop5m`) that's not standard, and so it won't stop the server. Option A stops the server, which can be manually restarted later or which will restart automatically when the system is rebooted, if it's configured to do so.

12. A, B. After editing `/etc/inetd.conf`, `inetd` should be restarted, typically by typing **`/etc/rc.d/init.d/inetd restart`** or something similar. An unused `/etc/inetd.conf` file can sometimes lure administrators used to configuring this file into editing it rather than configuring `xinetd` on systems that run this alternative super server. Running or editing the target server's startup script is unnecessary in this scenario because the server is started from the super server; it's not run directly.

13. D. Killing the server with `kill` will stop it from running. Local startup scripts don't accept `start` and `stop` parameters like those used by SysV startup scripts. Rerunning the startup script, even after editing it to remove references to the target server, won't kill running processes. `inetd` is a super server, and since the server in question isn't being run from a super server, restarting `inetd` won't kill the target server.

14. C. The `telinit` command changes runlevels. `telnet` is Linux's Telnet client for initiating remote logins. `runlevel` displays the current and previous runlevel, but doesn't change the runlevel. There is no `switch-runlevel` command.

15. D. The reboot time, when specified in *`hh`*`:`*`mm`* form, is given as a 24-hour clock time, so `01:00` corresponds to 1:00 A.M. The `-r` parameter specifies a reboot, not a halt. (`-h` specifies a halt.)

16. B, C, D. Most window managers provide many methods of resizing, although some window managers don't offer all of these options. Although some programs provide command-line options to set initial window size, there is no `enlarge` command to resize windows from the command line.

17. B, C. Xterm sessions can be closed by typing **`exit`**, by clicking a close widget maintained by the window manager, or sometimes by selecting a menu option in the xterm. (This last option is most common in advanced xterm-like programs like those provided with GNOME and KDE.) Xterm sessions are not "login sessions," which means that typing **`logout`** won't work. Desktop environment trash icons are used for deleting files, not closing programs.

18. A. Scripts, like binary programs, normally have at least one executable bit set, although they can be run in certain ways without this feature. `/usr/bin/scripts` isn't a standard directory, and scripts can reside in any directory. Scripts are interpreted programs, which means they don't need to be compiled. Typing **`bash`** ***`scriptname`*** will run the script. Viruses are extremely rare in Linux, and because you just created the script, the only ways it could possibly contain a virus would be if your system was already infected or if you wrote it as a virus.

19. C. The `cp` command is the only one called in the script, and that command copies files. Because the script passes the arguments (`$1` and `$2`) to `cp` in reverse order, their effect is reversed—where `cp` copies its first argument to the second name, the `cp1` script copies the second argument to the name of the first. `cp` has nothing to do with compiling C or C++ programs, so neither does the script.

20. C. Conditional expressions return a "true" or "false" response, allowing the script to execute one set of instructions or another, or to terminate or continue a loop.

Managing Partitions and Processes

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ 4.5 Manage and navigate the Linux hierarchy (e.g., `/etc`, `/usr`, `/bin`, `/var`).
- ✓ 4.6 Manage and navigate the standard Linux file system (e.g., `mv`, `mkdir`, `ls`, `rm`).
- ✓ 4.8 Mount and manage filesystems and devices (e.g., `/mnt`, `/dev`, `du`, `df`, `mount`, `umount`).
- ✓ 4.16 Create, edit, and save files using `vi`.
- ✓ 5.1 Create and manage local storage devices and file systems (e.g., `fsck`, `fdisk`, `mkfs`).
- ✓ 5.2 Verify user and `root` cron jobs and understand the function of cron.
- ✓ 5.3 Identify core dumps and remove or forward as appropriate.
- ✓ 5.6 Differentiate core services from non-critical services (e.g., `ps`, PID, PPID, `init`, timer).
- ✓ 5.7 Identify, execute, and kill processes (`ps`, `kill`, `killall`).
- ✓ 5.10 Perform and verify backups and restores.

Most computers' actions are tied very closely to their disk partitions and the files they contain. Web servers must be able to deliver Web files stored on disk, workstations must be able to run applications and store data on disk, and so on. Therefore, it's important that you be able to manage these files and the filesystems that contain them when you work with a Linux computer. Much of this chapter is devoted to this topic, starting with partition management and moving on to the Linux filesystem layout, backups, and commands to manipulate and edit files. This chapter concludes with a look at processes—essentially, running programs. This topic includes how to manage these processes so that they don't run out of control and what to do when a program crashes.

The term filesystem has two meanings. First, it may refer to an organized collection of files, stored in some specific set of directories. For instance, as described shortly, there are certain filesystem standards for Linux that specify in what directories certain types of files reside. Second, "filesystem" may refer to the low-level data structures used to organize files on a hard disk partition or removable disk. There are several different filesystems of this second variety, such as ext2fs, ReiserFS, and FAT. This chapter covers both types of filesystems; which meaning is meant is usually clear from the context. When it isn't, I clarify by using terms such as "directory structure" for the first type or "low-level filesystems" for the second variety.

Partition Management and Maintenance

Linux systems store their data on disk partitions. These are contiguous sections of a hard disk that hold a particular type of data. Most partitions hold filesystems, which in this context means low-level data structures that control the placement of files within the partition. Chapter 2, "Installing Linux," introduced these concepts, but this chapter covers some of the details of managing partitions and the filesystems that they contain.

Using *fdisk* to Create Partitions

Linux's native tool for partition creation is known as `fdisk`, which stands for "fixed disk." This utility is named after a DOS tool, which I refer to in this book in uppercase letters (`FDISK`) to differentiate it from Linux's `fdisk`; although the tools perform similar tasks, they're very different in operation. Most other OSs include their own disk partitioning software, as well.

Linux on non-*x*86 systems may not use a tool called `fdisk`. For instance, PowerPC versions of Linux use a tool called `pdisk`. Some important operational details differ between platforms, so if you're using a non-*x*86 system, consult the documentation for your distribution and its disk-partitioning tool.

Linux's `fdisk` is a text-based tool that requires you to type one-letter commands. You can obtain a list of commands by typing **`?`** or **`m`** at the `fdisk` prompt. The most important `fdisk` commands are as follows:

`d` Delete a partition.

`n` Create a new partition.

`p` Display (print) the partition layout.

`q` Quit without saving changes.

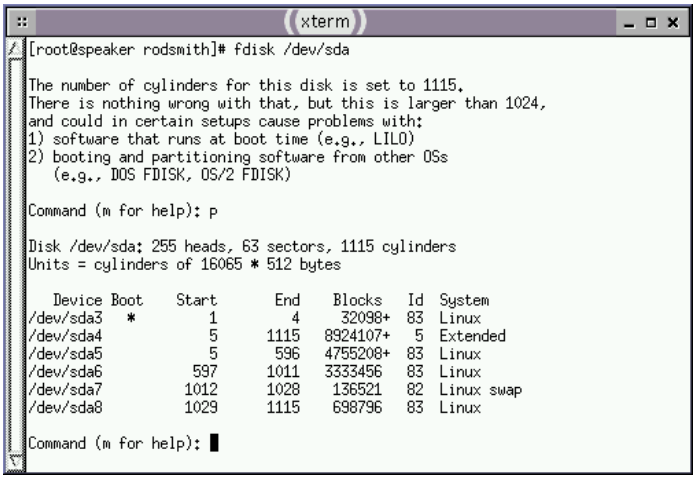
`t` Change a partition's type code.

`w` Write (save) changes and quit.

To start `fdisk`, type its name followed by the Linux device filename associated with your disk device, such as `/dev/sda` or `/dev/hdb`, as in **`fdisk /dev/hdb`**. When you first start `fdisk`, the program displays its prompt. It's

often helpful to type **p** at this prompt to see the current partition layout, as shown in Figure 7.1. This will help you verify that you're operating on the correct disk, if you have more than one hard disk. It will also show you the device IDs of the existing disk partitions.

FIGURE 7.1 As a text-based program, `fdisk` can be run in text mode or in an xterm, as shown here.



The *x*86 partitioning scheme uses three partition types. The main partition table on the disk has room for only four partitions. These partitions are referred to as the primary partitions, and Linux numbers them from 1 to 4, although any of these numbers may be missing. For instance, note that the disk depicted in Figure 7.1 has no partition number 1 or 2; its partitions begin with number 3, as shown in the numbers in the `Device` column. One primary partition may be set aside to be used as a placeholder for additional partitions. This special primary partition is known as an extended partition. `fdisk` identifies these in the `System` column of its output, as shown in Figure 7.1. The partitions that are contained within an extended partition are known as logical partitions, and are numbered from 5 up. Extended partition numbers *always* begin with 5, then go on to 6, and so on. If you add or delete a logical partition, the numbers adjust automatically. For instance, in Figure 7.1, if partition 7 were deleted, partition 8 would become the new partition 7.

You can use the commands outlined above to alter a disk's partition layout, but be aware that your changes are potentially destructive. Deleting partitions will make their data inaccessible. Some commands require you to enter additional information, such as partition numbers or sizes for new partitions. For instance, the following sequence illustrates the commands associated with adding a new partition:

```
Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (519-784, default 519): 519
Last cylinder or +size or +sizeM or +sizeK (519-784,
default 784): +2G
```

You can enter the partition size in terms of cylinder numbers or as a size in bytes, kilobytes, megabytes, or gigabytes (which isn't mentioned in the prompt but does work). When you've made your changes, type **w** to write them to disk and exit. If you make a mistake, type **q** immediately; doing this will exit from `fdisk` without committing changes to disk.

Creating New Filesystems

Just creating partitions isn't enough to make them useful in Linux. To make them useful, you must create a filesystem on the partition. (A task that's also sometimes called "formatting" a partition.) Linux uses the `mkfs` program to accomplish this task. This tool has the following syntax:

```
mkfs [-V] [-t fstype] [options] device [blocks]
```

`mkfs` is actually just a front-end to tools that do the real work for specific filesystems, such as `mke2fs` (also known as `mkfs.ext2`). You can call these tools directly if you prefer, although their syntax may vary from that of `mkfs`.

The meanings of the `mkfs` parameters are as follows:

`-V` This option causes `mkfs` to generate verbose output, displaying additional information on the filesystem-creation process.

`-t fstype` You specify the filesystem type with this option. Common values for *`fstype`* include `ext2` (for ext2fs), `msdos` (for FAT), and `minix` (for Minix).

`options` You can pass filesystem-specific options to the utility. Most underlying filesystem creation tools support `-c` (to perform a low-level disk check to be sure the hardware is sound) and `-v` (to perform a verbose creation).

`device` This is the name of the device on which you want to create the filesystem, such as `/dev/sda5` or `/dev/fd0`. You should *not* normally specify an entire hard disk here (such as `/dev/sda` or `/dev/hdb`). One exception might be if it's a removable disk, but even these are often partitioned.

`blocks` This is the size of the filesystem in blocks (usually 1024 bytes in size). You don't normally need to specify this value, since `mkfs` can determine the filesystem size from the size of the partition.

Depending upon the size and speed of the disk device, the filesystem creation process is likely to take anywhere from a second or less to a minute or two. If you specify a filesystem check (which is often a good idea), this process can take several minutes, or possibly over an hour. Once it's done, you should be able to mount the filesystem and use it to store files.

The filesystem creation process is inherently destructive. If you accidentally create a filesystem in error, it will be impossible to recover files from the old filesystem unless you're very knowledgeable about filesystem data structures, or you can pay somebody with such knowledge. Recovery costs are apt to be very high.

As noted earlier, `mkfs` is just a front-end to other utilities. These are sometimes called directly instead. For instance, the usual method of formatting a ReiserFS partition is to use the `mkreiserfs` utility.

Checking a Filesystem for Errors

Creating partitions and filesystems are tasks you're likely to perform every once in a while—say, when adding a new hard disk or making major changes to an installation. Another task is much more common, though: checking a

filesystem for errors. Bugs, power failures, and mechanical problems can all cause the data structures on a filesystem to become corrupt. The results are sometimes subtle, but if they are left unchecked, they can cause severe data loss. For this reason, Linux includes tools to verify a filesystem's integrity, and to correct any problems that might exist. The main tool you'll use for this purpose is called `fsck`. Like `mkfs`, `fsck` is actually a front-end to other tools, such as `e2fsck` (aka `fsck.ext2`). The syntax for `fsck` is as follows:

```
fsck  [-sACVRTNP] [-t fstype] [--] [fsck-options]
↳filesystems
```

The following list contains the meanings of the more common parameters to this command:

`-A` This option causes `fsck` to check all the filesystems marked to be checked in `/etc/fstab`. This option is normally used in system startup scripts.

`-C` This option displays a text-mode progress indicator of the check process. Most filesystems don't support this feature, but ext2fs does.

`-V` This option produces verbose output of the check process.

`-N` This option tells `fsck` to display what it would normally do, without actually doing it.

`-t` *`fstype`* Normally, `fsck` determines the filesystem type automatically. You can force the type with this flag, though. If used in conjunction with `-A`, this causes the system to check only the specified filesystem types, even if others are marked to be checked. If *`fstype`* is prefixed with `no`, then all filesystems *except* the specified type are checked.

`--` *`fsck-options`* Filesystem check programs for specific filesystems often have their own options. `fsck` passes options it doesn't understand, or those that follow a double dash (`--`), to the underlying check program. Common options include `-a` or `-p` (perform an automatic check), `-r` (perform an interactive check), and `-f` (force a full filesystem check even if it appears to be clean).

`filesystems` This is the name of the filesystem or filesystems being checked, such as `/dev/sda6`.

Normally, you run `fsck` with only the filesystem name, as in **`fsck /dev/sda6`**. You can add options as needed, however. Check the `fsck` man page for less common options.

Run `fsck` *only* on filesystems that are not currently mounted, or that are mounted in read-only mode. Changes written to disk during normal read/write operations can confuse `fsck` and result in filesystem corruption.

Linux runs `fsck` automatically at startup on partitions that are marked for this in `/etc/fstab`, as discussed in Chapter 6, "Managing Files and Services." The normal behavior of `e2fsck` causes it to perform just a quick cursory examination of a partition if it's been unmounted cleanly. The result is that the Linux boot process isn't delayed because of a filesystem check unless the system wasn't shut down properly. There are a couple of exceptions to this rule, though: `e2fsck` forces a check if the disk has gone longer than a certain amount of time without checks (normally six months), or if the filesystem has been mounted more than a certain number of times since the last check (normally 20). Therefore, you will occasionally see automatic filesystem checks of ext2fs partitions even if the system was shut down correctly.

A new generation of filesystems, exemplified by ext3fs, ReiserFS, JFS, and XFS, does away with filesystem checks at system startup even if the system was not shut down correctly. These journaling filesystems keep a log of pending operations on the disk so that in the event of a power failure or system crash, the log can be checked and its operations replayed or undone to keep the filesystem in good shape. This action is automatic when mounting such a filesystem. Nonetheless, these filesystems still require check programs to correct problems introduced by undetected write failures, bugs, hardware problems, and the like. If you encounter odd behavior with a journaling filesystem, you might consider unmounting it and performing a filesystem check—but be sure to read the documentation first. In mid-2001, Linux's journaling filesystems were still largely experimental, and the support utilities (including filesystem check programs) were under active development and were potentially buggy.

Partition Control

One of a system administrator's tasks is to manage disk partitions. "Planning Disk Partitioning" in Chapter 1, "Planning the Implementation," introduced some of the concerns involved in designing an initial partition

layout, but you may need to alter these decisions after installation. You may also need to mount and unmount removable media devices, such as floppies and CD-ROMs.

Identifying Partitions

Linux identifies partitions using device files whose names are based on those for the low-level hardware devices. Specifically, Linux numbers its partitions: 1–4 for primary and extended partitions, and 5 and up for logical partitions within an extended partition. To access a particular partition, append its number to the device filename for a particular hard disk. For instance, if the hard disk is `/dev/sda` (the first SCSI hard disk), you'd use `/dev/sda5` to access the first logical partition on that disk.

Removable media sometimes use partitions, but sometimes don't. You might access a removable SCSI disk through `/dev/sda`, `/dev/sda1`, `/dev/sda4`, or some other partition number. Zip disks come prepartitioned with a single partition: number 4. Magneto-optical discs and CD-ROMs are seldom partitioned.

Of course, you must first know the base name for the disk device. Two types of disk devices are common in Linux: EIDE and SCSI. Each type of device has its own identification rules. For EIDE, the device filenames all begin with `/dev/hd`, and continue with a letter to identify the specific device. EIDE devices can be classified according to two factors: The EIDE interface or chain (each chain has one physical connector on the motherboard or EIDE controller) and whether the drive is configured as a master or slave drive. Most *x*86 motherboards support two chains, and each chain supports one master and one slave drive. Linux labels these starting with `a` for the master on the primary chain, then `b` for the slave on the primary chain, `c` for the master on the secondary chain, and `d` for the slave on the secondary chain. If you add another chain, the letters continue to `e` and onward. Thus, almost all EIDE-based systems will have a `/dev/hda`, and additional drives may take other identifiers. A system might have `/dev/hda` and `/dev/hdc`, but no `/dev/hdb`, for instance. CD-ROM, Zip, and other removable-media devices use the same identification scheme.

SCSI works somewhat differently. With SCSI, the first physical disk is always `/dev/sda`, the second is always `/dev/sdb`, and so on. This is true no matter what the SCSI IDs of the specific drives, or if the drives are used from

the same host adapter. Most removable-media disks (like Zip or magneto-optical disks) use these same identifiers, but CD-ROM drives are an exception. The first SCSI CD-ROM drive is called `/dev/scd0`, the second is `/dev/scd1`, and so on.

Most systems create a link so that you can use `/dev/cdrom` to access your CD-ROM drive, no matter what it's called.

Linux also supports floppy disks, of course. On a one-floppy system, you'll most frequently use `/dev/fd0` as the device filename. Higher numbers (`/dev/fd1` and above) refer to additional floppy drives. There are also device filenames that refer to a floppy of a specific capacity, such as `/dev/fd0H1440` for a 1440KB floppy.

If you don't remember what Linux called your partitions at system installation, you can use the `fdisk` program to find out. Pass it the `-l` parameter (that's a lowercase L, not a number 1) and the name of a disk device (such as `/dev/hdb` or `/dev/sda`) to obtain a listing of the partitions on that disk, thus:

```
# fdisk -l /dev/hdb

Disk /dev/hdb: 255 heads, 63 sectors, 1216 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot  Start   End   Blocks   Id  System
/dev/hdb1         257  1216  7711200    5  Extended
/dev/hdb2           1   192  1542208+  fb  Unknown
/dev/hdb3         193   256   514080   17  Hidden HPFS/
↳NTFS
/dev/hdb5         257   516  2088418+   6  FAT16
/dev/hdb6   *     517   668  1220908+   7  HPFS/NTFS
/dev/hdb7         669  1216  4401778+  83  Linux
```

This output shows the device name associated with the partition, the start and end cylinder numbers, the number of 1024-byte blocks in the partition, the partition's hexadecimal (base 16) ID code, and the partition or OS type associated with that code.

Linux ignores the partition ID code except during installation and to identify extended partitions, but some other OSs use it to determine which partitions they should try to mount. Therefore, it's important that you set any Linux partition's ID code to 0x83. (Linux swap partitions use 0x82.)

Mounting and Unmounting Partitions

Linux provides the `mount` and `umount` commands to mount and unmount partitions, respectively. (Yes, `umount` is spelled correctly; it's missing the first `n`.) In practice, using these commands is usually not too difficult, but they support a large number of options.

Syntax and Parameters for *mount*

The syntax for `mount` is as follows:

```
mount [-alrsvw] [-t fstype] [-o options] [device]
↳[mountpoint]
```

The following is a list of the most common parameters for `mount`:

`-a` This parameter causes `mount` to mount all the files listed in the `/etc/fstab` file, which specifies the most-used partitions and devices. Chapter 6 includes a discussion of this file's format.

`-r` This parameter causes Linux to mount the filesystem read-only, even if it's normally a read/write filesystem.

`-v` As with many commands, `-v` produces verbose output—comments on operations as they occur.

`-w` This parameter causes Linux to attempt to mount the filesystem for both read and write operations. This is the default for most filesystems, but some experimental drivers default to read-only operation.

`-t fstype` Use this parameter to specify the filesystem type (*`fstype`*). Common filesystem types are `ext2` (for ext2fs), `vfat` (for FAT with VFAT long filenames), `msdos` (for FAT using only short DOS filenames), `iso9660` (for CD-ROM filesystems), and `nfs` (for NFS network exports). Linux supports many others, though. If this parameter is omitted, Linux will attempt to auto-detect the filesystem type.

`-o` *`options`* You can add many options using this parameter. Many of these are filesystem-specific.

`device` The *`device`* is the device filename associated with the partition or disk device, such as `/dev/hda4`, `/dev/fd0`, or `/dev/cdrom`. This parameter is usually required, but it may be omitted under some circumstances, as described shortly.

`mountpoint` This is the directory to which the device's contents should be attached. As with *`device`*, it's usually required, but it may be omitted under some circumstances.

The preceding list of `mount` parameters isn't comprehensive; consult the `mount` man page for some of the more obscure options. The most common use of `mount` uses few parameters, because Linux generally does a good job of detecting the filesystem type, and the default parameters work reasonably well. For instance, consider this example:

```
# mount /dev/sdb7 /mnt/shared
```

This command mounts the contents of `/dev/sdb7` on `/mnt/shared`, auto-detecting the filesystem type and using the default options. Ordinarily, only `root` may issue a `mount` command; however, if `/etc/fstab` specifies the `user` or `owner` option, an ordinary user may mount a filesystem using a simplified syntax in which only the device *or* mount point is specified, but not both. For instance, a user might type **`mount /mnt/cdrom`** to mount a CD-ROM drive, if `/etc/fstab` specifies `/mnt/cdrom` as its mount point and uses the `user` or `owner` option.

Many Linux distributions ship with auto-mounter support, which causes the OS to automatically mount removable media when they're inserted. In GUI environments, a file browser may also open on the inserted disk. The user will need to unmount the filesystem by using `umount`, as described shortly, or by selecting an option in the desktop environment.

Options for *mount*

When you do need to use special parameters, it's usually to add filesystem-specific options. Table 7.1 summarizes the most important filesystem options. Some of these are meaningful only in the `/etc/fstab` file.

TABLE 7.1 Important Filesystem Options for the `mount` Command

Option	Supported Filesystems	Description
`defaults`	All	Uses the default options for this filesystem. It's used primarily in the `/etc/fstab` file to ensure that there's an options column in the file.
`loop`	All	Uses the loopback device for this mount. Allows you to mount a file as if it were a disk partition. For instance, **`mount -t vfat -o loop image.img /mnt/image`** mounts the file `image.img` as if it were a disk.
`auto` or `noauto`	All	Mounts or does not mount the filesystem at boot time or when `root` issues the `mount -a` command. Default is `auto`, but `noauto` is appropriate for removable media. Used in `/etc/fstab`.
`user` or `nouser`	All	Allows or disallows ordinary users to mount the filesystem. Default is `nouser`, but `user` is often appropriate for removable media. Used in `/etc/fstab`. When included in this file, `user` allows users to type **`mount`** ***`/mountpoint`***, where *`/mountpoint`* is the assigned mount point, to mount a disk.
`owner`	All	Similar to `user`, except that the user must own the device file. Some distributions, such as Red Hat, assign ownership of some device files (such as `/dev/fd0`, for the floppy disk) to the console user, so this can be a helpful option.
`remount`	All	Changes one or more mount options without explicitly unmounting a partition. To use this option, you issue a `mount` command on an already-mounted filesystem, but with `remount` along with any options you want to change. Can be used to enable or disable write access to a partition, for example.

TABLE 7.1 Important Filesystem Options for the `mount` Command *(continued)*

Option	Supported Filesystems	Description
`ro`	All	Specifies a read-only mount of the filesystem. This is the default for filesystems that include no write access and for some with particularly unreliable write support.
`rw`	All read/write filesystems	Specifies a read/write mount of the filesystem. This is the default for most read/write filesystems.
`uid=`*`value`*	Most filesystems that don't support Unix-style permissions, such as `vfat`, `hpfs`, `ntfs`, and `hfs`	Sets the owner of all files. For instance, `uid=500` sets the owner to whoever has Linux user ID 500. (Check Linux user IDs in the `/etc/passwd` file.)
`gid=`*`value`*	Most filesystems that don't support Unix-style permissions, such as `vfat`, `hpfs`, `ntfs`, and `hfs`	Works like `uid=`*`value`*, but sets the group of all files on the filesystem. You can find group IDs in the `/etc/group` file.
`umask=`*`value`*	Most filesystems that don't support Unix-style permissions, such as `vfat`, `hpfs`, `ntfs`, and `hfs`	Sets the umask for the permissions on files. *`value`* is interpreted in binary as bits to be removed from permissions on files. For instance, `umask=027` yields permissions of 750, or `–rwxr-x---`. Used in conjunction with `uid=`*`value`* and `gid=`*`value`*, this option lets you control who can access files on FAT, HPFS, and many other foreign filesystems.
`conv=`*`code`*	Most filesystems used on Microsoft and Apple OSs: `msdos`, `umsdos`, `vfat`, `hpfs`, `ntfs`, `hfs`	If *`code`* is `b` or `binary`, Linux doesn't modify the files' contents. If *`code`* is `t` or `text`, Linux auto-converts files between Linux-style and DOS- or Macintosh-style end-of-line characters. If *`code`* is `a` or `auto`, Linux applies the conversion unless the file is a known binary file format. It's usually best to leave this at its default value of `binary` because file conversions can cause serious problems for some applications and file types.

TABLE 7.1 Important Filesystem Options for the `mount` Command *(continued)*

Option	Supported Filesystems	Description
`norock`	`iso9660`	Disables Rock Ridge extensions for ISO-9660 CD-ROMs.
`nojoliet`	`iso9660`	Disables Joliet extensions for ISO-9660 CD-ROMs.

Some filesystems support additional options that aren't discussed here. The `mount` man page covers some of these, but you may need to look to the filesystem's documentation for some filesystems and options. This documentation may appear in `/usr/src/linux/Documentation/filesystems` or `/usr/src/linux/fs/`*`fsname`*, where *`fsname`* is the name of the filesystem.

Using *umount*

`umount` is a simpler command than `mount`. The basic `umount` syntax is as follows:

```
umount [-anrv] [-t fstype] [device | mountpoint]
```

Most of these parameters have similar meanings to their meanings in `mount`; but there are some differences that deserve mention:

`-a` Rather than unmount partitions listed in `/etc/fstab`, this option causes the system to attempt to unmount all the partitions listed in `/etc/mtab`. On a normally running system, this operation is likely to succeed only partly because it won't be able to unmount some key filesystems, such as the root partition.

`-r` This option tells `umount` that if it can't unmount a filesystem, it should attempt to remount it in read-only mode.

`-t` ***`fstype`*** This option tells the system to unmount only partitions of the specified type. You can list multiple filesystem types by separating them with commas.

`device` **and** ***`mountpoint`*** You need to specify only the device or only the mount point, not both.

As with `mount`, normal users cannot ordinarily use `umount`. The exception is if the partition or device is listed in `/etc/fstab` and specifies the `user` or `owner` option, in which case normal users can unmount the device. (In the case of `owner`, the user issuing the command must also own the device file, as with `mount`.) This is most useful for removable-media devices.

Be cautious when removing floppy disks. Linux caches accesses to floppies, which means that data may not be written to the disk until some time after a write command. Because of this, it's possible to corrupt a floppy by ejecting the disk, even when the drive isn't active. You must *always* issue a `umount` command before ejecting a floppy disk. This isn't an issue for most non-floppy removable media because Linux can lock their eject mechanisms, preventing this sort of problem. Another way to write the cache to disk is to use the `sync` command, but because this command does *not* fully unmount a filesystem, it's not really a substitute for `umount`.

Using *df*

If you need information on disk space used on an entire partition, the `df` command does the job. This command summarizes total, used, and available disk space. `df` supports many options, the most important of which are listed here:

`-h` or `--human-readable` Normally, `df` provides output in 1024-byte blocks. This option makes it provide listings in labeled units of kilobytes (`k`), megabytes (`M`), or gigabytes (`G`) instead.

`-i` or `--inodes` By default, `df` displays disk space used, but this option causes `df` to display information on the consumption of inodes. These are data structures used on Linux filesystems that hold file information. Some filesystems, such as ext2fs, have a fixed number of inodes when formatted. Others, such as FAT and ReiserFS, don't, so this information is spurious or meaningless with these filesystems.

`-l` or `--local` This option causes `df` to ignore network filesystems.

`-T` or `--print-type` This option causes `df` to display the filesystem type code along with other information.

You can type **df** alone or in combination with options to obtain information on your system's mounted partitions. If you want information on just one partition, you can add either the device on which it resides or any file or directory on the filesystem to restrict df's output to that one partition. In action, df works like this:

```
# df -hT
Filesystem    Type  Size  Used Avail Use% Mounted on
/dev/hda9     ext2  2.0G  1.8G   96M  95% /
/dev/hdb5     vfat  2.0G  1.4G  564M  72% /mnt/windows
speaker:/home  nfs  4.5G  2.2G  2.3G  49% /mnt/speaker/
↳home
/dev/hdb7 reiserfs  4.2G  1.9G  2.3G  45% /home
```

df is extremely useful in discovering how much free space is available on a disk and how well distributed across partitions your files are.

Linux's ext2 filesystem normally reserves about 5 percent of its available space for root. The intent is that if users come close to filling the disk, there'll be enough space for the system administrator to log in and perform basic maintenance to correct problems. If a critical filesystem were to fill completely, root might not be able to log in.

Understanding the Linux Filesystem Hierarchy

Linux's placement of files is derived from thirty years of Unix history. Given that fact, the structure is remarkably simple and coherent, but it's easy for a new administrator to become confused. Some directories seem, on the surface, to fulfill similar or even identical roles, but in fact there are subtle but important differences. This section describes the Linux directory layout standards, presents an overview of what goes where, and explains the use of the du utility, which is useful in learning where your disk space is being used.

The FSSTND and FHS

Although Linux draws heavily on Unix, Unix's long history has led to numerous splits and variants, starting with the Berkeley Standard Distribution (BSD), which was originally a set of patches and extensions to AT&T's original Unix code. As a result of these schisms within the Unix community, early Linux distributions didn't always follow patterns that were identical to each other. The result was a great deal of confusion. This problem was quite severe early in Linux's history, and it threatened to split the Linux community into factions. Various measures were taken to combat this problem, one of which was the development of the *Filesystem Standard (FSSTND)*, which was first released in early 1994. The FSSTND standardized several specific features, including these:

- Standardizing the programs that reside in `/bin` and `/usr/bin`. Differences on this score caused problems when scripts referred to files in one location or the other.
- Specifying that executable files should not reside in `/etc`, as had previously been common.
- Removing changeable files from the `/usr` directory tree, allowing it to be mounted read-only, a useful security measure.

There have been three major versions of FSSTND: 1.0, 1.1, and 1.2. FSSTND began to reign in some of the chaos in the Linux world in 1994. By 1995, however, FSSTND's limitations were themselves becoming apparent. Thus, a new standard was developed—the *Filesystem Hierarchy Standard (FHS)*. This new standard is based on FSSTND but extends it substantially. The FHS was created in conjunction with developers of some non-Linux Unix-like OSs, for instance. For this reason, the FHS is more than a Linux standard; it may be used to define the layout of files on other Unix-like OSs.

One important distinction made by the FHS is the one between *shareable files* and *unshareable files*. Shareable files may be reasonably shared between computers, such as user data files and even program binary files. (Of course, you don't need to share such files, but you *may* do so.) If files are shared, they're normally shared through the NFS server, discussed briefly in Chapter 5, "Networking." Unshareable files contain system-specific information, such as configuration files. For instance, you're not likely to want to share a server's configuration file between computers.

A second important distinction used in the FHS is the one between *static files* and *variable files*. The former don't normally change, except through

direct intervention by the system administrator. Most program executables are good examples of static files. Variable files may be changed by users, automated scripts, servers, or the like. For instance, users' home directories and mail queues are composed of variable files. The FHS tries to isolate each directory into one cell of this 2 × 2 (shareable/unshareable × static/variable) matrix. For instance, `/home` is shareable and variable, `/usr` is shareable and static, and `/etc` is unshareable and static. Some directories are mixed, but in these cases, the FHS tries to specify the status of particular subdirectories. For instance, `/var` is variable, and it contains some shareable and some unshareable subdirectories.

Like the FSSTND, the FHS comes in numbered versions. Version 2.2 was released in May 2001. (Distributions available at the time of this writing followed FHS 2.1, but its major features aren't different from those of the new FHS 2.2.) The URL for FHS's official Web page is `http://www.pathname.com/fhs`.

Important Directories and Their Contents

The FHS defines some directories very precisely, but details for others are left unresolved. For instance, as described in Chapter 4, "Users and Security," users' files normally go in the `/home` directory, but you may have reason to call this something else or to use two separate directories for users' files. Overall, there are several common directories that are defined by the FHS or used by convention, including these:

`/` Every Linux filesystem traces its roots to a single directory, known as `/` (pronounced, and often referred to, as the *root filesystem* or *root directory*). All other directories branch off of this filesystem. Linux doesn't use drive letters; instead, every partition or removable disk is *mounted* at a *mount point* within another partition (`/` or something else). Certain critical subdirectories, such as `/etc` and `/sbin`, must reside on the root partition, but others can optionally be on separate partitions. Do not confuse the root directory with the `/root` directory, discussed shortly.

`/boot` The `/boot` directory contains static and unshareable files related to the initial booting of the computer. Higher-level startup and configuration files reside in another directory, `/etc`. Some systems impose particular limits on `/boot`. For instance, older *x*86 BIOSes and older versions of LILO may require that `/boot` reside below the 1024th cylinder of the hard disk. These requirements sometimes, but not always, necessitate that the `/boot` directory be a separate partition.

/bin This directory contains certain critical executable files, such as `ls`, `cp`, and `mount`. These commands are accessible to all users and constitute the most important commands that ordinary users might issue. You won't normally find commands for big application programs in `/bin` (although the Vi editor, discussed later in this chapter, is located here). `/bin` contains static files. Although in some sense the `/bin` files are shareable, because they're so important to the basic operation of a computer, the directory is almost never shared—any potential clients must have their own local `/bin` directories.

/sbin This directory is very similar to `/bin`, but it contains programs that are normally run only by the system administrator—tools like `fdisk` and `e2fsck`. Therefore, it's static and theoretically shareable, but in practice, it makes no sense to share it.

/lib This directory is similar to `/bin` and `/sbin`, but it contains program libraries, which are made up of code that's shared across many programs and stored in separate files to save disk space and RAM. The `/lib/modules` subdirectory contains kernel modules—drivers that can be loaded and unloaded as required. Like `/bin` and `/sbin`, `/lib` is static and theoretically shareable, although it's not shared in practice.

/usr This directory hosts the bulk of a Linux computer's programs. Its contents are shareable and static, so it can be mounted read-only and may be shared with other Linux systems. For these reasons, many administrators split `/usr` off into a separate partition, although this isn't required. Some subdirectories of `/usr` are similar to their namesakes in the root directory (such as `/usr/bin` and `/usr/lib`), but they contain programs and libraries that aren't absolutely critical to the basic functioning of the computer.

/usr/local This directory contains subdirectories that mirror the organization of `/usr`, such as `/usr/local/bin` and `/usr/local/lib`. `/usr/local` hosts files that a specific system administrator installs locally—for instance, packages that are compiled on the target computer itself. The idea is to have an area that's safe from automatic software upgrades when the OS as a whole is upgraded. Immediately after installing Linux, `/usr/local` should be empty except for some "stub" subdirectories. Some system administrators split this off into its own partition to protect it from OS reinstallation procedures that might erase the parent partition.

/usr/X11R6 This directory houses files related to the X Window System (X for short), Linux's GUI environment. Like /usr/local, this directory contains subdirectories similar to those in /usr itself, like /usr/X11R6/bin and /usr/X11R6/lib.

/opt This directory is similar to /usr/local in many ways, but it is intended for ready-made packages that don't ship with the OS, like commercial word processors or games. Typically, these programs reside in subdirectories in /opt named after themselves, such as /opt/applix. /opt is static and shareable. Some system administrators break it into a separate partition or make it a symbolic link to a subdirectory of /usr/local and make that a separate partition.

/home This directory contains users' data, and it is shareable and variable. /home is considered optional in FHS. In practice, it's a matter of the *name* being optional. For instance, if you add a new disk to support additional users, you might leave the existing /home directory intact and create a new /home2 directory to house the new users. /home often resides on its own partition.

/root This is the home directory for the root user. Because the root account is so critical and system-specific, this directory isn't really shareable, but it is variable.

/var This directory contains transient files of various types—system log files, print spool files, mail and news files, and so on. As such, the directory's contents are variable. Some subdirectories are shareable, but others are not. Many system administrators—particularly on systems that see a lot of activity in /var, like major Usenet news or mail servers—put /var in its own partition.

/tmp Many programs need to create temporary (hence variable) files, and the usual place to do so is in /tmp. Most distributions include routines that clean out this directory periodically, and sometimes wipe the directory clean at bootup. /tmp is seldom shared. Some administrators create a separate /tmp partition to prevent runaway processes from causing problems on the root filesystem when processes create too-large temporary files.

/mnt Linux mounts removable-media devices within its normal directory structure, and /mnt is provided for this purpose. Some distributions create subdirectories within /mnt, such as /mnt/floppy and /mnt/

`cdrom`, to function as mount points. Others use `/mnt` directly or even use separate mount points off of `/`, such as `/floppy` and `/cdrom`. The FHS mentions only `/mnt`; it doesn't specify how it's to be used. Specific media mounted in `/mnt` may be either static or variable. As a general rule, these directories are shareable.

`/media` This directory is an optional addition to the 2.2 version of FHS. It's like `/mnt`, but FHS 2.2 specifies that it should contain subdirectories for specific media types, such as `/media/floppy` and `/media/cdrom`.

`/dev` Because Linux treats most hardware devices as if they were files, the OS must have a location in its filesystem where these *device files* reside. `/dev` is that place. It contains a large number of files that function as hardware interfaces. If a user has sufficient privileges, that user may access the device hardware by reading from and writing to the associated device file.

`/proc` This is an unusual directory because it doesn't correspond to a regular directory or partition. Instead, it's a *virtual filesystem* that's created dynamically by Linux to provide access to certain types of hardware information that's not accessible via `/dev`. For instance, if you type **`cat /proc/cpuinfo`**, the system responds by displaying information on your CPU—its model name, speed, and so on. `/proc` isn't officially part of the FHS, but it exists on all major Linux distributions.

Knowledge of these directories and their purposes is invaluable in properly administering a Linux system. For instance, understanding the purpose of directories like `/bin`, `/sbin`, `/usr/bin`, `/usr/local/bin`, and others will help you when it comes time to install a new program. Placing a program in the wrong location can cause problems at a later date. For example, if you put a binary file in `/bin` when it should go in `/usr/local/bin`, that program may later be overwritten or deleted during a system upgrade, when leaving it intact would have been more appropriate.

Using *du*

Sometimes, you must estimate the amount of space that's being consumed by some set of files or directories. For instance, if you're running out of space in `/usr`, you might want to move some subdirectory to a new partition on another physical hard disk. To match the data moved to the target partition,

though, you must know how much space the files in various partitions consume. You can do this with the `du` command, which displays disk usage information. Its basic syntax is as follows:

```
du [options] [files]
```

The program accepts a large number of options, the most important of which are the following:

`-c` or `--total` This option creates a summary of the total space used. It's most useful when you provide a list of several files or directories rather than just a single directory.

`-h` or `--human-readable` `du` normally provides a number of blocks as output, which are usually 1KB in size. If you use this option, though, the software converts to kilobytes, megabytes, or gigabytes, as appropriate.

`-S` or `--separate-dirs` If you specify this option, `du` doesn't add subdirectory totals into the parent directory. For instance, if you use `du` on a directory that contains two 5KB files and a subdirectory that contains 500KB of files, it reports the 500KB subdirectory as having 500KB, and the main directory as having 5KB. If you omit this option, `du` reports the main directory as holding 505KB.

`-s` or `--summarize` This option causes `du` to omit reporting on the individual subdirectories. Instead, it provides just a total for each main file or directory. This is useful if the directory you've targeted has many subdirectories.

`--max-depth=`*`N`* This option tells the program to show the sizes of directories that are nested no more than *`N`* deep within the specified directories. (`du` counts files that are nested more deeply, but it doesn't report the directory sizes except in summary.) Using this option can constrain the size of a report to a manageable level. `--max-depth=0` is equivalent to `--summarize`.

`-x` or `--one-file-system` Normally, `du` scans the contents of subdirectories even if those subdirectories are separate partitions mounted in the main directory. Using this option causes the program to ignore mounted partitions. For instance, if `/usr/local` is a separate partition and you type **`du -x /usr`**, the report won't include the contents of `/usr/local`.

du supports some additional parameters that affect its reports. Consult the du man page for details. Consider the preceding example of wanting to move some files from /usr to another partition. You want to put roughly 150MB of files on a new partition, removing them from /usr. Therefore, you issue the following command and obtain the following output:

```
$ du /usr -h --max-depth=1
449M    /usr/share
149M    /usr/X11R6
109M    /usr/bin
4.0k    /usr/etc
1.2M    /usr/games
12M     /usr/include
313M    /usr/lib
92M     /usr/local
12M     /usr/sbin
281M    /usr/src
16k     /usr/libexec
260k    /usr/i586-mandrake-linux
3.5M    /usr/i486-linux-libc5
8.3M    /usr/i386-glibc20-linux
2.3M    /usr/doc
200k    /usr/man
1.4G    /usr
```

This result reveals that /usr as a whole holds 1.4GB of data, and the subdirectory that's closest to 150MB in size is /usr/X11R6, at 149MB. This directory is therefore a good choice for moving to the new partition.

Most files consume more disk space than is indicated by their file sizes. This is because Linux allocates disk space in chunks of hundreds or thousands of bytes (typically 1024 bytes or multiples thereof, depending upon the low-level filesystem). The result is wasted disk space at the end of each file, similar to the empty space on the last page of most chapters of a book. du takes this into consideration when it reports disk usage. When you move files to a new partition, though, the new partition might waste more or less space than the old one. The precise amount of change depends on the filesystems and the particular files in question. Therefore, it's best to consider du's output to be a rough estimate, especially when moving files from one low-level filesystem to another.

Backing Up and Restoring a Computer

Many things can go wrong on a computer that might cause it to lose data. Hard disks can fail, you might accidentally enter some extremely destructive command, a cracker might break into your system, or a user might accidentally delete a file, to name just a few possibilities. To protect against such problems, it's important that you maintain good backups of the computer. To do this, you need to select appropriate backup hardware, choose a backup program, and implement backups on a regular schedule. You should also have a plan in place to recover some or all of your data should the need arise.

Common Backup Hardware

Just about any device that can store computer data and read it back can be used as a backup medium. The best backup devices are inexpensive, fast, high in capacity, and reliable. They don't usually need to be *random access* devices, though. Random access devices are capable of quickly accessing any piece of data. Hard disks, floppy disks, and CD-ROMs are all random access devices. These devices contrast with *sequential access* devices, which must read through all intervening data before accessing the sought-after component. Tapes are the most common sequential-access devices. Table 7.2 summarizes critical information about the most common types of backup device. For some, such as tape, there are higher-capacity (and more expensive) devices for network backups.

TABLE 7.2 Vital Statistics for Common Backup Devices

Device	Cost of Drive	Cost of Media	Uncompressed Capacity	Speed	Access Type
Tape	$200–$1500	$1–4/GB	10–60GB	1–15 MBps	Sequential
Hard disks	$100 (for removable mounting kit)	$5/GB	30–70GB	15–30 MBps	Random

TABLE 7.2 Vital Statistics for Common Backup Devices *(continued)*

Device	Cost of Drive	Cost of Media	Uncompressed Capacity	Speed	Access Type
Removable disks	$100–$500	$25–200/GB	40MB–2.5GB	1–12 MBps	Random
Optical	$100–$4000	$1–7/GB	650MB–5.2GB	1–6 MBps	Random

Numbers are approximate as of mid-2001. Prices on all storage media have historically fallen rapidly, and capacities have risen. Costs are likely to be lower, and capacities higher, in the future.

The types of devices that appear in Table 7.2 are those most often used for backing up Linux systems. The pros and cons of using specific devices are as follows:

Tapes Tape drives are probably the most popular choice for backing up entire computers. Their sequential-access nature is a hindrance for some applications, but it isn't a problem for routine backups. The biggest problem with tapes is that they're less reliable than some backup media, although reliability varies substantially from one type of tape to another, and the best are reasonably reliable.

Hard disks It's possible to use hard disks for backup purposes. If your computer is equipped with a kit that allows a drive to be quickly removed from a computer, you can swap hard disks in and out, and move them offsite for storage, if desired. Without such a kit, however, hard drives are susceptible to theft or damage along with the computer they're meant to back up.

Removable disks Removable disks range from 40MB PocketZip drives to Orb, magneto-optical, and other disks that exceed 2GB in capacity. (Although floppies can in theory be used for backup, their limited capacity and slow speed means they aren't practical for anything but backing up small data files.) The high cost per gigabyte and low capacities of these drives makes them suitable for personal backup of data files, but not of entire systems.

Optical Optical media include CD recordable (CD-R), CD rewritable (CD-RW), and various recordable DVD (DVD-R, DVD-RAM, and so on) technologies. Linux doesn't treat these as removable hard disks; you must use special software like `cdrecord` to record to them. They're extremely reliable and therefore well-suited to long-term archival storage (most estimates suggest that CD-Rs, for instance, will last 10–100 years). Some of them are large enough to back up entire systems, especially if the computers are small, but for really large jobs, the higher capacity of tapes is desirable.

As a general rule, the best backup devices for entire computers and networks are tapes. The low cost and high capacity of tapes makes them well suited to performing multiple backups of entire computers. It's sometimes desirable to supplement tape backups with optical backups (typically to 650MB CD-R or CD-RW drives, although recordable DVD media are becoming increasingly affordable and common). CD-R backups are particularly helpful for small client systems, on which an entire installation may fit in 650MB, especially when compression is applied. Because a CD-R can be read in an ordinary CD-ROM drive, it's possible to use a networked backup server to create backups of clients' basic installations and, in an emergency situation, recover the data using an emergency Linux boot floppy, the CD-R, and the computer's ordinary hardware. A tape backup would require dedicated tape hardware on each client, an easily transportable tape drive, or network connections to restore the basic boot system.

If you restrict computers' main installation partitions to about 1–1.3GB, those entire partitions will most likely fit, when compressed, on standard 650MB CD-Rs. This can simplify backup and recovery efforts.

It's generally wise to keep multiple backups and to store some of them away from the computers they're meant to protect. Such off-site storage protects your data in case of fire, vandalism, or other major physical traumas. Keeping several backups makes it more likely you'll be able to recover something, even if it's an older backup, should your most recent backup medium fail.

If you decide to use a tape drive, your choices aren't over. There are several competing tape formats in common use in 2001. These include Travan, which dominates the low end of the spectrum; digital audio tape (DAT),

which is generally considered a step up; digital linear tape (DLT), which is well respected for use on servers and networks; 8mm, which is similar to DAT but has higher capacities; and Advanced Intelligent Tape (AIT), which is a high-end tape medium. Each of these competes at least partially with some of the others. Travan drives tend to be quite inexpensive (typically $200-$500), but the media are pricey. The other formats feature more costly drives ($500-$1500 for a single drive), but the media cost less. Maximum capacities vary, ranging from under 1GB for obsolete forms of Travan to 10GB for top-of-the-line Travan to 60GB for the largest 8mm drives. Overall, Travan is a good solution for low-end workstations; DAT is best used on high-end workstations, small servers, and small networks; and the other formats are all good for high-end workstations, servers, and networks.

Tape drives come in several different interfaces, the most common of which today are SCSI and EIDE (aka ATAPI). Some very old drives used floppy interfaces, but such devices are quite small and slow by today's standards. Some old external drives used parallel-port interfaces, and a few new external drives use USB interfaces. In theory, USB drives may be useable in Linux as if they were SCSI devices.

Tape devices are accessed through device files. For SCSI, appropriate filenames are `/dev/st0` and `/dev/nst0`, or variants with higher numbers if you have more than one tape drive. For EIDE/ATAPI, the filenames are `/dev/ht0` and `/dev/nht0`, or higher numbers if you have more than one tape drive. In each case, the filenames that lack the `n` are *rewinding tape devices*—after every operation, the tape rewinds to the beginning. The filenames that include an `n`, by contrast, are *non-rewinding tape devices*—after an operation, the tape remains at its current position. Rewinding devices are convenient for most backup operations, but non-rewinding devices are useful if you want to store more than one backup on a tape. Precisely how you use each type is discussed later.

Common Backup Programs

Linux supports several backup programs. Some are tools designed to back up individual files, directories, or computers. Others build on these simpler tools to provide network backup facilities. Basic backup programs include `tar` (described in Chapter 3, "Software Management"), `dump`, and `cpio`. ARKEIA (`http://www.arkeia.com`) and BRU (`http://www.estinc.com`) are two commercial backup packages that provide explicit network support

and GUI front ends. AMANDA (`http://www.amanda.org`) is a network-capable scripting package that helps `tar` or `dump` perform a backup of an entire network.

A Survey of Backup Software for Linux

Although it's got its problems, `tar` is generally considered the lowest common denominator backup program. Tapes created with `tar` can be read on non-Linux systems—something that's often not true of `dump` archives, whose format is tied to specific filesystems. For this reason, `dump` must explicitly support whatever filesystem you intend to back up. In mid-2001, `dump` supports Linux's ext2fs, but versions that support the new journaling filesystems, such as ReiserFS and XFS, are not yet available.

On the down side, `tar` has a compression problem: As discussed in Chapter 3, `tar` doesn't compress data itself. To do this, `tar` relies on an external program, such as `gzip` or `bzip2`. These programs compress an entire `tar` archive. The problem with this approach is that if an error occurs while restoring the compressed archive, all the data from that error onward will be lost. This makes compressed `tar` archives a risky format for backup. Fortunately, most tape drives support compression in their hardware, and these use more robust compression algorithms. Therefore, if your tape drive supports compression, you should *not* compress a `tar` backup. Let the tape drive do that job, and if there's a read error at restore, you'll probably lose just one or two files. If your tape drive doesn't include built-in compression features, you should either not compress your backups or use another utility, most of which don't suffer from this problem.

Real World Scenario

Backing Up Using Optical Media

Optical media, such as CD-R, require special backup procedures. Typically, these drives cannot be addressed directly, in the way that tape drives can be. Instead, you use special programs like `cdrecord` to copy files to the media. Normally, `cdrecord` accepts input from a program like `mkisofs`, which creates an ISO-9660 filesystem—the type of filesystem that's most often found on CD-ROMs.

One option for backing up to CD-R is to use `mkisofs` and then `cdrecord` to copy files to the CD-R. If you copy files "raw" in this way, though, you'll lose some information, such as write permission bits. You'll have better luck if you create a `tar` file on disk, much as you would when you back up to tape. You would then use `mkisofs` to place that tarball in an ISO-9660 filesystem, and then you would burn the ISO-9660 image file to CD-R. The result will be a CD-R that you can mount and that will contain a tarball you can read with `tar`.

A somewhat more direct option is to create a tarball and burn it directly to CD-R using `cdrecord`, thus bypassing `mkisofs`. Such a CD-R won't be mountable in the usual way, but you can access the tarball directly by using the CD-ROM device file. On restoration, this works much like a tape restore, except that you specify the CD-ROM device filename (such as `/dev/cdrom`) instead of the tape device filename (such as `/dev/st0`).

Using *tar* to Back Up a Computer

Chapter 3 provides a summary of `tar`'s features. In brief, you combine one `tar` command, such as `--create` (`c`) or `--extract` (`x`), with one or more `tar` qualifiers, such as `--file` (`f`) or `--verbose` (`v`). The command determines the basic action that `tar` takes, such as recording or recovering a backup. The qualifiers modify precisely how `tar` performs these actions, such as specifying a file (such as `/dev/st0`, a SCSI tape device). Chapter 3's Tables 3.7 and 3.8 summarize the most useful of `tar`'s commands and qualifiers, but you can consult the `tar` man page for more obscure options.

To back up a computer, a command like the following will do the job:

```
# tar --create --verbose --one-file-system --same-
↳permissions --file /dev/st0 /home / /boot /var
```

This command lists directories in a particular order. Because tape is a sequential-access medium, the system will restore items in the order in which they were backed up. Therefore, for the fastest partial restores, list the filesystems that you most expect to have to restore first. In this example, `/home` is listed first because users sometimes delete files accidentally. Backing up `/home` first, therefore, results in quicker restoration of such files.

This command can be stated more succinctly by using one-character abbreviations for the commands and qualifiers, thus:

```
# tar cvlpf /dev/st0 /home / /boot /var
```

Most of the `tar` options are unremarkable, but one deserves special comment: `--one-file-system`. This qualifier tells `tar` *not* to back up files on any filesystem that's mounted in the target directory unless it's explicitly listed as a target. Ordinarily, `tar` backs up such directories; for instance, if you were to back up `/usr`, and if `/usr/local` were mounted from another partition, `tar` would back up `/usr/local`. Using `--one-file-system` prevents this from happening. This is important in a backup because some filesystems probably shouldn't be backed up, such as network exports, removable media, and the `/proc` filesystem. `/proc` is a virtual filesystem—it's not tied to a disk partition; instead, it provides an interface to information on the hardware. Backing it up wastes tape, and restoring it can cause problems because the restore can alter system settings. Because of the use of `--one-file-system`, it's necessary to explicitly list the mount points of all the partitions you want to back up, including `/`.

After creating a backup, you may want to use the `tar --diff` (aka `--compare`, or `d`) command to verify the backup you've just written against the files on disk. Alternatively, you can include the `--verify` (`W`) qualifier to have this done automatically. Verifying your backup doesn't guarantee it will be readable when you need it, but it should at least catch major errors caused by severely degraded tapes. On the other hand, the verification will almost certainly return a few spurious errors because of files whose contents have legitimately changed between being written and being compared. This may be true of log files, for instance.

Using *mt* to Control a Tape Drive

In `tar` terminology, each backup is a file. This file is likely to contain many files from the original system, but like an RPM or Debian package file, the `tar` file is a single entity. Sometimes a `tar` file is far smaller than the tape on which it's placed. If you want to store more than one `tar` file on a tape, you can do so by using the non-rewinding tape device filename. For instance, the

following commands accomplish the same goal as the preceding one, but in a somewhat different manner, and with subtly different results:

```
# tar cvlpf /dev/nst0 /home
# tar cvlpf /dev/nst0 /
# tar cvlpf /dev/nst0 /boot
# tar cvlpf /dev/nst0 /var
```

After issuing these commands, the tape will contain four `tar` files, one for each of the four directories. In order to access each file after writing them, you need to use a special utility called `mt`. This program moves forward and backward among tape files and otherwise controls tape features. Its syntax is as follows:

```
mt -f device operation [count] [arguments]
```

`device` is the tape device filename. `mt` supports many operations, including the following:

`fsf` Moves forward *`count`* files.

`bsf` Moves backward *`count`* files.

`eod` or **`seod`** Moves to the end of data on the tape.

`rewind` Rewinds the tape.

`offline` or **`rewoffl`** Rewinds and unloads the tape. (Unloading is meaningless on some drives but ejects the tape on others.)

`retension` Rewinds the tape, winds it to the end, and then rewinds it again. This action improves reliability with some types of tape, particularly if the tape has been sitting unused for several months.

`erase` Erases the tape. (This command usually doesn't actually erase the data; it just marks the tape as being empty.)

`status` Displays information on the tape drive.

`load` Loads a tape into the drive. Unnecessary with many drives.

`compression` Enables or disables compression by passing an argument of `1` or `0`, respectively.

`datcompression` Also enables and disables compression.

The `compression` and `datcompression` operations aren't identical; sometimes a tape drive works with one but not the other.

For instance, suppose you created a backup on a SCSI tape, but now you want to create another backup on the same tape without eliminating the first backup. You could issue the following commands to accomplish this task:

```
# mt -f /dev/nst0 rewind
# mt -f /dev/nst0 fsf 1
# tar cvlpf /dev/nst0 /directory/to/back/up
# mt -f /dev/nst0 offline
```

These commands rewind the tape, space past the first file, create a new backup, and then unload the tape. Such commands are particularly useful when performing incremental backups, as described shortly.

Planning a Backup Schedule

It's important that you back up a computer regularly, but precisely *how* regularly is a matter that varies from one system to another. If a computer's contents almost never change (as might be true of a dedicated router or a workstation whose user files reside on a file server), backups once a month or less might be in order. For critical file servers, once a day is not too often. You'll have to decide for yourself just how frequently your systems require backup. Take into consideration factors such as how often the data change, the importance of the data, the cost of recovering the data without a current backup, and the cost of making a backup. Costs may be measured in money, your own time, users' lost productivity, and perhaps lost sales.

Even the most zealous backup advocate must admit that creating a full backup of a big system on a regular basis can be a tedious chore. A backup can easily take several hours, depending upon backup size and hardware speed. For this reason, most backup packages, including `tar`, support *incremental backups*. You can create these using the `--listed-incremental` *`file`* qualifier to `tar`, as shown in this example:

```
# tar cvplf /dev/st0 --listed-incremental /root/inc /
↳/home
```

This command stores a list of the files that have been backed up (along with identifying information to help `tar` determine when the files have changed) in `/root/inc`. The next time the same command is issued, `tar` will not back up files that have already been backed up; it will only back up new files. Thus, you can create a schedule in which you do a full backup of the entire computer only occasionally—say, once a week or once a month. You'd do this by deleting the increment file and running a backup as usual. On intervening weeks or days, you can perform an incremental backup, in which only new and changed files are backed up. These incremental backups will take comparatively little time.

You can use incremental backups in conjunction with `mt` to store multiple incremental backups on one tape. Typically, you'll have two tapes for a backup set: one for a full backup and one for intervening incremental backups. Suppose you do a full backup on Monday. On Tuesday, you'd insert the incremental tape and perform the first incremental backup. On Wednesday, you'd insert this tape and type **`mt -f /dev/nst0 fsf 1`** to skip past Tuesday's incremental backup, and then perform another incremental backup. On Thursday, you'd type **`mt -f /dev/nst0 fsf 2`**, and so on.

There are a couple of drawbacks to performing incremental backups. One is that they complicate restoration. Suppose you do a full backup on Monday and incremental backups every other day. If a system fails on Friday, you'll need to restore the full backup and several incremental backups. Second, after restoring an incremental backup, your system will contain files that you'd deleted since the full backup. If files have short life spans on a computer, this can result in a lot of "dead" files being restored when the time comes to do so.

Despite these problems, incremental backups can be an extremely useful tool for helping make backups manageable. They can also reduce wear and tear on tapes and tape drives, and they can minimize the time it takes to restore files if you know that the files you need to restore were backed up on an incremental tape.

Preparing for Disaster: Backup Recovery

Creating backups is advisable, but doing this isn't enough. You must also have some way to restore backups in case of disaster. There are two aspects to this task: partial restores and emergency recovery.

Partial restores involve recovering just a few noncritical files. For instance, users might come to you and ask you to restore files from their home directories. You can do so fairly easily by using the `--extract` (`x`) `tar` command, thus:

```
# cd /
# tar xvlpf /dev/st0 home/username/filename
```

This sequence involves changing to the root directory and issuing a relative path to the file or directory that must be restored. This is required because `tar` normally strips away the leading / in files it backs up, so the files are recorded in the archive as relative filenames. If you try to restore a file with an absolute filename, it won't work.

You'll need to know the exact name of the file or directory you want to restore in order to do this. If you don't know the exact filename, you may need to use the `--list` (`t`) command to examine the entire contents of the tape, or at least everything until you see the file you want to restore.

If you use incremental backups, you can use the incremental file list to locate the filename you want to restore.

A much more serious problem is that of recovering a system that's badly damaged. If your hard disk has crashed or your system has been invaded by crackers, you must restore the entire system from scratch, without the benefit of your normal installation. You can take any of several approaches to this problem, including the following:

Distribution's installation disk Most Linux distributions' installation disks have some sort of emergency recovery system. These may come as separate boot floppy images or as options to type during the boot process. In any event, these images are typically small but functional Linux systems with a handful of vital tools, such as `fdisk`, `mkfs`, Vi, and `tar`. Check your distribution's documentation or boot its boot media and study its options to learn more.

Emergency system on removable disk You can create your own emergency system on a removable disk. If you've got a moderately high capacity removable disk, like a Zip or LS-120 disk, you can create a moderately comfortable Linux system on this disk. The ZipSlack distribution (a variant of Slackware, `http://www.slackware.com`) is particularly handy for this purpose because it's designed to fit on a 100MB Zip disk. You can use this even if your regular installation is of another version of Linux.

Emergency recovery partition If you plan ahead, you might create a small emergency installation of your preferred distribution alongside the regular installation. You should *not* mount this system in `/etc/fstab`. This system can be useful for recovering from some problems, like software filesystem corruption, but it's not useful for others, like a total hard disk failure.

Partial reinstallation You can reinstall a minimal Linux system, and then use it to recover your original installation. This approach is much like the emergency recovery partition approach, but it takes more time at disaster recovery. On the other hand, it will work even if your hard disk is completely destroyed.

Whatever approach you choose to use, you should test it before you need it. Learn at least the basics of the tools available in any system you plan to use. If you use unusual backup tools (such as commercial backup software), be sure to copy those tools to your emergency system or have them available on a separate floppy disk. If you'll need to recover clients via network links, test those setups as well.

You may not be able to *completely* test your emergency restore tools. Ideally, you should boot the tools, restore a system, and test that the system works. This may be possible if you have spare hardware on which to experiment, but if you lack this luxury, you may have to make do with performing a test restore of a few files and testing an emergency boot procedure—say, using `LOADLIN` (a DOS-based boot loader that can boot a Linux system when LILO isn't installed or working). Note that a freshly restored system will not be bootable; you'll need a kernel on a DOS boot floppy and `LOADLIN`, or some other emergency boot system, to boot the first time. You can then reinstall LILO to restore the system's ability to boot from the hard disk.

File Manipulation Commands

Mounting, unmounting, and finding information about partitions is useful and even necessary, but to actually do anything on a Linux system, you must be able to manipulate individual files and directories. Linux provides traditional Unix commands to accomplish this task.

Navigating the Linux Filesystem

Moving about the Linux filesystem involves a few commands. It's also helpful to understand some features of common Linux shells that can help in this navigation. Some of these commands and features are similar to ones used in DOS and Windows. (This is no accident; DOS was partly modeled on Unix, and so it copied some Unix features that are now part of Linux.)

The *ls* Command

In order to manipulate files, it's helpful to know what they are. This is the job of the `ls` command, whose name is short for "list." `ls` displays the names of files in a directory. Its syntax is simple:

```
ls [options] [files]
```

The command supports a huge number of options; consult the `ls` man page for details. The most useful options include the following:

`-a` or `--all` Normally, `ls` omits files whose names begin with a dot (`.`). These dot files are often configuration files that aren't usually of interest. Adding this parameter displays dot files.

`--color` This option produces a color-coded listing that differentiates directories, symbolic links, and so on by displaying them in different colors. This works at the Linux console, in xterm windows in X, and from some types of remote logins, but some remote login programs don't support color displays.

`-d` or `--directory` Normally, if you type a directory name as one of the *`files`*, `ls` displays the contents of that directory. The same thing happens if a directory name matches a wildcard (discussed shortly). Adding this parameter changes this behavior to list only the directory name, which is sometimes preferable.

`-l` `ls` normally displays filenames only. This parameter (a lowercase L, not a digit 1) produces a long listing that includes information such as the file's permission string (discussed in Chapter 4), owner, group, size, and creation date.

`-p` or `--file-type` This option appends an indicator code to the end of each name so you know what type of file it is. (Chapter 4 discusses file types.) The meanings are as follows:

`/` directory

`@` symbolic link

`=` socket

`|` pipe

`-R` or `--recursive` This option causes `ls` to display directory contents recursively. That is, if the target directory contains a subdirectory, `ls` displays both the files in the target directory *and* the files in its subdirectory. The result can be a huge listing of a directory with many subdirectories.

Both the *`options`* list and the *`files`* list are optional. If you omit the *`files`* list, `ls` displays the contents of the current directory. You may instead give one or more file or directory names, in which case `ls` displays information on those files or directories, for instance:

```
$ ls -p /usr /bin/ls
/bin/ls

/usr:
X11R6/  games/                i386-glibc20-linux/  include/  man/    src/
bin/    i386-glibc20-linux/   i486-linux-libc5/    lib/      merge@  tmp@
doc/    i486-linux-libc5/     i586-mandrake-linux/ libexec/  sbin/
etc/    i586-mandrake-linux/  local/               share/
```

This output shows both the `/bin/ls` program file and the contents of the `/usr` directory. The latter consists mainly of subdirectories, but it includes a couple of symbolic links, as well.

Using Wildcards

You can use *wildcards* with `ls` (and with many other commands, as well). A wildcard is a symbol or set of symbols that stand in for other characters. Three classes of wildcards are common in Linux:

? A question mark (?) stands in for a single character. For instance, `b??k` matches `book`, `balk`, `buck`, or any other four-letter filename that begins with `b` and ends with `k`.

***** An asterisk (*) matches any character or set of characters, including no character. For instance, `b*k` matches `book`, `balk`, and `buck`, just as does `b??k`. `b*k` also matches `bk`, `bbk`, and `backtrack`.

Bracketed values Characters enclosed in square brackets ([]) normally match any character in the set. For instance, `b[ao][lo]k` matches `balk` and `book`, but not `buck`. It's also possible to specify a range of values; for instance, `b[a-z]ck` matches any `back`, `buck`, and other four-letter filenames of this form whose second character is a lowercase letter. This differs from `b?ck`—because Linux treats filenames in a case-sensitive way, `b[a-z]ck` doesn't match `bAck`, although `b?ck` does.

Wildcards are actually implemented in the shell and passed to the command you call. For instance, if you type **`ls b??k`**, and that wildcard matches the three files `balk`, `book`, and `buck`, the result is precisely as if you'd typed **`ls balk book buck`**.

The way wildcards are expanded can lead to some undesirable consequences. For instance, suppose you want to copy a couple of files (specified via a wildcard) to another directory, but you forget to give the destination directory. The `cp` command (discussed shortly) will interpret the command as a request to copy one of the files over the other.

Finding and Changing the Current Directory

Linux command shells implement the concept of a current directory. This is a directory that's displayed by default if `ls` or some other command doesn't specify a directory. You can discover what your current directory is by typing **`pwd`**. This command's name stands for "present working directory," and it can be very useful if you don't know in what directory you're currently operating.

You may specify either an *absolute directory name* or a *relative directory name* when giving a filename or directory name. The former indicates the directory name relative to the root directory. An absolute directory name uses a leading slash, as in `/usr/local` or `/home`. Relative directory names are specified relative to the current directory. They lack the leading slash. Relative directory names sometimes begin with a double dot (`..`). This is a code that stands for a directory's parent. For instance, if your current directory is `/usr/local`, `..` refers to `/usr`. Similarly, a single dot (`.`) as a directory name refers to the current directory. As an example, if your current directory is `/usr/local`, both `/usr/X11R6` and `../X11R6` refer to `/usr/X11R6`.

Another important shortcut character is the tilde (~). This character is a stand-in for the user's home directory. For instance, `~/document.wpd` refers to the `document.wpd` file within the user's home directory. This might be `/home/sally/document.wpd` for the user `sally`, for instance.

To change to another directory, use the `cd` command. Unlike most commands, `cd` is built into the shell (bash, `tcsh`, or what have you). Its name stands for "change directory," and it alters the current directory to whatever you specify. Type the command followed by your target directory, thus:

```
$ cd somedir
```

You may use either absolute or relative directory names with the `cd` command—or with other commands that take filenames or directory names as input.

Manipulating Files

A few file manipulation commands are extremely important to everyday file operations. These commands allow you to copy, move, rename, and delete files. (Chapter 4 discusses some additional commands related to Linux's file ownership and permissions models.)

cp

The `cp` command copies a file. Its basic syntax is as follows:

```
cp [options] source destination
```

The *`source`* is normally one or more files, and the *`destination`* may be a file (when the source is a single file) or a directory (when the source is one

or more files). When copying to a directory, cp preserves the original filename; otherwise it gives the new file the filename indicated by *destination*. The command supports a large number of options; consult its man page for more information. Following are some of the more useful options:

-f or --force This option forces the system to overwrite any existing files without prompting.

-i or --interactive This option causes cp to ask you before overwriting any existing files.

-p or --preserve Normally, a copied file is owned by the user who issues the cp command and uses that account's default permissions. This option preserves ownership and permissions, if possible.

-R or --recursive If you use this option and specify a directory as the *source*, the entire directory, including its subdirectories, will be copied.

-u or --update This option tells cp to copy the file only if the original is newer than the target, or if the target doesn't exist.

As an example, the following command copies the /etc/fstab configuration file to a backup location in /root, but only if the original /etc/fstab is newer than the existing backup:

```
# cp -u /etc/fstab /root/fstab-backup
```

mv

The mv command (short for "move") is commonly used both to move files and directories from one location to another and to rename them. Linux doesn't distinguish between these two types of operations, although many users do. The syntax of mv is similar to that of cp:

```
mv [options] source destination
```

The command takes many of the same *options* as does cp. From the earlier list, --preserve and --recursive don't apply to mv, but the others do.

To move one or more files or directories, specify the files as the *source* and specify a directory or (optionally for a single file move) a filename for the *destination*. Here is an example:

```
$ mv document.wpd important/purchases/
```

This command copies the document.wpd file into the important/purchases subdirectory. If the copy occurs on one low-level filesystem,

Linux does the job by rewriting directory entries; the file's data don't need to be read and rewritten. This makes `mv` fast. When the target directory is on another partition or disk, though, Linux must read the original file, rewrite it to the new location, and delete the original. This slows down `mv`. Also, `mv` can move entire directories within a filesystem, but not between filesystems.

The preceding example used a trailing slash (`/`) on the destination directory. This practice can help avoid problems caused by typos. For instance, if the destination directory were mistyped as `important/purchase` (missing the final `s`), `mv` would move `document.wpd` into the `important` directory under the filename `purchase`. Adding the trailing slash makes it explicit that you intend to move the file into a subdirectory. If it doesn't exist, `mv` complains, so you're not left with mysterious misnamed files. You can also use the Tab key to avoid problems. When you hit Tab in many Linux shells, such as bash, the shell tries to complete the filename automatically, reducing the risk of a typo.

Renaming a file with `mv` works much like moving a file, except that the source and destination filenames are in the same directory, as shown here:

```
$ mv document.wpd washer-order.wpd
```

This renames `document.wpd` to `washer-order.wpd` in the same directory. These two forms can be combined, as well:

```
$ mv document.wpd important/purchases/washer-order.wpd
```

This command simultaneously moves and renames the file.

rm

To delete a file, use the `rm` command, whose name is short for "remove." Its syntax is simple:

```
rm [options] files
```

`rm` accepts many of the same *`options`* as `cp` or `mv`. Of those discussed with `cp`, `--preserve` and `--update` are inapplicable to `rm`, but the others all apply to it. With `rm`, `-r` is synonymous with `-R`.

By default, Linux doesn't provide any sort of "trash can" functionality for its `rm` command; once you've deleted a file with `rm`, it's gone and cannot be recovered without retrieving it from a backup or performing low-level disk maintenance. Therefore, you should be cautious when using `rm`, particularly when logged on as `root`. This is particularly true when using the `-R` option—**`rm -R /`** will destroy an entire Linux installation! Many Linux GUI file managers do implement trash can functionality so that you can easily recover files moved to the trash (assuming you haven't emptied the trash), so you may want to use a file manager for removing files.

Manipulating Directories

Files normally reside in directories. Even normal users frequently create, delete, and otherwise manipulate directories. Some of the preceding commands can be used with directories—you can move or rename directories with `mv`, for instance. `rm` won't delete a directory unless used in conjunction with the `-R` parameter. Linux provides additional commands to manipulate directories.

mkdir

The `mkdir` command creates a directory. This command's official syntax is as follows:

```
mkdir [options] directory-names
```

In most cases, `mkdir` is used without *`options`*, but a few are supported, including the following:

`-m` or `--mode=`*`mode`* This option causes the new directory to have the specified permission mode, expressed as an octal number. (Chapter 4 discusses permission modes.)

`-p` or `--parents` Normally, if you specify the creation of a directory within another directory that doesn't exist, `mkdir` responds with a `No such file or directory` error and doesn't create the directory. If you include this option, though, `mkdir` creates the necessary parent directory.

rmdir

`rmdir` is the opposite of `mkdir`; it destroys a directory. Its syntax is much like that of `mkdir`:

```
rmdir [options] directory-names
```

Like `mkdir`, `rmdir` supports few options, the most important of which are as follows:

`--ignore-fail-on-non-empty` Normally, if a directory contains files or other directories, `rmdir` won't delete it and returns an error message. With this option, `rmdir` still won't delete the directory, but it doesn't return an error message.

`-p` or `--parents` This option causes `rmdir` to delete empty directories within the target directory.

When deleting an entire directory tree filled with files, `rm -R` is a better choice than `rmdir` because `rm -R` deletes files within the specified directory, but `rmdir` doesn't.

Editing Files with Vi

Vi was the first full-screen text editor written for Unix. It's designed to be small and simple. Vi is small enough to fit on tiny floppy-based emergency boot systems. For this reason alone, Vi is worth learning; you may need to use it in an emergency recovery situation. Vi is, however, a bit strange, particularly if you're used to GUI text editors.

Most Linux distributions actually ship with a variant of Vi known as Vim, or "Vi Improved." As the name implies, Vim supports more features than does the original Vi. The discussion presented here applies to both Vi and Vim. Most distributions that ship with Vim allow you to launch it by typing **`vi`**, as if it were the original Vi.

Vi Modes

At any given moment, Vi is running in one of three modes:

Command mode This mode accepts commands, which are usually entered as single letters. For instance, `i` and `a` both enter edit mode, although in somewhat different ways, as described shortly; and `o` opens a line below the current one.

Ex mode To manipulate files (including saving your current file and running outside programs), you use ex mode. You enter ex mode from command mode by typing a colon (`:`), typically directly followed by the name of the ex mode command you want to use. After running the ex mode command, Vi returns automatically to command mode.

Edit mode You enter text in edit mode. Most keystrokes result in text appearing on the screen. One important exception is the Esc key, which exits from edit mode back to command mode.

If you're not sure what mode Vi is in, press the Esc key. This will return you to command mode, from which you can reenter edit mode, if necessary.

Basic Text Editing Procedures

As a method of learning Vi, consider the task of editing `/etc/lilo.conf` to add a new kernel. Listing 7.1 shows the original `lilo.conf` file used in this example. If you want to follow along, enter it using a text editor with which you're already familiar, and save it to a file on your disk.

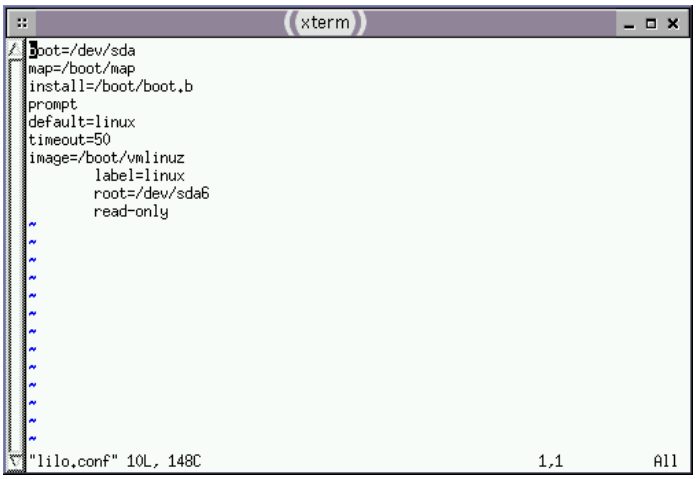
Listing 7.1: Sample `/etc/lilo.conf` File

```
boot=/dev/sda
map=/boot/map
install=/boot/boot.b
prompt
default=linux
timeout=50
image=/boot/vmlinuz
        label=linux
        root=/dev/sda6
        read-only
```

Don't try editing your *real* `/etc/lilo.conf` file as a learning exercise; a mistake could render your system unbootable the next time you type **`lilo`**. You might put your test `lilo.conf` file in your home directory for this exercise.

The first step to using Vi is to launch it and have it load the file. In this example, type **`vi lilo.conf`** while in the directory holding the file. The result should resemble Figure 7.2, which shows Vi running in an xterm window. The tildes (~) down the left side of the display indicate the end of the file. The bottom line shows the status of the last command—an implicit file load command because you specified a filename when launching the program.

FIGURE 7.2 The last line of a Vi display is a status line that shows messages from the program.



Adding a new entry to `lilo.conf` involves duplicating the lines beginning with the `image=` line and modifying the duplicates, as described in Chapter 3. Therefore, the first editing task is to duplicate these four lines. To do this, follow these steps:

1. Move the cursor to the beginning of the `image=` line by using the down arrow key; you should see the cursor resting on the `i`.
2. You must now "yank" four lines of text. This term is used much as "copy" is used in most text editors—you copy the text to a buffer from

which you can later paste it back into the file. To yank text, you use the `yy` command, preceded by the number of lines you want to yank. Thus, type **`4yy`** (*do not* press the Enter key, though). Vi responds with the message `4 lines yanked` on its bottom status line. The `dd` command works much like `yy`, but it deletes the lines as well as copying them to a buffer.

3. Move the cursor to the last line of the file by using the arrow keys.
4. Type **`p`** (again, without pressing the Enter key). Vi pastes the contents of the buffer starting on the line after the cursor. The file should now have two identical `image=` stanzas. The cursor should be resting at the start of the second one. If you want to paste the text into the document starting on the line *before* the cursor, use an uppercase `P` command.

Now that you've duplicated the necessary lines, you must modify one copy to point to your new kernel. To do so, follow these steps:

1. Move the cursor to the `v` in `vmlinuz` on the second `image=` line. You're about to begin customizing this second stanza.
2. Up until now, you've operated Vi in command mode. There are several commands that you can use to enter edit mode. At this point, the most appropriate is `R`, which enters edit mode so that it is configured for text replacement rather than insertion. If you prefer insert mode, you could use `i` or `a` (the latter advances the cursor one space, which is sometimes useful at the end of a line). For the purposes of these instructions, type **`R`** to enter edit mode. You should see `-- REPLACE --` appear in the status line.
3. Type the name of a new Linux kernel. For the purposes of this example, let's say you've called it `bzImage-2.4.3`, so that's what you'd type. This entry should replace `vmlinuz`.
4. Use the arrow keys to move the cursor to the start of `linux` on the next line. You must replace this label so that your new entry has its own label.
5. Type a new label, such as **`mykernel`**. This label should replace the existing `linux` label.
6. Exit from edit mode by pressing the Esc key.
7. Save the file and quit by typing **`:wq`**. This is actually an ex mode command, as described shortly.

There are many additional commands you might want to use in some situations. Here are some of the highlights:

Case changes Suppose you need to change the case of a word in a file. Instead of entering edit mode and retyping the word, you can use the tilde (~) key in command mode to change the case. Position the cursor on the first character you want to change and press ~ repeatedly until the task is done.

Undo To undo any change, type **u** in command mode.

Searches To search forward for text in a file, type **/** in command mode, followed immediately by the text you want to locate. Typing **?** will search backward rather than forward.

Global replacement To replace all occurrences of one string by another, type **:%s/*original*/*replacement***, where *original* is the original string and *replacement* is its replacement. Change % to a starting line number, comma, and ending line number to perform this change on just a small range of lines.

There's a great deal more depth to Vi than is presented here; the editor is quite capable, and some Linux users are very attached to it. There have been entire books written about Vi. Consult one of these, or a Vi Web page like `http://www.vim.org`, for more information.

Saving Changes

To save changes to a file, type **:w** from command mode. This enters ex mode and runs the `w` ex-mode command, which writes the file using whatever filename you specified when you launched Vi. Related commands include these:

:e This command edits a new file. For instance, **:e /etc/inittab** loads `/etc/inittab` for editing. Vi won't load a new file unless the existing one has been saved since its last change or unless you follow `:e` with an exclamation mark (`!`).

:r This command includes the contents of an old file in an existing one.

:q Use this command to quit from the program. As with `:e`, this command won't work unless changes have been saved or you append an exclamation mark to the command.

You can combine ex commands such as these to perform multiple actions in sequence. For instance, typing `:wq` writes changes and then quits from Vi.

Managing Cron Jobs

Some system maintenance tasks should be performed at regular intervals and are highly automated. For instance, the `/tmp` directory (which holds temporary files created by many users) tends to collect useless data files. Linux provides a means of scheduling tasks to run at specified times to handle such issues. This tool is the cron program, which runs what are known as *cron jobs*.

The Role of Cron

Cron is a daemon, which means that it runs continuously, looking for events that cause it to spring into action. Unlike most daemons, which are network servers, cron responds to temporal events. Specifically, it "wakes up" once a minute, examines configuration files in the `/var/spool/cron` and `/etc/cron.d` directories and the `/etc/crontab` file, and executes commands specified by these configuration files if the time matches the time listed in the files.

There are two types of cron jobs: *system cron jobs* and *user cron jobs*. System cron jobs are run as `root` and perform system-wide maintenance tasks. By default, most Linux distributions include system cron jobs that clean out old files from `/tmp`, perform *log rotation* (renaming log files and deleting old ones so that they don't grow to fill the disk), and so on. You can add to this repertoire, as described shortly. Ordinary users can create user cron jobs, which might run some user program on a regular basis. You can also create a user cron job as `root`, which might be handy if you need to perform some task at a time not allowed by the system cron jobs, which are scheduled rather rigidly.

One of the critical points to remember about cron jobs is that they run unsupervised. Therefore, you shouldn't call any program in a cron job if that program requires user input. For instance, you wouldn't run a text editor in a cron job. You might, though, run a script that automatically manipulates text files, such as log files.

Creating System Cron Jobs

The /etc/crontab file controls system cron jobs. This file normally begins with several lines that set environment variables, such as PATH and MAILTO (the former sets the path, and the latter is the address to which programs' output is mailed). The file then contains several lines that resemble the following:

```
02 4 * * * root run-parts /etc/cron.daily
```

This line begins with five fields that specify the time. The fields are, in order, the minute (0-59), the hour (0-23), the day of the month (1-31), the month (1-12), and the day of the week (0-7; both 0 and 7 correspond to Sunday). For the month and day of the week values, you can use the first three letters of the name rather than a number, if you like.

In all cases, you can specify multiple values in several ways:

- An asterisk (*) matches all possible values.
- A list separated by commas (such as 0,6,12,18) matches any of the specified values.
- Two values separated by a dash (-) indicate a range, inclusive of the end points. For instance, 9-17 in the hour field specifies a time of from 9:00 A.M. to 5:00 P.M.
- A slash, when used in conjunction with some other multivalue option, specifies stepped values—a range in which some members are skipped. For instance, */10 in the minute field indicates a job that's run every 10 minutes.

After the first five fields, /etc/crontab entries continue with the account name to be used when executing the program (root in the preceding example) and the command to be run (run-parts /etc/cron.daily in this example). The default /etc/crontab entries generally use run-parts, cronloop, or a similar utility that runs any executable scripts within a directory. Thus, the preceding example runs all the scripts in /etc/cron.daily at 4:02 A.M. every day. Most distributions include monthly, daily, weekly, and hourly system cron jobs, each corresponding to scripts in a directory called /etc/cron.*interval*, where *interval* is a word associated with the run frequency. Others place these scripts in /etc/cron.d/*interval* directories.

The exact times chosen for system cron jobs to execute vary from one distribution to another. Normally, though, daily and longer-interval cron jobs run early in the morning—between midnight and 6:00 A.M. Check your `/etc/crontab` file to determine when your system cron jobs run.

To create a new system cron job, you may create a script to perform the task you want performed (as described in Chapter 6), and copy that script to the appropriate `/etc/cron.`*`interval`* directory. The next time the run time rolls around, cron will run the script.

Before submitting a script as a cron job, test it thoroughly. This is particularly important if the cron job will run when you're not around. You don't want a bug in your cron job script to cause problems by filling the hard disk with useless files or producing thousands of e-mail messages when you're not present to quickly correct the problem.

If you need to run a cron job at a time or interval that's not supported by the standard `/etc/crontab`, you can either modify that file to change or add the cron job run time, or create a user cron job, as described shortly. If you choose to modify the system cron job facility, model your changes after an existing entry, changing the times and script storage directory as required.

System cron job storage directories should be owned by `root`, and only `root` should be able to write to them. If ordinary users can write to a system cron directory, unscrupulous users could write scripts to give themselves superuser privileges and place them in the system cron directory. The next time cron runs those scripts, the users will have full administrative access to the system.

Creating User Cron Jobs

To create a user cron job, you use the `crontab` utility, not to be confused with the `/etc/crontab` configuration file. The syntax for `crontab` is as follows:

```
crontab [-u user] [-l | -e | -r] [file]
```

If given without the `-u` *`user`* parameter, `crontab` modifies the cron job associated with the current user. (User cron jobs are often called crontabs, but with the word already used in reference to the system-wide configuration file and the utility itself, this usage can be confusing.) The `crontab` utility can become confused by the use of `su` to change the current user identity, though, so if you use this command, it's safest to also use `-u` *`user`*, even when you are modifying your own cron job.

If you want to work directly on a cron job, use one of the `-l`, `-e`, or `-r` options. `-l` causes `crontab` to display the current cron job. `-r` removes the current cron job. `-e` opens an editor so that you can edit the current cron job. (Vi is the default editor, but you can change this by setting the `VISUAL` or `EDITOR` environment variables, as described in Chapter 6.)

Alternatively, you can create a cron job configuration file and pass the filename to `crontab` using the *`file`* parameter. For instance, **`crontab -u tbaker my-cron`** causes `crontab` to use `my-cron` for `tbaker`'s cron jobs.

Whether you create the cron job and submit it via the *`file`* parameter or edit it via `-e`, the format of the cron file is similar to that described earlier. You can set environment variables by using the form *`VARIABLE`*`=`*`value`*, or you can specify a command preceded by five numbers or wildcards to indicate when the job is to run. In a user cron job, however, you do *not* specify the username used to execute the job, as you do with system cron jobs. That information is derived from the owner of the cron job. Listing 7.2 shows a sample cron job file. This file runs two programs at different intervals: The `fetchmail` program runs every thirty minutes (on the hour and half hour), and `clean-adouble` runs on Mondays at 2:00 A.M. Both programs are specified via complete paths, but you could include a `PATH` environment variable and omit the complete path specifications.

Listing 7.2: A Sample User Cron Job File

```
SHELL=/bin/bash
MAILTO=tbaker
HOME=/home/tbaker
0,30 * * * * /usr/bin/fetchmail -s
0 2 * * mon /usr/local/bin/clean-adouble $HOME
```

Handling Core Dumps

Unfortunately, few programs are perfect. One particularly important class of imperfections causes a program to *crash*—to abruptly stop working and exit. Some types of crashes on Linux cause the system to create a special file called `core`. These files are often referred to as *core dumps*, and understanding what they're for and how to take advantage of them can help you track down problem programs and the users who run them. You can then replace the software or, if you know enough about programming, you can use the core dump to help fix the software. If you lack this knowledge or don't have the time or inclination to follow through on it, you may want to delete core dumps because they can clutter your computer, and in extreme cases, they may consume enough disk space to be a problem. If your users engage in programming, you may want to leave their own core dumps alone because they may use them even if you don't.

Understanding Core Dumps

A core dump is so called because it is a recording (dump) of the memory a program was using at the time it crashed. (In very early computer history, what we now call RAM was called core memory because it was built from magnetized rings known as cores.) The reasoning behind producing a core dump is that a programmer should be able to study a crashed program to discover why it crashed. Because a crashed program is no longer active in memory, though, this task is difficult or impossible without a core dump. With a core dump, a programmer can use the `core` file to study the program's state when it crashed. If the program included appropriate debugging code, a programmer can even trace through the steps that led to the crash, thus making it relatively easy to locate the source of the problem.

Because core dumps are recordings of a program's state in memory at the time of a crash, they vary in size. A short program that operates on little data will produce a small `core` file, but a large program that works on large data sets will produce a large `core` file. In any event, these files normally appear in the directory from which the program was run.

Not all program crashes create core dumps. One common reason for this is that the user ran the program in a directory to which the user did not have write access. Another reason is that the shell may have limited the size of `core` files. In bash, this can be done with the `ulimit` command, by passing

it the `-c` parameter and the maximum core size in kilobytes. For instance, **`ulimit -c 60`** ensures that no core file greater than 60KB will be created. If a `core` file would be larger than this amount, it will be truncated to this size or smaller. Such truncated `core` files are unlikely to be very useful. Typing **`ulimit -c 0`** stops the creation of `core` files altogether. (This command is sometimes found in bash configuration files.) On the opposite end of the scale, typing **`ulimit -c unlimited`** allows the system to create `core` files of any size.

Locating and Deleting Unneeded Core Files

You can locate all `core` files on a system by typing the following command:

```
# find / -name core
```

This command will, however, return some files that aren't `core` files in the sense just described. For instance, `/dev/core` is a device file and `/proc/sys/net/core` is a directory. There's also a `core` directory in the Linux kernel source code. None of these is a `core` file in the sense of a holding area for a crashed program's memory. If you issue the preceding `find` command as an ordinary user, you'll get some `Permission denied` error messages because you are not allowed to read certain directories. As a result, you won't find `core` files in those directories.

Once you've found `core` files on a system, the question arises of what to do with them. As discussed shortly, a `core` file can be useful, but it is most useful to programmers with access to the source code for the program that created the `core` file. One of the advantages of Linux, of course, is that you have source code to most of the programs you run, so if you have the skill and inclination, you can use a `core` file to help debug a problem.

To be most useful, the program that created the `core` file must have been compiled with debugging code enabled. Such code is often not included in programs that are released as part of a Linux distribution, so if you want to debug such a program that's creating core dumps, you may need to recompile it with debugging code enabled.

It's not always obvious just what created the `core` file, and therefore whether it's useful or not. There are some clues you may want to examine, though:

Owner A `core` file is owned by the person who executed the program. This information may be useful in tracking down the source of core files in certain directories and in determining whether or not to delete the files. For instance, if you know that `smccoy` is writing programs on the system and you find `core` files owned by `smccoy`, you might not want to delete them immediately, or at least you might want to ask `smccoy` what to do with them. It's possible that the user needs these `core` files, or they could just be rubbish that's cluttering the system.

Creation date Like all files, `core` files have creation dates. To use the core file, you normally examine it to trace the actions that caused a crash. This is best done when the crash is fresh in your mind. Therefore, the older a `core` file, the less likely it is to be useful.

Creating program Although `core` files all have the same filename, it's possible to determine what program created the file. You can do this by typing **`gdb -c core`**. This will launch `gdb`, the GNU debugger, which will examine the `core` file and report, among other things, the name of the program that generated the file. You'll then need to type **`quit`** to exit from `gdb`. At the very least, this information should help you decide whether a `core` file represents, say, a crash of a program under development or a crash of a standard tool on the computer.

With this information in hand, you can decide what to do with the `core` file. You might decide to move it to a user's directory and send an e-mail to that user about that fact; replace a buggy utility that's crashing; attempt further debugging yourself; delete the `core` file; send the file to the program's author (after checking that the author wants it); or something else. Note that this is a judgment call; an appropriate action for one system may not be appropriate for another. The use to which a system is put, the expertise of its users, and your own expertise are important factors in deciding what to do with a `core` file.

You may be tempted to automate the process of searching for and deleting `core` files. I don't recommend automatically deleting all `core` files on a computer because there are, as noted earlier, several files and directories named `core` that are not core dumps. Deleting one of these files by mistake can cause problems. If a system's users write programs and use `core` files for

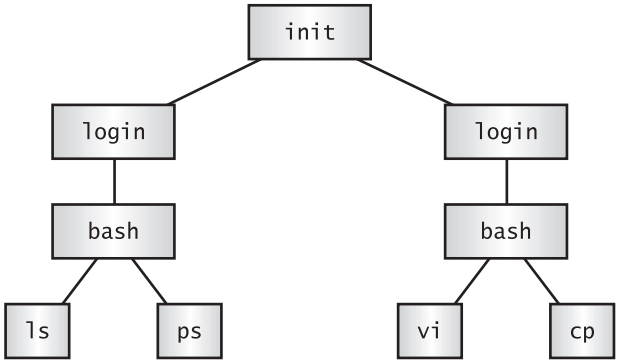
debugging, deleting them might earn you these users' ire. Even moving the files can cause problems if the move makes these files inaccessible. You might, however, include the `find` command noted earlier in a system cron job. If you do this, assuming the `MAILTO` environment variable is set correctly, the result will be a report of all the `core` files on the system every time the job runs. A weekly or monthly report such as this will alert you to the presence of any `core` files you might want to manually inspect.

Managing Processes

Even programs that don't crash outright occasionally misbehave in other ways. For instance, a program might stop responding, or it may consume an inordinate amount of CPU time. In these cases, it's important that you know how to exercise superuser control over these programs so that you can rein in their appetites or terminate them outright. The first step to doing this, though, is knowing how to find out what programs are running on the computer, so that's where this section begins.

Before proceeding, though, it's important that you understand a bit of terminology. In Linux, a *process* is more or less synonymous with a running program. Because Linux is a multiuser, multitasking OS, it's possible for one program to be running as more than one process at a time, however. For instance, suppose that `tbaker` and `smccoy` both use Vi to edit text files. The computer will have two Vi processes running at once. Indeed, a single user can do this. It's also possible for a single program to create (or *spawn*) subprocesses. For instance, Vi can launch a spell checker program. In fact, this is what happens when you launch a program from a shell—the shell spawns the program you're launching. When one process spawns another, the original process is known as the *parent process*, and the spawned subprocess is known as the *child process*. This parent/child relationship produces a tree-like hierarchy that ultimately leads back to `init`, the first process (described in Chapter 6). Figure 7.3 shows a simplified example. In Figure 7.3, `init` spawns the `login` processes, which in turn spawn `bash` processes, which spawn additional processes. (It's actually slightly more complex than this. `init` doesn't directly spawn `login`; instead, it does this by using another process, such as `getty`.) This can continue for an arbitrary number of layers, although many programs aren't able to spawn others.

FIGURE 7.3 Linux processes have parents, leading back to `init`, the first program the Linux kernel runs.



Examining Process Lists with *ps*

One of the most important tools in process management is `ps`. This program displays processes' status (hence the name, `ps`). It sports many useful options, and it's useful in monitoring what's happening on a system. This can be particularly critical when the computer isn't working as it should be—for instance, if it's unusually slow.

Useful *ps* Options

The official syntax for `ps` is fairly simple:

```
ps [options]
```

This simplicity of form hides considerable complexity because `ps` supports three different *types* of options, as well as many options within each type. The three types of options are as follows:

Unix98 options These single-character options may be grouped together and are preceded by a single dash (`-`).

BSD options These single-character options may be grouped together and must *not* be preceded by a dash.

GNU long options These multicharacter options are not grouped together. They're preceded by two dashes (`--`).

Options that may be grouped together may be clustered without spaces between them. For instance, rather than typing **`ps -a -f`**, you can type **`ps`**

-af. The reason for so much complexity is that the ps utility has historically varied a lot from one Unix OS to another. The version of ps that ships with major Linux distributions attempts to implement most features from all these different ps versions, so it supports many different personalities. In fact, you can change some of its default behaviors by setting the PS_PERSONALITY environment variable to posix, old, linux, bsd, sun, digital, or various others. The rest of this section describes the default ps behavior on most Linux systems.

Some of the more useful ps options include the following:

-A, -e, or x By default, ps displays only processes that were run from its own terminal (xterm, text-mode login, or remote login). The -A and -e options cause it to display all the processes on the system, and x displays all processes owned by the user who gives the command. x also increases the amount of information that's displayed about each process.

-u *user*, U *user*, or --User *user* You can display processes owned by a given user with these options. *user* may be a username or a user ID.

-f, -l, j, l, u, and v These options all expand the information provided in the ps output. Most ps output formats include one line per process, but ps can display enough information that it's impossible to fit it all on one line. Therefore, these options provide various mixes of information.

-H, f, or --forest These options group processes and use indentation to show the hierarchy of relationships between processes. This is very useful if you're trying to trace the parentage of a process.

-w or w ps output can be more than 80 columns wide. Normally, ps truncates its output so it will fit on your screen or xterm. These options tell ps not to do this, which can be useful if you direct the output to a file, as in **ps w > ps.txt**. You can then examine the output file in a text editor that supports wide lines.

You can combine these ps options in many ways to produce the output you want. You'll probably need to experiment to learn which options produce the desired results because each of these options modifies the output in some way. Even those that would seem to influence just the selection of processes to list sometimes modify the information that's provided about each process.

Interpreting *ps* Output

Listings 7.3 and 7.4 show a couple of examples of `ps` in action. Listing 7.3 shows **`ps -u rodsmith --forest`**, and Listing 7.4 shows **`ps u U rodsmith`**.

Listing 7.3: Output of `ps -u rodsmith --forest`

```
$ ps -u rodsmith --forest
  PID TTY          TIME CMD
 2451 pts/3    00:00:00 bash
 2551 pts/3    00:00:00 ps
 2496 ?        00:00:00 kvt
 2498 pts/1    00:00:00 bash
 2505 pts/1    00:00:00  \_ nedit
 2506 ?        00:00:00      \_ csh
 2544 ?        00:00:00          \_ xeyes
19221 ?        00:00:01 dfm
```

Listing 7.4: Output of `ps u U rodsmith`

```
$ ps u U rodsmith
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START
TIME COMMAND
rodsmith 19221  0.0  1.5  4484 1984 ?        S    May07
↳0:01 dfm
rodsmith  2451  0.0  0.8  1856 1048 pts/3    S    16:13
↳0:00 -bash
rodsmith  2496  0.2  3.2  6232 4124 ?        S    16:17
↳0:00 /opt/kd
rodsmith  2498  0.0  0.8  1860 1044 pts/1    S    16:17
↳0:00 bash
rodsmith  2505  0.1  2.6  4784 3332 pts/1    S    16:17
↳0:00 nedit
rodsmith  2506  0.0  0.7  2124 1012 ?        S    16:17
↳0:00 /bin/cs
rodsmith  2544  0.0  1.0  2576 1360 ?        S    16:17
↳0:00 xeyes
rodsmith  2556  0.0  0.7  2588  916 pts/3    R    16:18
↳0:00 ps u U
```

The output produced by `ps` normally begins with a heading line, which displays the meaning of each column. Important information that might be displayed (and labeled) includes the following:

Username The name of the user who runs the programs. Listings 7.3 and 7.4 restricted this output to one user to limit the size of the listings.

Process ID The process ID (PID) is a number that's associated with the process. This item is particularly important because you need it to modify or kill the process, as described later in this chapter.

Parent process ID The parent process ID (PPID) identifies the process's parent.

TTY The teletype (TTY) is a code used to identify a terminal. As illustrated by Listings 7.3 and 7.4, not all processes have TTY numbers—X programs and daemons, for instance, do not. Text-mode programs do have these numbers, though, which point to a console, xterm, or remote login session.

CPU time The `TIME` and `%CPU` headings are two measures of CPU time used. The first indicates the total amount of CPU time consumed, and the second represents the percentage of CPU time the process is using when `ps` executes. Both can help you spot runaway processes—those that are consuming too much CPU time. Unfortunately, just what constitutes "too much" varies from one program to another, so it's impossible to give a simple rule to help you spot a runaway process.

CPU priority As described shortly, in "Restricting Processes' CPU Use," it's possible to give different processes different priorities for CPU time. The `NI` column lists these priority codes. The default value is 0. Positive values represent *reduced* priority, while negative values represent *increased* priority.

Memory use Various headings indicate memory use—for instance, `RSS` is resident set size (the memory used by the program and its data) and `%MEM` is the percentage of memory the program is using. Some output formats also include a `SHARE` column, which is memory that's shared with other processes (such as shared libraries). As with CPU use measures, these can help point you to the sources of difficulties, but because legitimate memory needs of programs vary so much, it's impossible to give a simple criterion for when a problem exists.

Command The final column in most listings is the command used to launch the process. This is truncated in Listing 7.4 because this format lists the complete command, but so much other information appears that the complete command won't usually fit on one line. (This is where the wide-column options can come in handy.)

As you can see, there's a lot of information that can be gleaned from a `ps` listing—or perhaps that should be the plural *listings*, because no one format includes all of the available information. For the most part, the PID, username, and command are the most important pieces of information. In some cases, though, you may need specific other components. If your system's memory or CPU use has skyrocketed, for instance, you'll want to pay attention to the memory or CPU use columns.

It's often necessary to find specific processes. You might want to find the PID associated with a particular command in order to kill it, for instance. This information can be gleaned by piping the `ps` output through `grep`, as in **`ps ax | grep bash`** to find all the instances of bash. (Both `grep` and pipes are covered in more detail in Chapter 9.)

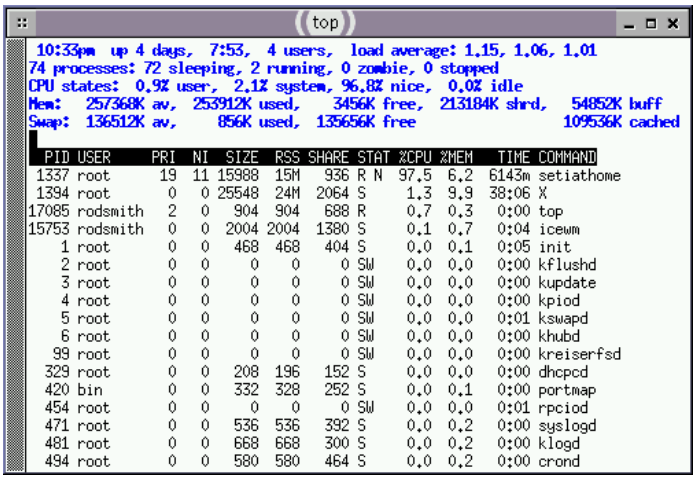
Although you may need a wide screen or xterm to view the output, you may find **`ps -A --forest`** to be a helpful command in learning about your system. Processes that don't fall off of others are started directly by `init` or have had their parents killed, and so they have been "adopted" by `init`. Most of these processes are fairly important—they're servers, login tools, and so on. Processes that hang off of several others in this tree view, such as `xeyes` and `nedit` in Listing 7.3, are mostly user programs launched from shells.

top: A Dynamic *ps* Variant

If you want to know how much CPU time various processes are consuming relative to one another, or if you simply want to quickly discover which processes are consuming the most CPU time, a tool called `top` is the one for the job. `top` is a text-mode program, but of course it can be run in an xterm, as shown in Figure 7.4, and there are also GUI variants, like `gtop`. By default, `top` sorts its entries by CPU use, and it updates its display every few seconds. This makes it a very good tool for spotting runaway processes on an otherwise lightly loaded system—those processes almost always appear in the first position or two, and they consume an inordinate amount of CPU time. By looking at Figure 7.4, you might think that `setiathome` is such a process, but in fact, it's legitimately consuming a lot of CPU time. You'll need to be familiar with the purposes and normal habits of programs running on *your*

system in order to make such determinations; the legitimate needs of different programs vary so much that it's impossible to give a simple rule for judging when a process is consuming too much CPU time.

FIGURE 7.4 `top` shows system summary information and information on the most CPU-intensive processes on a computer.



Like many Linux commands, `top` accepts several options. The following is a list of the most useful of these options:

`d` *`delay`* This specifies the delay between updates, which is normally five seconds.

`p` *`pid`* If you want to monitor specific processes, you can list them using this option. You'll need the PIDs, which you can obtain with `ps`, as described earlier. You can specify up to 20 PIDs by using this option multiple times, once for each PID.

`n` *`iter`* You can tell `top` to display a certain number of updates (*`iter`*) and then quit. (Normally, `top` continues updating until you exit from the program.)

`b` This specifies batch mode, in which `top` doesn't use the normal screen update commands. You might use this to log CPU use of targeted programs to a file, for instance.

You can do more with `top` than watch it update its display. When it's running, you can enter any of several single-letter commands, some of which prompt you for additional information. These commands include the following:

`h` or `?` These keystrokes display help information

k You can kill a process with this command. `top` will ask for a PID number, and if it's able to kill it, it will do so. (The upcoming section "Killing Processes" describes other ways to kill processes.)

q This option quits from `top`.

r You can change a process's priority with this command. You'll have to enter the PID number and a new priority value—a positive value will decrease its priority and a negative value will increase its priority, assuming it has the default 0 priority to begin with. Only `root` may increase a process's priority. The `renice` command (discussed shortly, in "Restricting Processes' CPU Use") is another way to accomplish this task.

s This command changes the display's update rate, which you'll be asked to enter (in seconds).

P This sets the display to sort by CPU usage, which is the default.

M You can change the display to sort by memory usage with this command.

There are more commands available in `top` (both command-line options and interactive commands) than can be summarized here; consult the `top` man page for more information.

One of the pieces of information provided by top is the *load average*, which is a measure of the demand for CPU time by applications. In Figure 7.4, you'll see three load average estimates on the top line; these correspond to the current load average and two previous measures. A system on which no programs are demanding CPU time will have a load average of 0. A system with one program running CPU-intensive tasks will have a load average of 1. Higher load averages reflect programs competing for available CPU time. You can also find the current load average via the `uptime` command, which displays the load average along with information on how long the computer has been running. The load average can be very useful in detecting runaway processes. For instance, if a system normally has a load average of 0.5, but it suddenly gets stuck at a load average of 2.5, there may be a couple of CPU-hogging processes that have *hung*—that is, become unresponsive. Hung processes sometimes needlessly consume a lot of CPU time. You can use `top` to locate these processes and, if necessary, kill them.

Restricting Processes' CPU Use

There may be times when you'll want to prioritize your programs' CPU use. For instance, you might be running a program that's very CPU-intensive but

that will take a long time to finish its work, and you don't want that program to interfere with others that are of a more interactive nature. Alternatively, on a heavily loaded computer, you might have a job that's more important than others that are running, so you might want to give it a priority boost. In either case, the usual method of accomplishing this goal is through the `nice` and `renice` commands. You can use `nice` to launch a program with a specified priority, or use `renice` to alter the priority of a running program.

You can give a priority to `nice` in any of three ways: by specifying the priority preceded by a dash (this works well for positive priorities, but makes them look like negative priorities); by specifying the priority after a `-n` parameter; or by specifying the priority after the `--adjustment=` parameter. In all cases, these parameters are followed by the name of the program you want to run. For instance, the following three commands are all equivalent:

```
$ nice -12 number-crunch data.txt
$ nice -n 12 number-crunch data.txt
$ nice --adjustment=12 number-crunch data.txt
```

All three of these commands run the `number-crunch` program at priority 12, and pass it the `data.txt` file. If you omit the adjustment value, `nice` uses 10 as a default. The range of possible values is –20 to 19, with negative values having the highest priority. Only `root` may launch a program with increased priority (that is, give a negative priority value), but any user may use `nice` to launch a program with low priority. The default priority for a program run without `nice` is 0.

If you've found that a running process is consuming too much CPU time or is being swamped by other programs and so should be given more CPU time, you can use the `renice` program to alter its priority without disrupting the program's operation. The syntax for `renice` is as follows:

```
renice priority [[-p] pids] [[-g] pgrps] [[-u] users]
```

You must specify the *`priority`*, which takes the same values as with `nice`. In addition, you must specify one or more PIDs (*`pids`*), one or more group IDs (*`pgrps`*), or one or more usernames (*`users`*). In the latter two cases, `renice` changes the priority of all programs that match the specified criterion—but only `root` may use `renice` in this way. Also, only `root` may increase a process's priority. If you give a numeric value without a `-p`, `-g`, or `-u` option, `renice` assumes the value is a PID. You may mix and match these

methods of specification. For instance, you might enter the following command:

```
# renice 7 16580 -u pdavison tbaker
```

This command sets the priority to 7 for PID 16580 and for all processes owned by `pdavison` and `tbaker`.

Killing Processes

Sometimes reducing a process's priority isn't a strong enough action. A program may have become totally unresponsive, or you might want to terminate a process that shouldn't be running at all. In these cases, the `kill` command is the tool to use. This program sends a *signal* (a method that Linux uses to communicate with processes) to a process. The signal is usually sent by the kernel, the user, or the program itself to terminate the process. Linux supports many numbered signals, each of which is associated with a specific name. You can see them all by typing **`kill -l`**. If you don't use `-l`, the syntax for `kill` is as follows:

```
kill -s signal pid
```

The `-s` *`signal`* parameter sends the specified signal to the process. You can specify the signal using either a number (such as 9) or a name (such as `SIGKILL`). The signals you're most likely to use are 1 (`SIGHUP`, which causes many daemons to reread their configuration files), 9 (`SIGKILL`, which causes the process to exit without performing routine shutdown tasks), and 15 (`SIGTERM`, which causes the process to exit but allows it to close open files and so on). If you don't specify a signal, the default is 15 (`SIGTERM`). You can also use the shortened form *`-signal`*. If you do this and use a signal name, you should omit the `SIG` portion of the name—for instance, use `KILL` rather than `SIGKILL`. *`pid`* is, of course, the PID for the process you want to kill. You can obtain this number from `ps` or `top`.

The `kill` program will only kill processes owned by the user who runs `kill`. The exception is if that user is `root`; the superuser may kill any user's processes.

A variant on `kill` is `killall`. This command kills a process based on its name, rather than its PID number. For instance, **`killall vi`** kills all the running processes called `vi`. You may specify a signal in the shortened form

(*-signal*). As with `kill`, the default is 15 (`SIGTERM`). One potentially important option to `killall` is `-i`, which causes it to ask for confirmation before sending the signal to each process. You might use it like this:

```
$ killall -i vi
Kill vi(13211) ? (y/n) y
Kill vi(13217) ? (y/n) n
```

In this example, there were two instances of the Vi editor running, but only one should have been killed. As a general rule, if you run `killall` as `root`, you should use the `-i` parameter; if you don't, it's all too likely that you'll kill processes that you should not, particularly if the computer is being used by many people at once.

Some versions of Unix provide a `killall` command that works very differently from Linux's `killall`. This alternate `killall` kills all the processes started by the user who runs the command. This is a potentially much more destructive command, so if you ever find yourself on a non-Linux system, *do not* use `killall` until you've discovered what that system's `killall` does, say by reading the `killall` man page.

Summary

Linux uses a unified filesystem, which means it doesn't use drive letters as Windows does. Instead, partitions are mounted within a single directory structure, starting at the root (`/`) partition. You can create filesystems on partitions or removable disks, mount them, store files on them, and back them up individually or across partitions. The organization of these directories and filesystems is described in the FHS, which defines where files should reside in a Linux system. Manipulating individual files can be extremely important on a Linux system, and Linux includes many tools to help you do this, ranging from commands for copying and moving files to editors like Vi for manipulating the contents of files.

Certain files are associated with particular types of running programs. For instance, control files manage cron jobs, which are processes that run at regularly scheduled times. Core dumps are another type of file that's associated with running processes, but these are processes gone bad—programs that have crashed leave core dumps as a clue to what caused them to crash. You

may want to delete these, examine them, or send them to the individual whose program crashed as a debugging aid.

Even when a program doesn't crash, it might cause other problems, such as consuming too much memory or CPU time. You may need to identify such problem processes and take action to ensure that they don't bring your system to its knees.

Exam Essentials

Explain the operation of the `mount` command. In its basic form, `mount` takes a device filename and directory and ties the two together so that files on the device may be accessed in the specified directory. A number of parameters and options can modify its function or how it treats the filesystem that it mounts.

Summarize backup hardware options. Backup hardware includes tapes, dedicated hard disks, removable disks, and optical media. Tapes are the most common type of backup hardware, but each of the others has its place for particular backup types.

Describe how you can learn how much disk space a system is using. The `du` command reports on the disk space used by a single directory or a group of directories. The `df` command summarizes the disk space consumed on entire disk partitions.

Summarize Vi's three editing modes. You enter text using the edit mode, which allows for text entry and deletion. The command and ex modes are used to perform more complex commands or run outside programs to operate on the text entered or changed in edit mode.

Create a new filesystem on a disk or partition. The `mkfs` program creates new filesystems on removable media drives or hard disk partitions. This program is actually a front-end to programs that do the actual work, such as `mke2fs` (aka `mkfs.ext2`) for ext2fs.

Check a filesystem for errors. The `fsck` program checks a filesystem's internal consistency. Like `mkfs`, it's a front end to filesystem-specific programs, such as `e2fsck` (aka `fsck.ext2`) for ext2fs.

Create a cron job. You create a system cron job by placing a script in an appropriate directory, such as `/etc/cron.daily`. You can create a user cron job by using the `crontab` command, which allows you to edit a script or pass one to the utility for appropriate handling.

Limit the CPU time used by a process. You can launch a program with `nice`, or use `renice` to alter its priority in obtaining CPU time. If a process is truly out of control, you can terminate it with the `kill` command.

Commands in This Chapter

Command	Description
`du`	Displays disk usage information for a file, directory, or set of files or directories
`mount`	Mounts a partition or device to a specified location in the Linux directory tree
`umount`	Removes a partition or device from its location in the Linux directory tree
`df`	Displays disk usage information for one or all mounted partitions or devices
`ls`	Displays the contents of a directory, or information on a file
`pwd`	Displays the present working directory
`cd`	Changes the present working directory
`cp`	Copies one or more files or directories
`mv`	Moves or renames one or more files or directories

Command	Description
`rm`	Deletes one or more files or directories
`mkdir`	Creates a directory
`rmdir`	Deletes a directory
`fdisk`	Modifies partitions on an *x*86 computer
`mkfs`	Creates a filesystem
`crontab`	Creates a user cron job
`ulimit`	Limits the size of `core` files
`gdb`	Debugs programs; dissects `core` files
`ps`	Displays process status information
`top`	Dynamic variant of `ps`; shows most CPU-hungry programs and updates the display periodically
`nice`	Runs a program with a specified priority
`renice`	Changes a running program's priority
`kill`	Terminates a process based on its PID
`killall`	Terminates a process based on its name

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

absolute directory name

child process

core dump

crash

cron job
device file
Filesystem Hierarchy Standard (FHS)
Filesystem Standard (FSSTND)
hung
incremental backup
load average
log rotation
mounted
non-rewinding tape device
parent process
process
random access
relative directory name
rewinding tape device
root directory
root filesystem
sequential access
shareable files
signal
spawn
static files
system cron job
unshareable files
user cron job
variable files
virtual filesystem
wildcard

Review Questions

1. You've installed a commercial spreadsheet program, called WonderCalc, on a workstation. In which of the following directories are you *most* likely to find the program executable file?

   A. `/usr/sbin`

   B. `/etc/X11`

   C. `/bin`

   D. `/opt/wcalc/bin`

2. Under which of the following circumstances is it most important to create a large partition to hold `/var`?

   A. The computer hosts several user accounts, and many of these users create large data files.

   B. The computer functions exclusively as a print server for a seldom-used printer.

   C. The computer is a workstation used to run X programs on another system.

   D. The computer operates as a mail server for hundreds of users.

3. Typing **`fdisk -l /dev/hda`** produces a listing of four partitions: `/dev/hda1`, `/dev/hda2`, `/dev/hda5`, and `/dev/hda6`. Which of the following is true?

   A. The disk contains two primary partitions and two extended partitions.

   B. Either `/dev/hda1` or `/dev/hda2` is an extended partition.

   C. The partition table is corrupt; there should be a `/dev/hda3` and a `/dev/hda4` before `/dev/hda5`.

   D. If you add a `/dev/hda3` with `fdisk`, `/dev/hda5` will become `/dev/hda6`, and `/dev/hda6` will become `/dev/hda7`.

4. Which of the following pieces of information can `df` *not* report?

   A. How long the filesystem has been mounted

   B. The number of inodes used on an ext2fs partition

   C. The filesystem type of a partition

   D. The percentage of available disk space used on a partition

5. Which of the following commands backs up the `/home` directory to an EIDE/ATAPI tape drive?

   A. `tar cvlpf /home /dev/st0`

   B. `tar cvlpf /home /dev/ht0`

   C. `tar cvf /dev/st0 /home`

   D. `tar cvf /dev/ht0 /home`

6. What is wrong with the following commands, which are intended to record an incremental backup on a tape that already holds one incremental backup?

```
# mt -f /dev/st0 fsf 1
# tar cvlpf /dev/st0 --listed-incremental /root/inc
↳/home
```

   A. The `mt` command should terminate in `2`, rather than `1`, to skip to the second position on the tape.

   B. When backing up `/home`, the incremental file must reside in `/home`, not in `/root`.

   C. The device filename should be a non-rewinding name (such as `/dev/nst0`), not a rewinding name (`/dev/st0`).

   D. The incremental backup must include the root (`/`) directory; it cannot include only `/home`.

7. You want to discover the sizes of several dot files in a directory. Which of the following commands might you use to do this?

A. `ls -la`

B. `ls -p`

C. `ls -R`

D. `ls -d`

8. You want to move a file from your hard disk to a floppy disk. Which of the following is true?

   A. You'll have to use the `--preserve` option to `mv` to keep ownership and permissions set correctly.

   B. The `mv` command will adjust filesystem pointers without physically rewriting data if the floppy uses the same filesystem type as the hard disk partition.

   C. You must use the same filesystem type on both media to preserve ownership and permissions.

   D. `mv` will delete the file on the hard disk after copying it to the floppy.

9. You type **`mkdir one/two/three`** and receive an error message that reads, in part, `No such file or directory`. What can you do to overcome this problem? (Choose all that apply.)

   A. Add the `--parents` parameter to the `mkdir` command.

   B. Issue three separate `mkdir` commands: **`mkdir one`**, then **`mkdir one/two`**, then **`mkdir one/two/three`**.

   C. Type **`touch /bin/mkdir`** to be sure the `mkdir` program file exists.

   D. Type **`rmdir one`** to clear away the interfering base of the desired new directory tree.

10. Which mode in Vi would you use to type text?

    A. Ex mode

    B. Command mode

    C. Type mode

    D. Edit mode

11. How would you remove two lines of text from a file using Vi?

    A. In command mode, position the cursor on the first line and type **2dd**.

    B. In command mode, position the cursor on the last line and type **2yy**.

    C. In edit mode, position the cursor at the start of the first line, hold the shift key down while pressing the down arrow key twice, and hit the Delete key on the keyboard.

    D. In edit mode, position the cursor at the start of the first line and press Ctrl+K twice.

12. You run Linux's `fdisk` and modify your partition layout. Before exiting from the program, though, you realize that you've been working on the wrong disk. What can you do to correct this problem?

    A. Nothing; the damage is done, so you'll have to recover data from a backup.

    B. Type **w** to exit from `fdisk` without saving changes to disk.

    C. Type **q** to exit from `fdisk` without saving changes to disk.

    D. Type **u** repeatedly to undo the operations you've made in error.

13. What does the following command accomplish?

```
# mkfs -V -t ext2 /dev/sda4
```

    A. It sets the partition table type code for `/dev/sda4` to `ext2`.

    B. It converts a FAT partition into an ext2fs partition without damaging the partition's existing files.

    C. It creates a new ext2 filesystem on `/dev/sda4`, overwriting any existing filesystem and data.

    D. Nothing; the `-V` option isn't valid, and so it causes `mkfs` to abort its operation.

14. Which of the following tasks is likely to be handled by a cron job? (Choose all that apply.)

A. Starting an important server when the computer boots

B. Finding and deleting old temporary files

C. Scripting supervised account creation

D. Monitoring the status of servers and e-mailing a report to the superuser

15. Which of the following lines, if used in a user cron job, will run `/usr/local/bin/cleanup` twice a day?

A. `15 7,19 * * * tbaker /usr/local/bin/cleanup`

B. `15 7,19 * * * /usr/local/bin/cleanup`

C. `15 */2 * * * tbaker /usr/local/bin/cleanup`

D. `15 */2 * * * /usr/local/bin/cleanup`

16. Which of the following bash commands prevents programs run from the bash shell from creating core dumps?

A. **`core -limit 0`**

B. **`climit -0`**

C. **`dump --none`**

D. **`ulimit -c 0`**

17. What is the safest way to automate the processing of `core` files?

A. Create a cron job that searches for and deletes all files called `core`.

B. Create a cron job that searches for all files called `core` and e-mails a report to you on its findings.

C. Create a cron job that searches for files called `core` and deletes those not on an exception list.

D. Create a cron job that searches for files called `core` and moves them to a special directory in which you can study them.

18. What process lies at the root of the Linux process hierarchy?

A. The BIOS

B. LILO

C. `init`

D. `ps`

19. A workstation ordinarily runs with a load average of 0.25. Suddenly, its load average is 1.25. Which of the following might you suspect, given this information? (Choose all that apply.)

A. The workstation's user may be running more programs or more CPU-intensive programs than usual.

B. A process may have hung—locked itself in a loop consuming CPU time but doing no useful work.

C. A process may have begun consuming an inordinate amount of memory.

D. The CPU may be malfunctioning and require replacement.

20. Which of the following commands is *most* likely to stop a runaway process with PID 2939?

A. `kill -s SIGHUP 2939`

B. `kill -s SIGTERM 2939`

C. `kill -s SIGKILL 2939`

D. `kill -s SIGDIE 2939`

Answers to Review Questions

1. D. The `/opt` directory tree exists to hold programs that aren't a standard part of a Linux distribution, such as commercial programs. These programs should install in their own directories under `/opt`; these directories usually have `bin` subdirectories of their own, although this isn't required. `/usr/sbin` holds programs that are normally run only by the system administrator, and `/bin` holds critical basic binary files. Neither is an appropriate place for a spreadsheet program. `/etc/X11` holds X-related configuration files.

2. D. Most mail servers store mail files in `/var`, so creating a large and separate `/var` partition can help isolate that disk activity from other partitions, improving safety in case of a disk error. Of the remaining uses, only option B is likely to generate more than normal accesses to `/var` (to store printer queue files), and because the print server is seldom used, this is less likely to require a large separate `/var` partition than is a heavily used mail server.

3. B. Logical partitions are numbered from 5 and up, and they reside inside an extended partition with a number between 1 and 4. Therefore, one of the first two partitions must be an extended partition that houses partitions 5 and 6. Because logical partitions are numbered starting at 5, their numbers won't change if `/dev/hda3` is subsequently added. The disk holds one primary, one extended, and two logical partitions.

4. A. A default use of `df` reports the percentage of disk space used. The number of inodes and filesystem types can both be obtained by passing parameters to `df`. This utility does *not* report how long a filesystem has been mounted.

5. D. The device filename for an EIDE/ATAPI tape drive is `/dev/ht0`; `/dev/st0` refers to a SCSI tape drive. The `tar` filename must follow the `--file` (`f`) qualifier; the first two options try to back up the contents of the tape device to the `/home` file.

6. C. The `/dev/st0` device (and `/dev/ht0`, for that matter) rewinds after every operation. Therefore, the first command as given will wind past the first incremental backup, and then immediately rewind. The second command will therefore overwrite the first incremental backup.

7. A. The `-l` parameter produces a long listing, including file sizes. The `-a` parameter produces a listing of all files in a directory, including the dot files. Combining the two produces the desired information (along with information on other files).

8. D. When moving from one partition or disk to another, `mv` must necessarily read and copy the file, then delete the original if that copy was successful. If both filesystems support ownership and permissions, they'll be preserved; `mv` doesn't need an explicit `--preserve` option to do this, and this preservation isn't reliant upon having exactly the same filesystem types. Although `mv` doesn't physically rewrite data when moving within a single low-level filesystem, this approach cannot work when copying to a separate low-level filesystem (such as from a hard disk to a floppy disk); if the data isn't written to the new location, it won't be accessible should the disk be inserted in another computer.

9. A, B. If you try to create a directory inside a directory that doesn't exist, `mkdir` responds with a `No such file or directory` error. The `--parents` parameter tells `mkdir` to automatically create all necessary parent directories in such situations. You can also manually do this by creating each necessary directory separately. (It's possible that **`mkdir one`** wouldn't be necessary in this example, if the directory `one` already exists, but no harm will come from trying to create a directory that already exists, although `mkdir` will return a `File exists` error.)

10. D. Edit mode is used for entering text. Ex mode is used for file operations (including loading, saving, and running external programs). Command mode is used for entering commands of various sorts. There is no "type mode" in Vi.

11. A. `dd` is the command-mode command to delete lines. Preceding this command by a number deletes that number of lines. `yy` works similarly, but it copies ("yanks") text rather than deleting it. Option C works in many more modern text editors, but not in Vi. Option D works in Emacs and similar text editors, but not in Vi.

12. C. Linux's `fdisk` doesn't write changes to disk until you exit from the program by typing **`w`**. Typing **`q`** exits without writing those changes, so typing **`q`** in this situation will avert disaster. Typing **`w`** would be precisely the wrong thing to do. Typing **`u`** would do nothing useful since it's not an undo command.

13. C. `mkfs` creates a new filesystem, overwriting any existing data and therefore making existing files inaccessible. This command does not set the partition type code in the partition table. The `-V` option is valid; it causes `mkfs` to be more verbose in reporting its activities.

14. B, D. Cron is a good tool for performing tasks that can be done in an unsupervised manner, like deleting old temporary files or checking to see that servers are running correctly. Tasks that require interaction, like creating accounts, are not good candidates for cron jobs, which must execute unsupervised. Although a cron job could restart a crashed server, it's not normally used to start a server when the system boots; that's done through SysV startup scripts or a super server.

15. B. User cron jobs don't include a username specification (`tbaker` in options A and C). The `*/2` specification for the hour in options C and D causes the job to execute every other hour; the `7,19` specification in options A and B causes it to execute twice a day, on the 7th and 19th hours (in conjunction with the `15` minute specification, that means at 7:15 A.M. and 7:15 P.M.).

16. D. `ulimit` is a bash command that can be used to set a limit on the size of core dumps. Specifying `-c 0` sets a limit of 0KB, so no core files will be created.

17. B. Some files may be called `core` but not be core dumps. Therefore, any script that automatically deletes or moves files called `core` can potentially do damage. Furthermore, automatic movement or deletion can cause problems for users who may rely upon core files to debug programs they're writing. An exception list can mitigate these problems but not eliminate them.

18. C. Although the BIOS and (often) LILO are critical to booting the computer, neither is a running process once Linux is fully booted, and neither ever was, strictly speaking, a Linux process. During the boot process, Linux starts `init`, from which all other processes are descended. Although `ps` allows you to examine the status of running processes, it holds no privileged position in the process hierarchy; it's just another process.

19. A, B. Sudden jumps in load average indicate that programs are making heavier demands on the CPU than is normal. This may be because of legitimate factors like users running more or more demanding programs, or it could mean that a program has locked itself into an unproductive loop. Memory use isn't reflected in the load average. A malfunctioning CPU is likely to manifest itself in system crashes, not a change in the load average.

20. C. Many servers use `SIGHUP` as a code to reread their configuration files; this signal doesn't normally terminate the process. `SIGTERM` is a polite way to stop a process; it lets the process control its own shutdown, including closing open files. `SIGKILL` is a more forceful method of termination; it's more likely to work than `SIGTERM`, but open files won't be saved. There is no `SIGDIE` signal.

Chapter

8

Hardware Issues

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **3.7 Identify when swap space needs to be increased.**
- ✓ **3.8 Add and configure printers.**
- ✓ **3.9 Install and configure add-in hardware (e.g., monitors, modems, network interfaces, scanners).**
- ✓ **3.13 Load, remove, and edit list modules (e.g., `insmod`, `rmmod`, `lsmod`, `modprobe`).**
- ✓ **4.15 Manage print spools and queues.**
- ✓ **7.6 Remove and replace hardware and accessories (e.g., cables and components) based on symptoms of a problem by identifying basic procedures for adding and removing field replaceable components.**
- ✓ **7.7 Remove and replace hardware and accessories (e.g., cables and components) based on symptoms of a problem by identifying common symptoms and problems associated with each component and how to troubleshoot and isolate the problems.**
- ✓ **7.9 Identify proper procedures for diagnosing and troubleshooting ATA devices.**
- ✓ **7.10 Identify proper procedures for diagnosing and troubleshooting SCSI devices.**
- ✓ **7.11 Identify proper procedures for diagnosing and troubleshooting peripheral devices.**
- ✓ **7.12 Identify proper procedures for diagnosing and troubleshooting core system hardware.**
- ✓ **7.13 Identify and maintain mobile system hardware (e.g., PCMCIA, APM).**

Most Linux distributions can detect and configure themselves to properly use your software at system installation. In fact, distributions increasingly include the facility to do this even after installation, through tools like Red Hat's Kudzu and Mandrake's HardDrake. Sometimes, though, you need to manually configure new hardware or tweak an automatic configuration. This chapter covers this matter, with particular emphasis devoted to a few hardware issues that deserve extra attention: swap space, printing, and portable computing. Another area that deserves special attention is configuring the X Window System; this topic is covered in Chapter 4, "Users and Security." Whatever the hardware, troubleshooting that hardware can sometimes be a tricky task, so this chapter covers that topic, as well.

Adding Swap Space

Linux allows you to run programs that consume more memory than you have RAM in your system. It does this through the use of *swap space*, which is disk space that Linux treats as an extension of RAM. When your RAM fills with programs and their data, Linux moves some of this information to its swap space, freeing actual RAM for other uses. This feature, which is common on modern operating systems, is very convenient when your users run an unusually large number of programs. If relied on too much, though, performance suffers because disk accesses are far slower than are RAM accesses. It's also important that you have adequate swap space on your system. If the computer runs out of swap space, programs may begin to behave erratically.

Evaluating Swap Space Use

An invaluable tool in checking your system's memory use is `free`. This program displays information on your computer's total memory use. Its syntax is as follows:

```
free [-b | -k | -m] [-o] [-s delay] [-t] [-V]
```

The following is a list of the options to this command:

`-b | -k | -m` These options specify output values to be displayed in bytes, kilobytes, or megabytes, respectively. If you don't include any of these parameters, `free` uses kilobytes by default.

`-o` By default, `free` presents a correction for memory used by disk caches and buffers, as described shortly. Adding this parameter omits this correction.

`-s delay` This option causes `free` to display a memory-use report every *`delay`* seconds. The default is to show a single report and then quit.

`-t` This option adds a line to the output that includes totals (RAM plus swap space).

`-V` This option causes `free` to display its version number and then quit.

Listing 8.1 shows a sample output from `free` on a system with 128MB of RAM. (The total memory reported is less than 128MB because of memory consumed by the kernel and inefficiencies in the *x*86 architecture.)

Listing 8.1: Sample Output from `free`

```
$ free
             total       used       free     shared    buffers     cached
Mem:        127592     125320       2272     149524       2736      77068
-/+ buffers/cache:      45516      82076
Swap:       136512      12292     124220
```

The `Mem` line shows the total RAM used by programs, data, buffers, and caches. Unless you need information on memory used by buffers or caches, this line isn't very useful. The next line, `-/+ buffers/cache`, shows the total RAM use *without* considering buffers and caches. This line can be very informative in evaluating your system's overall RAM requirements, and hence in determining when it makes sense to add RAM. Specifically, if the `used` column routinely shows values that approach your total installed RAM (or

alternatively, if the `free` column routinely approaches 0), then it's time to add RAM. This information isn't terribly helpful in planning your swap space use, though.

The final row shows swap space use. In the case of Listing 8.1, 136,512KB of swap space is available. Of that, 12,292KB is in use, leaving 124,220KB free. Given the small amount of swap space used (about 9 percent), it seems that the system depicted in Listing 8.1 has plenty of swap space.

A single run of `free`, however, provides just a snapshot of a dynamic system. You can use the `-s` option to get a running look at memory use, but this requires either sitting around to watch the data as it rolls past on your screen, or using the redirection operator (>) to send the output into a file that you can study later, as in **`free -s 600 > freemem.txt`**. (Chapter 9, "Troubleshooting," covers redirection operators in more detail.) Alternatively, you can periodically use `free`, particularly at times when the system seems slow. If the available swap space shrinks to small values, you may want to boost available swap space, as described in the next two sections.

Real World Scenario

When to Add Swap, When to Add RAM

Swap space exists because hard disks are less expensive than RAM, on a per-megabyte basis. With the price of both falling, however, it's often wise to forego expanding your swap space in favor of adding extra RAM. RAM is faster than swap space, so all other things being equal, RAM is better.

A general rule of thumb derived from the days of Unix mainframes far less powerful than today's *x*86 boxes is that swap space should be 1.5–2 times as large as physical RAM. For instance, a system with 256MB of RAM should have 384–512MB of swap space. With 2.2.*x* kernels, it's often more helpful to look at this as a *maximum* for swap space. If your swap space use regularly exceeds 1.5–2 times your RAM size, your overall system performance will very likely be severely degraded. Adding RAM to such a system will almost certainly improve its performance. It won't hurt to have extra swap space, though, aside from the fact that this reduces the disk space available for programs and data files. The 2.4.*x* kernels have changed how swap space is managed, so 2.4.*x* kernels use more swap space than 2.2.*x* kernels do when they are running the same programs. For this reason, you should ensure that a system using a 2.4.*x* kernel has at least twice as much swap space as physical RAM.

Adding a Swap File

One method of adding swap space is to create a *swap file*. This is an ordinary disk file that's configured to be used by Linux as swap space. To add a swap file, follow these steps:

1. Create an empty file of the appropriate size. You can do this by copying bytes from `/dev/zero` (a device file that returns bytes containing the value 0) using the `dd` utility. `dd` takes parameters of `bs` (block size, in bytes) and `count` (the number of blocks to copy); the total file size is the product of these two values. You specify the input file with `if` and the output file with `of`. For instance, the following command creates a file called `/swap.swp` that's 134,217,728 bytes (128MB) in size:

   ```
   # dd if=/dev/zero of=/swap.swp bs=1024 count=131072
   ```

Swap space can reside on most Linux filesystem types. Swap space may *not* reside on Network Filesystem (NFS) mounts, though. If you try creating a swap file and the `swapon` command in step 3 doesn't work, this could be the problem. The Linux ext2 and VFAT filesystem drivers can both support swap space, as can the new Reiser filesystem. These are the filesystems you're most likely to want to use for this purpose.

2. Use the `mkswap` command to initialize the swap file for use. This command writes data structures to the file to allow Linux to swap memory to disk, but `mkswap` does *not* activate the swap file. For instance, the following command does this job:

   ```
   # mkswap /swap.swp
   ```

3. Use the `swapon` command to begin using the newly initialized swap space:

   ```
   # swapon /swap.swp
   ```

If you use `free` before and after performing these steps, you should see the total swap space count increase, reflecting the addition of the new swap space. If you want to make your use of this swap file permanent, you must add an entry to `/etc/fstab` (described in Chapter 6, "Managing Files and Services"). This entry should resemble the following:

```
/swap.swp   swap   swap   defaults   0 0
```

One key point is to list the complete path to the swap file in the first column, including the leading /. Once this entry is added, the system will use the swap file after you reboot.

To deactivate use of swap space, use the `swapoff` command, thus:

```
# swapoff /swap.swp
```

This command may take some time to execute if the swap file has been used much because the system takes time to read data from the disk for storage in memory or in other swap areas. The `swapon` and `swapoff` commands are actually the same program on most systems; this program does different things depending upon the name you use to call it.

Adding swap space in the form of a swap file can be a convenient way to add swap space quickly; however, this approach does have certain problems. Most importantly, if you create a large swap file on a partition that's already been heavily used, it's likely that the swap file will be *fragmented*—that is, that the file's contents will be spread across multiple groups of sectors on the disk. Fragmentation of disk files slows performance, and this can be a major problem in a swap file. The ability to quickly add a temporary swap file makes this method appealing in many cases, though. Indeed, the difficulty of repartitioning, as described shortly, makes adjusting swap partitions a task you may not want to undertake unless you're already planning to perform other system maintenance.

Adding a Swap Partition

Traditionally, Unix and Linux have used *swap partitions* for swap space. These are entire disk partitions devoted to swap space. In fact, some distributions won't install unless you create at least one swap partition. Therefore, chances are good you already have such a partition configured.

If you want to install multiple Linux distributions on one computer, they may share a single swap partition.

What if your existing swap partition is too small, though? The easiest approach in this case is usually to create a supplementary swap file, as described earlier. If you like, though, you can create a new swap partition. This approach works best if you're adding a hard disk or want to repartition

the disk for some other reason. In this case, you'll be adjusting your partition layout anyway, so you might as well take the opportunity to add new swap space. The basic procedure for doing this is as follows:

1. Clear space for the swap partition. This can be done by deleting existing partitions or by using a previously unused hard disk.
2. Create a new partition and give it a type code of 0x82 ("Linux swap"). Many OSs (but not Linux) use type codes to help them identify their partitions. Type codes 0x82 and 0x83 stand for Linux swap and filesystem partitions, respectively. The main reason to use these codes is to keep other OSs from damaging the Linux partitions.
3. When you're done partitioning or repartitioning, use `mkswap` to prepare the swap partition to be swap space. This operation works just like using `mkswap` on a file, except that you apply it to a partition, thus:

   ```
   # mkswap /dev/sdc6
   ```

4. Once the swap space has been prepared for use, you can add it manually using the `swapon` command described above, but you'll need to specify the swap partition's device rather than a swap file. For instance, you might use the following to access a swap partition on `/dev/sdc6`:

   ```
   # swapon /dev/sdc6
   ```

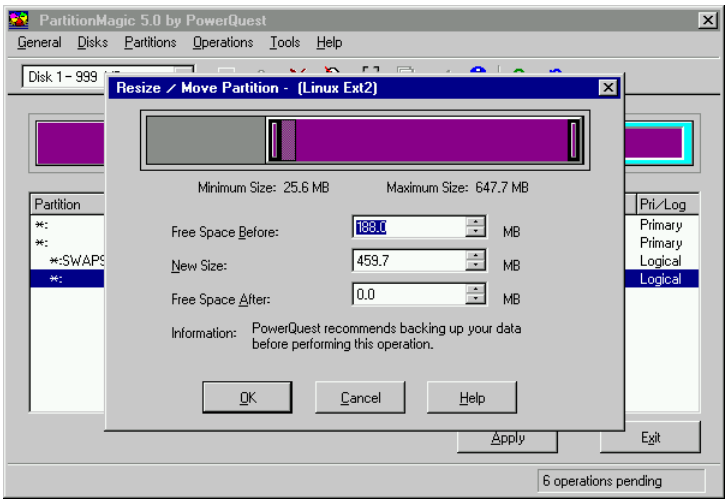
5. To use the swap partition permanently, add an entry for it to `/etc/fstab`, as described earlier in reference to swap files.

This procedure glosses over several critically important details concerning partition management. For one thing, when you modify an existing disk's partitions, you may need to adjust the device filenames for Linux filesystems in `/etc/fstab`. You'll have to do this either from an emergency boot or *before* you make the changes to the disk. It is at least as important, if you delete any existing partitions, to back up their contents before you delete the partition, even if you intend to re-create the partition with a smaller size. You may also need to reinstall the LILO boot loader if you modify your boot partition. In any event, this procedure will require the use of a disk partitioning tool such as Linux's `fdisk` (described in Chapter 7, "Managing Partitions and Processes.").

When you are resizing existing partitions, dynamic partition resizing tools can be very helpful. One of these that ships with some Linux distributions is `resize2fs`, which resizes an ext2 filesystem. This particular tool is a bit tedious to use, however, and it requires `fdisk` to complete the process. The experimental advanced journaling filesystems, such as ReiserFS, also at least theoretically support partition-resizing tools, but in mid-2001 they're still a bit less reliable, if they work at all.

Another option for ext2fs partitions is to use the commercial PartitionMagic from PowerQuest (`http://www.powerquest.com`). This program comes in DOS and Windows versions, including a DOS boot floppy so that you can run it from a floppy disk on a Linux-only system. PartitionMagic is particularly useful because it provides a GUI point-and-click interface, shown in Figure 8.1. You might still have to adjust your `/etc/fstab` entries and reinstall LILO, and backing up your data before performing such an operation is always wise. Nonetheless, dynamic partition resizers can greatly simplify reconfiguring swap files.

FIGURE 8.1 The PartitionMagic partition resizer can resize Linux ext2 and swap partitions.



You may also occasionally want to *shrink* a swap partition. This action is most likely when you find that you've accidentally created a ludicrously large swap partition. You may use dynamic partition resizers to adjust the size of your partitions in this case, or you can delete the swap partition and

create two or more in its place: a smaller swap partition and one or more partitions to be used for data storage. You can use these additional data partitions for any purpose to which you ordinarily put data partitions.

Basic Printing

Printing in Linux is a cooperative effort of several different tools. A system administrator must be familiar with what each of the tools in this collection does, as well as how they interact. As with many other programs that are part of Linux, there are several different versions of some of these tools, which can lead to confusion or incompatibilities if you're not aware of how the system as a whole functions.

The Linux Printing Architecture

Linux printing is built around the concept of a *print queue*. This is a sort of holding area where files wait to be printed. A single computer can support many distinct print queues. These frequently correspond to different physical printers, but it's also possible to configure several queues to print in different ways to the same printer. For instance, you might use one queue to print single-sided and another queue for double-sided printing on a printer that supports duplexing.

Users submit print jobs by using a program called `lpr`. Users can call this program directly, or they may let another program call it. In either case, `lpr` sends the print job into a specified queue. This queue corresponds to a directory on the hard disk, typically in a subdirectory of the `/var/spool/lpd` directory. This directory is named after the `lpd` program, which is at the core of Linux's printing system. This program is a *daemon* (its name stands for "line printer daemon"), which means that it runs in the background watching for certain events that will trigger it to spring into action. `lpd` accepts print jobs from `lpr` or from remote computers, monitors print queues, and serves as a sort of "traffic cop," directing print jobs in an orderly fashion from print queues to printers. To do all of this, `lpd` relies upon the `/etc/printcap` configuration file, which is described in more detail shortly.

One important and unusual characteristic of Linux printing is that it's highly network-oriented. As just noted, `lpd` can accept print jobs that originate from remote systems as well as from local ones. In fact, even local print

jobs are submitted via network protocols, although they don't normally use network hardware, so even a computer with no network connections can print. In addition to being a server for print jobs, `lpd` can function as a client, passing print jobs on to other computers that run the same protocols.

One of the deficiencies of the Linux printing system is that it's essentially unidirectional—print jobs originate in an application, which blindly produces PostScript (as described shortly) without knowing anything about the printer to which it's printing. The print queue takes this output and sends it on to the printer, which must deal with it as best it can. There's no way for a Linux application to directly query a printer concerning its capabilities, such as whether it supports multiple paper trays or wide forms.

One confusing aspect of Linux printing is that Linux supports several competing printing systems. The two most popular are the original Berkeley Standard Distribution (BSD) printing system and the newer LPRng package. Both work according to the outline just presented, but they differ in some details, some of which are discussed in the upcoming sections. An alternative to both of these is the new Common Unix Printing System (CUPS), which works in a similar manner in broad outline, but more of its details differ. For instance, CUPS doesn't use `lpd` or `/etc/printcap`; instead, it uses `cupsd` and various files in `/etc/cups` for configuration. CUPS supports advanced features designed to give applications more information on the printers to which they print, but in 2001, it's a less popular printing system than BSD or LPRng.

Using PostScript and Ghostscript

If you've configured printers under Windows, MacOS, OS/2, or certain other OSs, you're probably familiar with the concept of a *printer driver*. In these OSs, the printer driver stands between the application and the printer queue. In Linux, the printer driver is part of Ghostscript (`http://www.cs.wisc.edu/~ghost`), which exists as part of the printer queue, albeit a late part. This relationship can be confusing at times, particularly because not all applications or printers need Ghostscript.

PostScript: The De Facto Linux Printer Language

Laser printers as we know them today began to become popular in the 1980s. The first laser printers were very expensive devices, and many of them supported what was at that time a new and powerful printer language: *PostScript*. PostScript printers became quite popular as accessories for the Unix

systems of the day. Unix print queues were not designed with Windows-style printer drivers in mind, so Unix programs that took advantage of laser printer features were typically written to produce PostScript output directly. As a result, PostScript developed into the de facto printing standard for Unix and, by inheritance, Linux. Where programs on Windows systems were built to interface with the Windows printer driver, similar programs on Linux generate PostScript and send the result to the Linux printer queue.

A few programs violate this standard. Most commonly, many programs can produce raw text output. Such output seldom poses a major problem for modern printers, although some PostScript-only models choke on raw text. Some other programs can produce either PostScript or *Printer Control Language (PCL)* output for Hewlett-Packard laser printers or their many imitators. A very few programs can generate output that's directly accepted by other types of printers.

The problem with PostScript as a standard is that it's uncommon on the low- and mid-priced printers with which Linux is often paired. Therefore, to print to such printers using traditional Unix programs that generate PostScript output, you need a translator, and a way to fit that translator into the print queue. This is where Ghostscript fits into the picture.

Ghostscript: A PostScript Translator

When it uses a traditional PostScript printer, a computer sends a PostScript file directly to the printer. PostScript is a programming language, albeit one that's oriented towards the goal of producing a printed page as output. As a result, a PostScript printer needs a fair amount of RAM and CPU power. In fact, in the 1980s it wasn't uncommon for PostScript printers to have more RAM and faster CPUs than the computers to which they were connected. Today, though, printers frequently have little RAM and anemic CPUs—particularly on inexpensive inkjet models.

Ghostscript is a PostScript interpreter that runs on a computer, offloading some of the need for RAM and CPU power. It takes PostScript input, parses it, and produces output in any of dozens of different bitmap formats, including formats that can be accepted by many non-PostScript printers. This makes Ghostscript a way to turn many inexpensive printers into Linux-compatible PostScript printers at very low cost. (Ghostscript is available as open source software, with a more advanced variant available for free, but it is not freely redistributable in any commercial package. Because all Linux distributions are available on CD-ROMs sold for a price, they therefore ship with the older GNU Ghostscript, which works well enough for most users.)

One of Ghostscript's drawbacks is that it produces large output files. A PostScript file that produces a page filled with text may be just a few kilobytes in size. If this page is to be printed on a 600 dots per inch (dpi) printer using Ghostscript, the resulting output file could be as large as 4MB—assuming it's black and white. If the page includes color, the size could be much larger. In some sense, this is unimportant because these big files will only be stored on your hard disk for brief periods of time. They do still have to get from the computer to the printer, though, and this process can be slow. Also, some printers (particularly laser printers) may require memory expansion to operate reliably under Linux.

Real World Scenario

Choosing an Appropriate Printer for Linux

If you want a speedy printer for Linux, choose a model with built-in PostScript. This is particularly true for textual and line-art output, which suffers the most in terms of size expansion going from PostScript to bitmap. In my experience, Ghostscript-driven printers work well enough for 600dpi black-and-white printers with speeds of up to about 6 pages per minute (ppm). If the printer's speed is greater than that, the parallel or USB port may not be able to deliver the necessary performance, although you may be able to tweak it to get somewhat better speed.

Color inkjet printers are generally limited more by the speed of the print head than by the speed of the data coming over their ports. Few such printers directly support PostScript, either. Some models come with Windows-based PostScript engines that are conceptually similar to Ghostscript, but such software is useless under Linux. There are a few PostScript inkjets on the market, as well as color PostScript printers that use other printing technologies.

For information on what printers are supported by Ghostscript, check the Ghostscript Web page or the GNU/Linux Printing Web page (`http://www.linuxprinting.org/printer_list.cgi`).

Squeezing Ghostscript into the Queue

Printing to a non-PostScript printer in Linux requires fitting Ghostscript into the print queue. This is generally done through the use of a *smart filter*. This is a program that's called as part of the printing process. The smart filter examines the file that's being printed, determines its type, and passes the file through one or more additional programs before `lpd` sends it on to the printer. The smart filter can be configured to call Ghostscript with whatever parameters are appropriate to produce output for the queue's printer.

The smart filter is specified with the `if` field in `/etc/printcap`, as described shortly. There are several smart filter packages available for Linux, including `rhs-printfilters` (used in Red Hat and some of its derivatives), APSFilter (used in several other distributions), and magicfilter.

Configuration of the smart filter can be tricky, but most distributions include setup tools that help immensely. The upcoming section, "Using a GUI Configuration Tool," describes the use of one such tool. I highly recommend that you use such programs when configuring your system to print.

The end result of a typical Linux printer queue configuration is the ability to treat any supported printer as if it were a PostScript printer. Applications that produce PostScript output can print directly to the queue. The smart filter detects that the output is PostScript and runs it through Ghostscript. The smart filter can also detect other file types, such as plain text and various graphics files, and it can send them through appropriate programs instead of or in addition to Ghostscript, in order to create a reasonable printout.

If you have a printer that can process PostScript itself, the smart filter is usually still involved, but it doesn't pass PostScript through Ghostscript. In this case, the smart filter passes PostScript directly to the printer, but it still sends other file types through whatever processing is necessary to turn them into PostScript.

BSD and LPRng Configuration and Use

Fortunately, basic printer configuration for both the original BSD printing tools and LPRng is similar. You can configure everything by hand by directly editing configuration files, but certain critical details—namely, how your smart filter is set up—differ from one distribution to another, and they can be tedious to track down. Therefore, direct file editing is best reserved for cases where you can forego the smart filter or if you're willing to track down the documentation for whatever smart filter your system uses. In most cases,

it's easier to use a GUI configuration tool to do the initial printer configuration, and then you can tweak that configuration by hand, if necessary. Either way, the daemon runs in the background and accepts print jobs submitted via `lpr`.

Configuring the */etc/printcap* File

The `/etc/printcap` file is at the heart of both the BSD and LPRng printing systems. Listing 8.2 illustrates the format of `/etc/printcap` by showing an entry for a single printer. You can define multiple printers in `/etc/printcap`, though; just be sure to use different names.

Listing 8.2: A Sample `/etc/printcap` File

```
lp|hp4000:\
        :lp=/dev/lp0:\
        :br#57600:\
        :rm=:\
        :rp=:\
        :sd=/var/spool/lpd/lp:\
        :mx#0:\
        :sh:\
        :if=/var/spool/lpd/lp/printfilter:
```

Technically, each printer definition is one line long. `/etc/printcap` entries, however, traditionally make heavy use of the common Linux convention of using a backslash (`\`) to signal a line continuation. (Note that every line in Listing 8.2 *except* the last one ends in a backslash.) This makes the printer definitions easier to read. Each component within the `/etc/printcap` entry is separated from the others by colons (`:`). Common components of a print queue definition include the following:

Printer name Each printer definition begins with one or more names for the printer. If the printer has multiple names, they're separated from each other by vertical bars (`|`). Traditionally, the default printer is called `lp`. Listing 8.2's example expands on this by adding the name `hp4000`. Users may print using either name with the same results.

`lp` This entry defines the device filename for the printer. In the case of Listing 8.2, the printer device is `/dev/lp0`, which corresponds to the first parallel port. Many modern printers support USB interfaces, which use

the `/dev/usb/lp`*n* devices, where *n* is a number from 0 up. A few printers use the old RS-232 serial ports, which may be accessed as `/dev/ttyS`*n*. This entry may be omitted if the printer is shared from another computer.

`br` `br` stands for *baud rate*; it defines the communications rate for RS-232 serial printers. This option is normally omitted for parallel-port, USB, and network printers. It doesn't do any harm to leave it in, however, as in Listing 8.2.

`rm` If you're defining a printer queue for a printer that's connected to another computer or that's connected directly to the network, you specify its machine name with the `rm` option. Like `br`, it can be omitted if not used.

`rp` This option is used in conjunction with `rm`, but it specifies the name of the print queue on the remote system. For instance, your local `epson` queue might print to a queue called `inkjet` on a remote system. Your local users will use the name `epson`, and your `lpd` will pass the job on to the remote system's `lpd`, which will print to the remote `inkjet` queue. This option may be omitted if the printer is local, but leaving it blank in this case (as in Listing 8.2) does no harm.

`sd` `sd` stands for *spool directory*. This is the location of the print queue on your hard disk. By convention, this is a subdirectory of the `/var/spool/lpd` directory, named after the print queue's primary name. If you create print queues by hand, you'll need to create this directory. It should normally have fairly restrictive permissions (such as `rwx------` and ownership by `root`; see Chapter 4) so that people can't read or delete each other's print jobs.

`mx` The `mx` option sets the maximum size of a print job, in bytes. You can use this option to restrict abuses, but be aware that the print job size in bytes and its size in pages are poorly correlated. If it is set to `0`, there's no limit on job size. This option uses a pound sign (`#`) rather than an equal sign (`=`) to set its value.

`sh` The `sh` option takes no value. It stands for suppress header, and if it's present, Linux does *not* print a header page with information on the user who printed the job. This configuration makes sense for workstations, but on multiuser systems and print servers, omitting the `sh` option and using the resultant headers can help you organize printouts from multiple users.

`if` This option sets the input filter filename, which is part of the smart filter associated with the queue. This is frequently a script located within the spool directory. The script sets various options (such as the name of the Ghostscript driver to be used) and calls the smart filter files located elsewhere on the disk.

`/etc/printcap` is a very complex file that supports many options. You can learn about more of them from the file's man page (type **`man printcap`**). The preceding options cover what you're likely to do with the queue, however, aside from smart filter configuration.

After reconfiguring your print queues, you may need to restart your printer daemon. On most systems, you can do this by passing the `restart` parameter to the LPRng or BSD printing startup script (startup scripts are described in Chapter 6). The following is an example of how this might be done:

```
# /etc/rc.d/init.d/lpd restart
```

The exact name and location of this file will vary from one distribution to another. You should use this command only when your system isn't actively printing.

Using a GUI Configuration Tool

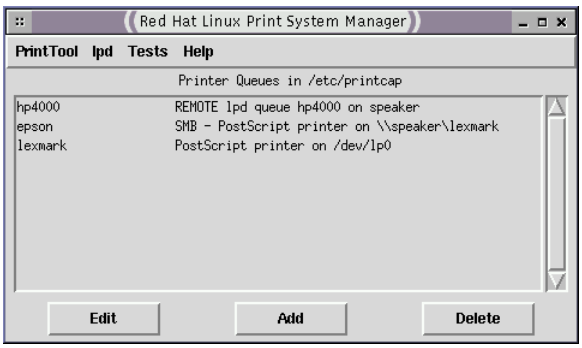
Much of the challenge of printing in Linux doesn't come from the `/etc/printcap` file; it comes from telling the system about your printer—that is, smart filter and Ghostscript configuration. GNU Ghostscript comes standard with all major Linux distributions, and it is probably adequate for your system. In a few cases, though, you may need to upgrade to the more recent Aladdin Ghostscript or obtain a version with some unusual drivers compiled into it. The GNU/Linux Printing Web page (`http://www.linuxprinting.org/printer_list.cgi`) can be an extremely useful resource in tracking down appropriate drivers.

In most cases, the easiest way to configure a print queue with a smart filter for a specific non-PostScript printer is to use a GUI printer configuration tool. Most major Linux distributions come with these. For instance, Caldera's COAS, SuSE's YaST and YaST2, and the `printtool` that comes with Red Hat and some of its derivatives all help step you through the printer setup process. The details of these systems differ, but they must perform the

same tasks, so they're similar in outline. As an example of these, consider the use of `printtool`. To use this program, follow these steps:

1. Type **`printtool`** as `root` from an `xterm` or similar X-based command prompt window. The result is the main `printtool` window, shown in Figure 8.2. (Figure 8.2 shows three queues preconfigured, but the first time you launch this program, the queue list will probably be empty.)

FIGURE 8.2 The main `printtool` window shows you the available print queues and allows you to perform tests of these queues.

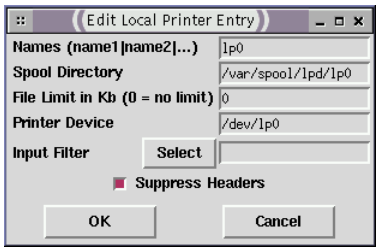


2. Click Add to add a printer. `printtool` displays a small dialog box asking what type of printer you want to add. Options include a local printer, a remote Unix (`lpd`) queue, a Windows print queue, a NetWare print queue, and direct to port (bypassing `lpd`). This example follows the first option. Remote printer configuration is similar, but you enter different identifying information for the printer, such as the remote host and print queue names.

3. If you opted to create a local print queue, the system shows the information on the printer hardware that it has detected. In my experience, `printtool` fails to detect working hardware rather frequently, so don't be alarmed if you don't see a device that you know is working.

4. After you dismiss the dialog box informing you of available printer devices, `printtool` displays the Edit Local Printer Entry dialog box shown in Figure 8.3. This dialog box allows you to enter critical `/etc/printcap` information, such as the printer names, spool directory name, and printer device filename. Once you've entered this information, though, *do not* click OK.

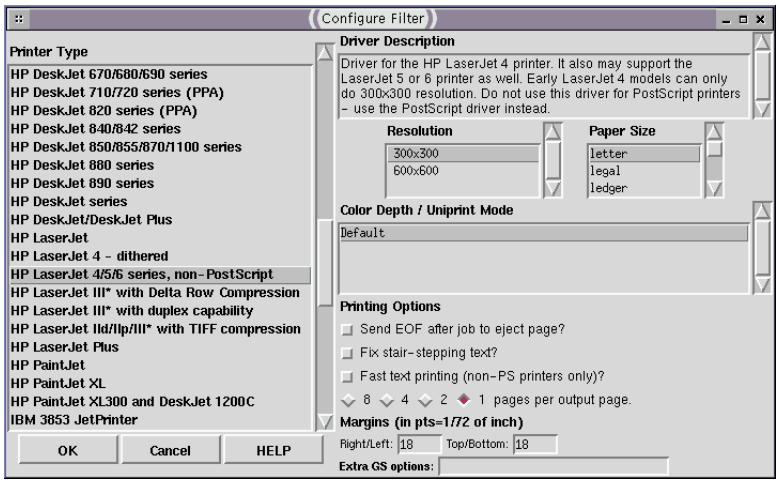
FIGURE 8.3 The Edit Local Printer Entry dialog box provides an uninspired but usable default name for your queue.



5. Click Select in the Edit Local Printer Entry dialog box. This brings up the Configure Filter dialog box (Figure 8.4). It's here that you select the Ghostscript driver to be used by the queue, along with other options:

   a. The Printer Type field on the left of the Configure Filter dialog box shows a list of printer types. Note that this list does *not* include most printer models because most printers use command sets pioneered by other models. For instance, if you've got *any* printer that's compatible with the HP LaserJet 4/5/6 series, you should select that printer, as shown in Figure 8.4. There's a single entry in this list for *all* PostScript printers, so if your printer understands PostScript, select PostScript Printer from this list.

FIGURE 8.4 The Configure Filter dialog box lets you specify a printer driver and other printing options for a print queue.



b. The Resolution and Paper Size fields let you set the printer's resolution and paper size, as you might imagine. Some drivers don't use these options, however; instead, they push these features into the Color Depth/Uniprint Mode field, which allows you to set options related to the Uniprint Ghostscript driver. (This driver is used by many inkjet printers.)

c. The Printing Options area lets you set a few miscellaneous options. The Send EOF after Job to Eject Page option is necessary on a few printers that don't otherwise print or eject the last page. Some printers, when fed Linux-style text files, print each line horizontally *after* the preceding one. Fix Stair-Stepping Text fixes this effect. Fast Text Printing (Non–PS Printers Only) sends text files to the printer as text, rather than converting to PostScript first. This speeds up plain text printing on many non-PostScript printers. The Pages per Output Page option lets you set up a queue that compresses multiple pages into a single page to save paper. If your outputs are off-center, you can adjust the margins using the Margins fields. Finally, Extra GS Options lets you send additional options to Ghostscript.

6. After setting all the options you require in Configure Filter, click OK in it and in the Edit Local Printer Entry dialog box. You'll see your new queue appear in the main `printtool` window (Figure 8.2).

7. Select lpd ➢ Restart lpd to restart the printer daemon and have it recognize the new print queues.

8. There are three tests you can run from the `printtool` window (from the Tests menu):

Print ASCII Test Page This option prints a plain text test page to the printer. You should see a page filled with information. If you see just a couple of lines, you may need to go back and adjust the stair-stepping option (step 5c).

Print PostScript Test Page This option works like the preceding one, but it prints a PostScript test page. This gives your printer's PostScript interpreter a basic workout, or it tests your Ghostscript configuration. If you get pages of gibberish, chances are you selected the wrong printer in step 5a.

Print ASCII Directly to Port If you have problems performing both of the preceding tests, try this one. If it doesn't work, either, there's probably something wrong with your hardware configuration. You may have specified the wrong device (step 4), or you may need to load a kernel module, as described in "Managing Kernel Modules" later in this chapter. If this test works, then the problem lies in some other aspect of your configuration, such as your selection of a printer model (step 5a).

9. When the printer is working as expected, select PrintTool ➢ Quit to exit from the utility.

You can create several print queues, even if you have just one printer. For instance, you might create one queue that prints normally, and another that prints two input pages per output page (step 5c). Your users can then use this option by choosing a different print queue, as described shortly—for instance, you might call one `okidata` and the other `okidata2up`. Similarly, you can create separate queues for different printer resolutions. You can even create one queue that prints with the normal print filters and another that doesn't use a filter—that is, a raw queue. A raw print queue can be useful if you have programs that can print directly to the printer type you use. For instance, WordPerfect 8.0 (but not WordPerfect 9.0) and The GIMP include drivers for many specific printer models, and so they can do without Ghostscript in many cases. To create a raw queue, omit all of step 5, or edit `/etc/printcap` afterward and remove the filename referenced on the `if` line.

Printing Files with *lpr*

Once you've configured the system to print, you probably want to do so. As mentioned earlier, Linux uses the `lpr` program to submit print jobs. This program accepts many options, the most useful of which are as follows:

`-P`*`queuename`* This option allows you to specify a print queue. This is useful if you have several printers or if you've defined several queues for one printer, as just described.

In the original BSD version of `lpr`, there should be no space between the `-P` and the *queuename*. LPRng is more flexible in this respect; you can insert a space or omit it, as you see fit.

`-r` Normally, `lpr` sends a copy of the file you print into the queue, leaving the original unharmed. With this option, `lpr` deletes the original file after printing it.

`-h` This option suppresses the banner for a single print job.

`-J` *`jobname`* Print jobs have names to help identify them, both while they're in the queue and once printed (if the queue is configured to print banner pages). The name is normally the name of the first file in the print job, but you can change it by including the `-J` option.

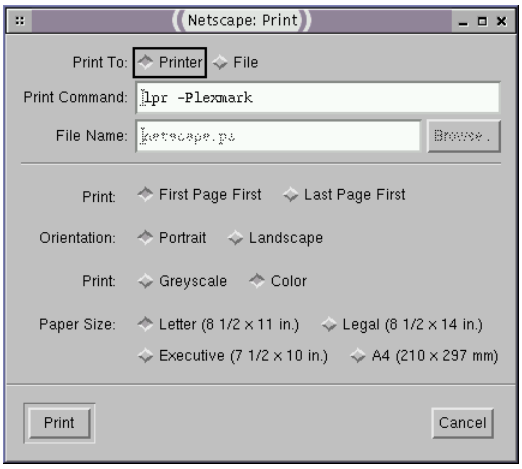
`-m` *`username`* This option causes `lpd` to send e-mail to *`username`* when the print job is complete.

Suppose you have a file called `report.txt` that you want to print to the printer attached to the `lexmark` queue. This queue is often quite busy, so you want the system to send e-mail to `ljones` when it's done so that you know when to pick up the printout. You could use the following command to accomplish this task:

```
$ lpr -Plexmark -m ljones report.txt
```

The `lpr` command is accessible to ordinary users as well as to `root`, so anybody may print using this command. It's also called from many programs that need to print directly, such as graphics programs and word processors. These programs typically give you some way to adjust the print command so that you can enter parameters such as the printer name. For instance, Figure 8.5 shows Netscape's Print dialog box. Note that the Print Command text entry field includes a complete `lpr` command, but without the filename. Some other programs hide the `lpr` command in a configuration tool and provide a selectable list of printers in their main Print dialog boxes. Consult the program's documentation if you're not sure how it works.

FIGURE 8.5 Most Linux programs that can print do so by using lpr, and they generally let you edit the exact command used to print.



Monitoring and Controlling the Print Queue

There are several utilities that can be used to examine and manipulate a Linux print queue. These utilities are `lpq`, `lprm`, and `lpc`. All of these commands can take the `-P` parameter to specify that they operate on a specific print queue.

lpq

`lpq` displays information on the print queue—how many files it contains, how large they are, who their owners are, and so on. By entering the user's name as an argument, you can also use this command to check on any print jobs owned by a particular user. To use `lpq` to examine a queue, you might use a command like the following:

```
$ lpq -Plexmark
Printer: lexmark@speaker
 Queue: 1 printable job
 Server: pid 14817 active
 Unspooler: pid 14822 active
```

```
 Status: printing 'rodsmith@speaker+787', file 1
'Insight.ps', size 672386, format 'l' at 14:57:10
 Rank   Owner/ID           Class Job  Files        Size
↳Time
active  rodsmith@speaker+787  A  787 Insight.ps  672386
↳14:56:22
```

This example shows the output of LPRng's `lpq`. Systems that use the original BSD printing system display less information, but the most important information (such as the job number, job owner, job filename, and job size) is present in both cases.

Of particular interest is the job number—787 in the case of this example. You can use this number to delete a job from the queue or reorder it so that it prints before other jobs. Any user may use the `lpq` command.

lprm

The `lprm` command removes one or more jobs from the print queue. There are several different ways to issue this command:

- If it's issued with a number, that number is understood to be the job ID (as shown in `lpq`'s output) that's to be deleted.
- If `root` runs `lprm` and passes a username to the program, it removes all the jobs belonging to that user.
- If a user runs the BSD `lprm` and passes a dash (`-`) to the program, it removes all the jobs belonging to the user. LPRng uses `all` instead of a dash for this purpose.
- If `root` runs the BSD `lprm` and passes a dash (`-`) to the program, it removes all print jobs belonging to all users. Again, LPRng uses `all` for this purpose.

This program may be run by `root` or by an ordinary user, but as just noted, its capabilities vary depending upon who runs it. Ordinary users may remove only their own jobs from the queue, but `root` may remove anybody's print jobs.

lpc

`lpc` starts, stops, and reorders jobs within print queues. `lpc` takes commands, some of which require additional parameters. You can pass the printer name with `-P`, as with other printer utilities, or you can pass this information *without* the `-P` parameter. In the latter case, the print queue name appears immediately after the command. The following are the most useful `lpc` commands:

`abort` Stops printing the current job and any other jobs in the queue but leaves those jobs intact. Subsequently issuing the `start` command will resume printing.

`disable` Sets the queue to reject further print jobs but does *not* halt printing of jobs currently in the queue.

`down` Stops printing to the queue and sets the queue to reject further print jobs.

`enable` Enables the queue so that it begins accepting print jobs again. This is the opposite of `disable`.

`exit` If `lpc` is started in interactive mode (described shortly), this command will exit from this mode.

`start` Begins printing and starts an `lpd` process for the queue.

`stop` Stops `lpd` so further printing is disabled after the current job completes.

`topq` *`jobid`* Moves the job whose ID is *`jobid`* to the start of the queue (after the currently printing document). Use this to reprioritize your print queue.

`up` Enables the specified print queue; the opposite of `down`.

You can run `lpc` in interactive mode, in which you issue one command after another, or you can have it execute a single command and then exit by specifying the command on the same command line you use to launch `lpc`. As an example, suppose you want to adjust the printing of job 945 on the `brother` queue (identified through a previous `lpq` command) so that it's next to print. You could issue the following command to do this:

```
# lpc topq brother 945
```

Although ordinary users may run `lpc`, for the most part, they can't do anything with it. Typical `lpc` operations require superuser privileges.

Adding New Hardware

Printers are an important class of hardware, but they're unusual in Linux for a variety of reasons. The location of printer drivers (in Ghostscript), the external nature of printers, and the presence of printer queues are all features that are unusual for hardware. Most hardware is configured in other ways.

When a computer runs Windows, adding hardware is, if not a trivial undertaking, something for which explicit support is almost always available. *x*86 hardware manufacturers ensure that their products have Windows 9*x*/Me and, increasingly, Windows 2000 drivers available. (The exceptions relate to highly specialized products.) Manufacturers also include explicit installation instructions with their products. In Linux, on the other hand, matters aren't so simple. Drivers for any given piece of hardware may or may not be available. When Linux drivers exist, they often do not come with the hardware product, so you may need to look to a third-party supplier to get your device working. Documentation on using hardware in Linux is quite variable in quality. This book cannot provide a complete guide to Linux hardware because such a guide would take an entire book. (My *Linux Hardware Handbook* [Sams Publishing, 2000] is such a guide.)This section provides some general guidelines to help you locate, install, and use drivers.

Locating Hardware Drivers

The Linux kernel serves as the interface between most hardware and software. Therefore, most hardware drivers either come as part of the kernel or are added to the kernel in one way or another. Because of the open source nature of Linux kernel development, most kernel drivers eventually find their way into the official Linux kernel source tree. There are exceptions to this rule, though. Some hardware—most notably printers, scanners, and video cards—are driven partly or wholly through non-kernel software. All told, there are several possible sources of hardware drivers for Linux:

Main kernel tree As just noted, the standard Linux kernel includes drivers for many devices, and for most classes of hardware, the kernel is the

first place to check. Most distributions compile most of these drivers, either directly into the kernel or as modules. It's possible you'll need to upgrade or recompile your kernel to get access to drivers for certain devices, though.

New kernel drivers As new devices reach the market, Linux developers write drivers for them. Initially, many of these projects aren't part of the official kernel tree. They may also be integrated into the unstable development kernel before the stable release kernel. (See Chapter 3, "Software Management," for a discussion of kernel versions.) Unfortunately, tracking down such new drivers can be tricky. Doing a Web search on the device name and "Linux" will often turn up pointers to the driver. You can also try a development kernel, but these are likely to be unstable.

Hardware manufacturers An increasing number of hardware manufacturers provide Linux drivers for their products. These are sometimes nothing more than the standard kernel drivers, but a few manufacturers make their own drivers available.

Ghostscript Printers under Linux rely on drivers in the Ghostscript package, described earlier in this chapter. `http://www.linuxprinting.org/printer_list.cgi` hosts information on drivers and compatibility of specific printers with Linux.

Software modems Most external modems are compatible with standard Linux serial port drivers. Most internal modems sold today, however, are *software modems*. These devices require special driver support, which is spotty under Linux. The LinModems Web site, `http://www.linmodems.org`, has information on driver developments for such modems.

USB devices Starting with the 2.2.18 and 2.4.*x* kernels, Linux includes good USB support. (Some distributions patched this support into earlier kernels, as well.) Supported USB devices include keyboards, mice, CDC-ACM modems, many scanners, printers, Zip drives, and more. Development for USB devices is still ongoing. Also, some devices, such as printers and scanners, require drivers in secondary packages, such as Ghostscript, in order to work. The Linux USB Project home page, `http://www.linux-usb.org`, has additional information on drivers for USB devices.

X servers XFree86 is the main X server for Linux. The XFree86 Web site (`http://www.xfree86.org`) can be an important resource when you

want to find information on Linux support for video cards. In a few cases, XFree86 won't have a driver, but one of two commercial X servers—Accelerated X (`http://www.xig.com`) or Metro-X (`http://www.metrolink.com`)—may have support for the card.

Linux can use any video card in text mode, and almost any card in basic video modes like 640 × 480 VGA. Support in XFree86 or another X server is only required if you intend to run X at higher resolutions. This is a virtual requirement for most workstations, but some servers can get by without this support.

Sound drivers The Linux kernel includes support for many sound cards, but this support is weak for some. Two alternative projects with improved support for at least some sound cards exist. One is the open source Advanced Linux Sound Architecture (ALSA; `http://www.alsa-project.org`) project. The other is the commercial Open Sound System (OSS; `http://www.4front-tech.com`) project. The standard 2.4.*x* and earlier kernel drivers are actually derived from a stripped-down version of the OSS drivers, but the ALSA drivers may be integrated into a 2.5.*x* or later kernel series.

Scanners Scanners in Linux require non-kernel drivers. These are provided by the open source Scanner Access Now Easy (SANE; `http://www.mostang.com/sane`) project. The commercial OCR Shop (`http://www.vividata.com/ocrshop.html`) package works without SANE.

In addition to drivers, some types of hardware require special user-level software. For instance, CD-R and CD-RW drives need special CD-R burning software, such as X-CD-Roast (`http://www.xcdroast.org`), and digital cameras need a package like gPhoto (`http://www.gphoto.org`). Such software uses low-level Linux drivers (such as SCSI or USB drivers in the kernel) to access the hardware, but the software provides the means to make the hardware work, possibly including drivers for specific devices. In fact, printer support works the same way; Ghostscript relies on low-level parallel, RS-232 serial, USB, or network drivers to get its output to the printer. Printers are more common and fundamental to a computer's function, however, so I've classified Ghostscript as a main Linux driver.

Configuring Hardware in Linux

Hardware configuration can range from trivially easy to extremely complex, depending upon the hardware—both its general type and the specific model you're using. There are several classes of task you must perform. Some hardware requires you to perform all these tasks, but other hardware allows you to omit one or more.

Proper Procedures for Replacing Hardware

The most basic task in adding or replacing hardware is in physically installing the device. There are three general classes of hardware, each of which requires different procedures: external devices, internal cabled devices (like hard disks and CD-ROM drives), and internal expansion cards. A few devices straddle the lines in some way.

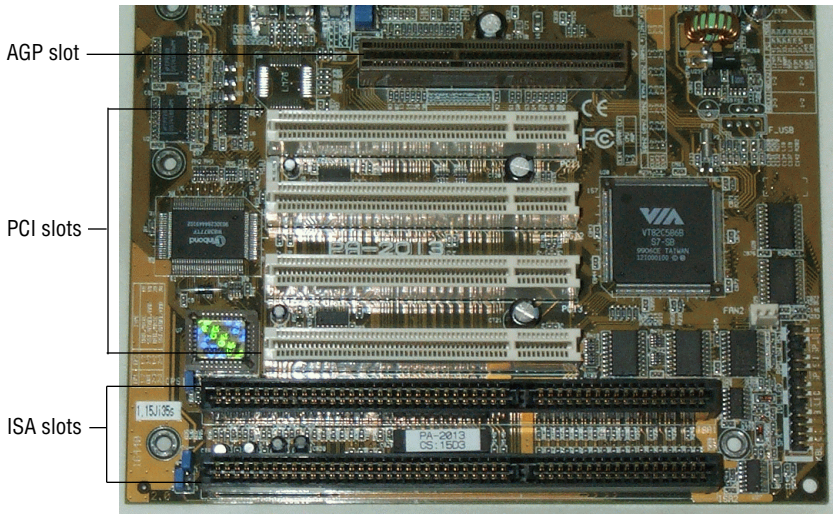
External devices are plugged into the computer through an external port, such as an RS-232 serial port, a parallel port, a SCSI port, or a USB port. Many, but not all, of these devices also require power from a wall outlet, so the device itself has at least two cables. External connectors are usually shaped in such a way that they can only be plugged in one way, so it's impossible to plug the device in backward. Most devices should only be attached when the computer's and the device's power are turned off. USB devices, however, can be safely attached and detached when the power is turned on.

Internal cabled devices typically attach to the motherboard or an expansion card. They usually require connecting at least two cables: a power cable and a data cable. Power cables are keyed to prevent accidental backward insertion, as are some data cables. Some data cables, though, lack this keying. If you must use such a cable, check for a colored stripe along one edge, and insert it to match up with the "pin 1" markings on both the device and the motherboard or card. You should always power off a computer before attaching internal cabled devices.

Internal cabled devices are usually mounted to the computer's case using screws. Details vary from one computer to another. Sometimes you may need to partially disassemble a system to reach the screws, which can be a nuisance. For instance, I've seen mini-tower systems that require you to partially remove the motherboard to reach screws for 3.5-inch hard disks. If you've removed screws but you can't remove a drive, look for other areas (even hidden ones) where additional screws might be.

Internal expansion cards plug into slots on the motherboard, as shown in Figure 8.6. Note that because of size and placement differences, it's impossible to insert the wrong type of card into a slot—for instance, a PCI card won't fit in an ISA slot. (If you run out of one type, you may be able to find a device in the slot type that's available to you, or you may have to remove one card to make room for another. Sometimes you can resort to using another type of device, like a USB-to-Ethernet adapter rather than an in-computer Ethernet card.) To insert a card, you *must* power off the computer. Failing to do so is virtually certain to result in damage to the card or the computer. After the power is off, align the card so that its connector matches that of the slot, and apply some force to push it into the expansion slot. You should then use a screw to secure the top of the card to the computer's case. Some components, like RAM and even the CPU, are installed in slots or sockets similar to those used by expansion cards. Their installation procedures are similar, but these devices don't attach in a way that makes them accessible from outside the computer.

FIGURE 8.6 Expansion slots come in several different and incompatible varieties.



You've probably had the experience of walking across a carpeted room and receiving an electrostatic shock when you touch a door knob. This phenomenon is known as an *electrostatic discharge (ESD)*. The same thing can happen when you work on computer hardware, but such an event can actually damage the computer or component. For this reason, it's important that

you ground yourself when working on computer hardware—either a complete computer or individual components. The best way to do this is with an ESD wrist strap. This is a strap that resembles a bracelet or hospital ID tag, but it connects to a wall electrical outlet's ground prong. The result is that your body is electrically grounded, so any static you might build up is discharged, literally into the ground, before it can do any harm to a computer. If you lack an ESD strap, you should at least not move around on carpet wearing rubber-soled shoes when working on a computer (such movement is likely to build up electrostatic charges), and you should frequently ground yourself in other ways, such as by touching a radiator or water tap. If you fail to take such actions, you may destroy a computer or a component just by touching it, particularly if you're working in a dry environment.

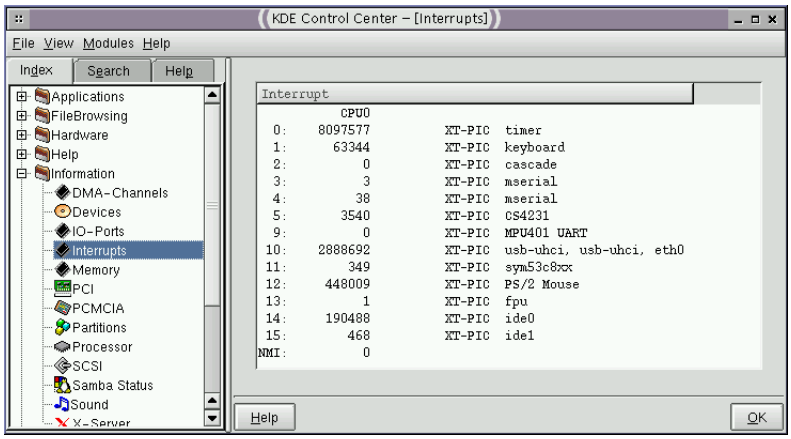
Protecting the computer is important, but still more important is protecting yourself. You'll be safest if you unplug a computer from the wall before working on it. Modern computers don't shut off all power when they're turned off, so even if you've shut down a system, it will still have a few live circuits, which might give you a jolt if you accidentally touch one. A few computers have toggle switches on their backs that can cut all power, and using such a switch can have nearly the same safety effect as unplugging the computer. (When switched off but plugged in, a computer with such a switch is also a useful ground source—you can ground yourself by touching the power supply rather than a radiator or water tap.)

Determining Available Resources

Many devices require the use of particular types of limited hardware resources. These include an interrupt request (IRQ), which allows hardware to signal the CPU when an important hardware event occurs; a direct memory access (DMA) channel, which allows for data transfer; and an input/output (I/O) port, which is another means of data transfer. As a general rule, each device requires one of these (or a range, in the case of I/O ports), and it's not possible to share resources. Some hardware, though, *can* share resources with other devices. This is particularly true of Peripheral Component Interconnect (PCI) boards.

You can discover what resources a Linux computer is currently using from three pseudo-files in the `/proc` filesystem: `/proc/interrupt`, `/proc/dma`, and `/proc/ioports`. You can use the `cat` command to view the contents of any of these pseudo-files. Some GUI environments also allow you to view this information (for instance, the KDE Control Center shown in Figure 8.7).

FIGURE 8.7 GUI tools can be convenient for obtaining hardware information, but the same information is available from the /proc filesystem.



When you are planning to add hardware, you should look for gaps in the available resources. For instance, in Figure 8.7, IRQs 6–8 are unused, and so they might (at least theoretically) be used by some new device. One key point to remember when you are checking on your available resources, however, is that Linux doesn't list resources used by devices for which drivers are not currently loaded. For instance, the floppy drive uses IRQ 6, but because the floppy drive wasn't in use on the system when Figure 8.7 was captured, IRQ 6 shows up as being free, when in fact it's not.

Figure 8.7 also shows an instance of a shared IRQ: IRQ 10 is used by both USB and Ethernet drivers. This works because on this system, both devices are PCI components capable of sharing an interrupt.

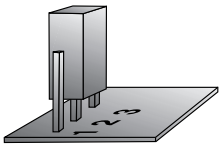
When you add SCSI devices, USB devices, printers, scanners, and many newer PCI devices, there's little reason to check on available resources. (This assumes the basic port, such as a SCSI adapter or USB port, already has resources assigned.) These peripherals either use a device that's already installed, and so they don't consume additional resources, or they're automatically configured by the system and can probably share resources. If you encounter problems with such devices, you might want to check your available resources to see if there might be a conflict.

Setting Jumpers

Many older hardware devices (particularly old ISA cards) are configured through *jumpers* (Figure 8.8). These are metallic pins on the hardware device

that are covered by caps made of metal with plastic insulation. The result of covering jumper pins is that an electrical contact is made between them, which can adjust how the hardware operates. Today, most add-in cards don't use jumpers, but many motherboards still use them for setting the CPU speed or other low-level options.

FIGURE 8.8 Jumpers allow you to adjust key aspects of hardware operation, particularly on older hardware.



In order to set a jumper, you'll need to locate the hardware's documentation. (Some devices include this on the card itself.) Jumpers generally control features like the IRQ and DMA channel used by a device. They can also enable or disable features, if a device has several. For instance, an RS-232 card might support two RS-232 ports, one of which may be disabled by setting a jumper.

ISA PnP Configuration

In the mid-1990s, the *plug-and-play (PnP)* specification for ISA cards became popular. This specification allows for jumperless configuration of many features of ISA hardware. There are two main ways to handle ISA PnP configuration in Linux: through the `isapnp` program and through a kernel ISA PnP configuration option.

`isapnp` is a program that can read configuration information from a file to set options on ISA PnP devices. To create a configuration file, it's best to start with the `pnpdump` program. This program creates a file that can be modified into an `isapnp` configuration file. The following is an example of this:

```
# pnpdump > isapnp.conf
```

The resulting `isapnp.conf` file contains sets of configuration options that are commented out by preceding their lines with pound signs (`#`). When you locate a configuration you'd like to set, uncomment the options in parentheses corresponding to the hardware settings you want to use, as well

as the (ACT Y) line after these options. You'll probably uncomment a total of 2–6 lines per card.

Many distributions start `isapnp` when they boot. For others, you may include a call to the utility in a startup script such as `/etc/rc.d/rc.local`. You can also run the program manually when you are testing drivers. Be sure to include the name of the configuration file. Here is an example:

```
# isapnp /etc/isapnp.conf
```

The 2.4.*x* kernels include ISA PnP support that doesn't require the separate `isapnp` utility. In these kernels, ISA PnP devices are auto-detected and configured by the kernel. Alternatively, you may be able to force specific options in the `/etc/modules.conf` file (described in Chapter 6). If you prefer, you can omit the kernel's ISA PnP support and use the `isapnp` tools with 2.4.*x* kernels.

There's no need to use ISA PnP support for PCI devices. These devices are auto-configured by the kernel or BIOS, although `/etc/modules.conf` options may be able to force particular configurations.

Loading Appropriate Drivers

The Linux kernel can load drivers in either of two ways: It may include the drivers in the main kernel file, or it may load drivers from modules. Drivers that aren't directly associated with the kernel, such as Ghostscript or SANE drivers, may be built into the application itself or may be loaded separately, as well. Ghostscript always uses drivers built into the main application, which means that if you want to add an unusual driver, you must locate a Ghostscript binary that includes it, or recompile Ghostscript yourself. (The latter is a very tedious process.) SANE normally uses modular drivers.

To add a driver to the kernel itself, you will recompile the kernel; you must tell the system to include the driver in the kernel rather than build it as a module. The upcoming section, "Managing Kernel Modules," covers loading kernel modules. Configuring a printer queue to use specific Ghostscript drivers was described earlier, in "BSD and LPRng Configuration and Use."

Setting Driver Options

It's sometimes necessary to set some options related to the hardware in a driver. For instance, you may need to tell a driver what hardware settings you used for a device, or you might tell the driver that you've installed *two*

identical devices (such as two Ethernet cards). The Linux kernel refers to these as driver options, and there are two ways to set them:

Kernel options When a driver is compiled into the kernel proper, you pass driver options to the kernel itself. In most cases, this is done by using the `append` option in `/etc/lilo.conf`. For instance, you might use the following line to tell the kernel to use IRQ 10 and I/O port 6200 for the Ethernet device (`eth0`):

```
append="ether=10,0x6200,eth0"
```

Module options If a driver is loaded as a module rather than as part of the kernel file proper, you enter the module options in the `/etc/modules.conf` file, as described later in this chapter and in Chapter 6.

In either case, the options that a driver accepts are highly specific to the driver in question. Sometimes the same information (such as an IRQ number) may be passed to different modules in different ways. Therefore, it's necessary to consult the documentation for a particular driver to use it properly.

Making Hardware Accessible to Users

As a general rule, Linux makes hardware accessible through device file entries in the `/dev` directory tree. Table 8.1 summarizes common device filenames and their uses. These files all have ownership and permission like other Linux files, as described in Chapter 4's "File Permissions" section. These permissions control who may access particular files and therefore the hardware.

TABLE 8.1 Common Linux Device Filenames and Their Uses

Device Filename	Major Numbers	Function
`/dev/fd*`	2	Floppy disk access.
`/dev/hd`*xy*	3, 22, 33, and 34	EIDE hard disk access. *x* is a letter from a onward representing a disk drive; *y* is an optional number from 0 up representing a partition. Also used for EIDE CD-ROMs.

TABLE 8.1 Common Linux Device Filenames and Their Uses *(continued)*

Device Filename	Major Numbers	Function
/dev/sd*xy*	8, 65–71	SCSI hard disk access. *x* is a letter from a onward representing a disk drive; *y* is an optional number from 0 up representing a partition.
/dev/scd*x*	11	SCSI CD-ROM drives.
/dev/ht*x* and /dev/nht*x*	37	EIDE tape backup devices.
/dev/st*x* and /dev/nst*x*	9	SCSI tape backup devices.
/dev/sg*x*	21	"Generic" SCSI devices. *x* is a letter from a onward.
/dev/ttyS*x*	4	RS-232 serial ports.
/dev/lp*x*	6	Parallel ports.
/dev/usb/*	variable	USB devices.
/dev/dsp*x*	14	Digital audio (sound cards).
/dev/sequencer*x*	14	MIDI audio playback (sound cards).
/dev/mixer*x*	14	Audio mixer (controls sound card volume).
/dev/psaux	10	PS/2-style mouse (note x is *not* a variable; there's only one /dev/psaux device).

Unless otherwise noted, *x* is a number from 0 up representing the device number. This variable may sometimes be omitted.

Table 8.1 specifies a "major number" for each device type. In Linux, every device file is associated with two numbers: a major number and a minor number. Together, these numbers determine the device class and the specific device. For instance, /dev/ttyS0 uses major number 4 and minor number 64; /dev/ttyS1 uses minor number 65; /dev/ttyS2 uses minor number 66; and so on. If necessary, you can use the mknod command to create a device file for new hardware. This command requires the major and minor number to create a device file. Creating a device file with mknod is almost

never necessary because Linux distributions invariably ship with device files for all common hardware. You're most likely to need to create special device files if you're using a very new and experimental hardware driver. In such a case, you'll need to consult your driver's documentation for complete instructions.

Sometimes, users must have direct access to device files in the `/dev` directory tree. For instance, this is required if users should be able to make outgoing calls on a modem (typically a serial port device), to play or record sounds on a sound card, or to play audio CDs on a CD-ROM drive. Some distributions, such as Red Hat, include code in their login procedures to change the ownership of such files to the user who logs in at the console. This approach generally works well for workstations. Other distributions don't use this approach; instead, you as an administrator must adjust the permissions to give users access to a device. For instance, you might create an `audio` group, assign sound-related device files to this group, and give them `rw-rw----` permissions. Any user who's a member of the `audio` group can then use the sound card. Some distributions include groups designed for this purpose.

Other device files are accessed in different ways, typically through the kernel. For instance, disk devices are accessed through filesystem drivers. Therefore, it's usually not necessary to give ordinary users direct access to disk device files. (Floppies are sometimes an exception, particularly if you want users to be able to format blank floppies or use certain floppy utility packages.) In fact, giving ordinary users direct access to such device files can be an invitation to disaster. With direct read access to a disk device, an ordinary user can read any file on the disk, for instance—a major security hole. Write access is worse still. Even access to serial ports—necessary for some purposes—can be a problem if users begin making unauthorized long-distance calls.

Some devices aren't accessed through files in the `/dev` directory tree. Notable among these are networking devices, such as Ethernet cards. Most programs can't directly touch networking hardware; their access must be filtered through the network protocol stack, which buffers all network communications. Basic network configuration is covered in Chapter 5, "Networking."

Managing Kernel Modules

Linux *kernel modules* are equivalent to driver files in many other OSs. Kernel modules are very dynamic, though; they can be loaded and unloaded on demand, without restarting the computer. Also, some kernel modules depend on others. For instance, the SCSI system consists of a driver that handles the SCSI host adapter (and is very specific to particular models), and additional modules that handle SCSI hard disks, SCSI tape devices, and so on. These latter modules rely upon the former ones. Understanding these module dependencies and loading all the modules needed to use a device is critically important to effective use of hardware in Linux.

Kernel Module Configuration Files

Linux kernel module configuration is handled through two files:

`/etc/modules.conf` This file (called `/etc/conf.modules` on some distributions) contains information on the modules to be used for particular tasks, as well as instructions on how the kernel should automatically load and unload modules. Its format is outlined in Chapter 6.

`/lib/modules/`*`x.y.z`*`/modules.dep` This file, which is located in a directory named after the kernel version in use (*`x.y.z`*), contains information on module dependencies—each line of this file lists a module along with the names of modules upon which the first one relies.

You may need to adjust `/etc/modules.conf` to have Linux automatically load modules when needed, as outlined in Chapter 6. Linux maintains the `modules.dep` file semiautomatically. This file is normally generated whenever you install kernel modules, but if you manually change your configuration, you should type **`depmod -a`** to update the file.

`depmod` can be used for other purposes, as well. Most importantly, `depmod -e` reviews your kernel modules to ensure that they're internally consistent. If you type this command and receive back a message that there are "unresolved symbols," it means that some modules are missing or were compiled for a different kernel than the one you're using. This problem can prevent modules from loading, but it may be unimportant if it involves modules you never use. If it's preventing your modules from loading, you may need to completely recompile your kernel from scratch to ensure that all the modules are synchronized with one another.

Inserting and Removing Kernel Modules

If your kernel was compiled with kernel module loader support, the kernel has the ability to load modules as it sees the need for them. For instance, if your system has USB support compiled as modules, any attempt to access a USB device causes the kernel to look for and load the appropriate kernel modules. Typically, these modules are then automatically unloaded when the device is no longer in use. This arrangement is certainly very convenient, but it sometimes doesn't work correctly, particularly with new or unusual modules, or if `/etc/modules.conf` doesn't include appropriate aliases to help the system determine which modules are needed for particular devices (say, to load the driver for an Adaptec SCSI card rather than a Symbios SCSI card). For this reason, there are three programs used to manually load and remove kernel modules. (In fact, the kernel uses these tools itself when auto-loading modules.) You may include these commands in startup scripts or enter them manually as you see fit. The latter is often helpful when testing new hardware.

Automatically unloading modules can cause performance degradation in a few cases, such as if a device must respond quickly to some event or if the device is used frequently.

insmod

The first of these kernel-handling commands is `insmod`. This program inserts a single module into the kernel. This command takes a module name as a parameter, and it may accept several options, as well. The more important `insmod` options are as follows:

`-f` Forces `insmod` to load the module even if it was compiled for a different version of the kernel. This is most useful when you are dealing with commercial binary-only modules.

`-k` Sets the auto-clean flag on the module. This causes the system to automatically remove the module after it's fallen into disuse.

`-p` Tests to see whether the module *could* be loaded, without actually doing so.

`-s` Outputs all messages to the system log, rather than to the console.

For instance, you might use the following command to load the `ltmodem` module for the Lucent software modems that are used on many laptop computers:

```
# insmod -f ltmodem
```

Modules usually have the same module name and filename, except that module filenames generally include a trailing `.o`. Therefore, the `ltmodem` module is called `ltmodem.o` on the hard disk. These modules are stored in subdirectories of `/lib/modules/x.y.z`, where `x.y.z` is the kernel version number (sometimes with a distribution build number added).

modprobe

`insmod` is a useful tool, but it's limited because it loads *one* module. In cases where a module depends on many others, using `insmod` requires you to determine what modules depend upon others and manually insert all of the depended-upon modules before the one you want to use. This can be a tedious process. Therefore, the `modprobe` program exists to do that work for you. `modprobe` uses the contents of `modules.dep` to determine what modules must be loaded in order to use a target module. Like `insmod`, `modprobe` supports many options; the most important of these are described here:

`-c` or `--showconfig` Displays the current module configuration. This includes paths to module files, a summary of `/etc/module.conf` information, and more.

`-k` or `--autoclean` Sets the auto-clean flag, just like the `-k` option to `insmod`.

`-n` or `--show` Doesn't perform the module insertion, but summarizes what actions would occur.

`-s` or `--syslog` Outputs all error messages to the system log, rather than to the console.

`-r` or `--remove` Removes stacks of modules. If no module is named, this option forces an auto-clean of the stack.

For instance, to insert the module `snd-card-interwave` and all the modules upon which it depends with the auto-clean flag set, you'd type the following:

```
# modprobe -k snd-card-interwave
```

rmmod

To remove modules, you can use the `--remove` option to `modprobe`, or you can use the `rmmod` command. This command supports three options:

`-a` Removes all unused modules that have their auto-clean options set.

`-r` Removes a stack of modules, not just a single module.

`-s` Outputs all messages to the system logger, rather than to the console.

In practice, these commands are most useful when you're trying new drivers or new hardware and want to manually configure things. This is particularly helpful when you are debugging. When you've found a working configuration, you can generally get it working correctly using the kernel auto-loader by setting appropriate aliases in `/etc/modules.conf`. In some cases, you may find it easier to force matters by installing the modules using these commands in a startup file. Manually inserting the modules also keeps the kernel from wasting time doing the task repeatedly. Although the time spent loading modules is small, this can be a factor if your system performs some other very time-critical task, such as real-time data collection.

lsmod

If you want to know what modules are loaded at any given time, you can use the `lsmod` command. This command lists the loaded modules, their sizes, and the names of the modules that depend upon them. Here is an example:

```
$ lsmod
Module                  Size  Used by
dc2xx                   2512   0 (unused)
ip_masq_ftp             2320   0 (unused)
lp                      5276   0
parport_pc              7316   2
parport                 7236   2 [lp parport_pc]
via-rhine               8996   1
```

This output reveals that the `dc2xx` and `ip_masq_ftp` modules have gone unused for a while; but since these modules did not have their auto-clean flags set, they haven't been removed. The `parport` module lists the `lp` and `parport_pc` modules as depending upon it, so you won't be able to remove `parport` without first unloading `lp` and `parport_pc`.

Diagnosing Hardware Problems

Sometimes, hardware you add doesn't work as you expect it to. There are many possible causes of such problems, ranging from defective hardware to errors when you load kernel modules. Diagnosing such problems is as much an art as a science, but this section provides some pointers to help you diagnose some common hardware problems.

Core System Problems

The motherboard (aka the mainboard), CPU, and RAM are the most critical hardware components on any computer. If these components act up, nothing else is likely to work reliably. Problems in RAM and the CPU are likely to affect many or even all programs. Motherboard problems might do the same, or they might be isolated to specific hardware devices on the motherboard, such as a USB or keyboard port.

Your first chance to spot core system problems comes during the system boot process. At this time, *x*86 BIOSes engage in a *power-on self-test (POST)*. This is a test of certain critical components, such as the RAM, the presence of a keyboard and video card, and so on. Most computers beep if they fail the POST. In fact, most BIOSes produce a different number of beeps depending upon the exact nature of the problem. Unfortunately, these beep codes aren't standardized, so you'll have to check with your motherboard or BIOS manufacturer to learn what the codes mean for your particular system. If a system fails its POST, a good starting point is to reconnect all the devices that are connected to the computer, especially the keyboard, CPU, RAM, and all expansion cards. Sometimes a POST failure is accompanied by an on-screen indication of the problem. For instance, most systems display a progress indicator when they perform their memory tests. If that indicator stops partway through, there's a good chance that the BIOS has found defective RAM.

Other core system problems don't make themselves felt until Linux has begun booting, or even later. Defective CPUs and RAM often manifest in the form of kernel oopses, for instance. The Linux kernel includes code that displays a summary of low-level problems on the screen (and logs it, if the system still works well enough to do this). This summary includes the word `oops`, and it's usually the result of a hardware problem or a kernel bug. If you're running a release kernel, a kernel oops is almost always the result of

a hardware problem, such as defective RAM, an overheating CPU, or a defective hard disk.

If a problem occurs only in warm weather or after the computer's been running for a while after starting, you may need to get a better heat sink or fan for your CPU or improve the computer's internal case ventilation. The Lm_sensors package (`http://www.netroedge.com/~lm78`) is a good way to monitor your CPU's temperature, assuming your motherboard includes temperature monitoring features, as most Pentium II, Athlon, or better motherboards do.

EIDE/ATA Problems

Most *x*86 computers use hard disks and CD-ROM drives that attach via the Enhanced Integrated Device Electronics (EIDE) port, which also goes by the name Advanced Technology Attachment (ATA) or ATA Packet Interface (ATAPI). Problems with these devices can be quite serious because they can prevent Linux from booting or can cause data corruption.

One class of problem with these devices relates to what numbers Linux uses to access a particular sector on a disk. There are several different incompatible disk geometries for large disks, and if Linux attempts to use one method when another was used to define partitions, Linux may fail to boot or cause data corruption if the OS does boot. In many cases, these problems result in an inability of the Linux Loader (LILO) to boot Linux, as described in Chapter 9.

Another common type of EIDE problem relates to bugs in EIDE controllers. The Linux kernel source configuration procedures give you many options to enable workarounds and fixes for buggy EIDE controllers. Most Linux distributions ship with all of these fixes enabled, so if you're using a common controller, you shouldn't have any problems. If your computer has a particularly new controller or if you've recompiled your kernel and not enabled a fix, you may experience bizarre filesystem errors. You might find that files you've written are corrupt or that your filesystem may have errors that appear in routine `fsck` runs. In extreme cases, your computer might crash. You can overcome such problems by recompiling the kernel with appropriate bug workarounds enabled. Sometimes these problems occur because you're using a controller that's very new but that has bugs. In

such cases, you may need to replace the controller or upgrade your Linux kernel to a newer version.

Some Linux users experience very slow disk transfer speeds. These can be caused by several different factors. For instance, although Linux's basic EIDE drivers work with almost all EIDE controllers, you must use specialized drivers to obtain the best possible performance from your drive. You can use the `hdparm` utility both to test disk speed and to set various options that can improve the performance of a hard disk. `hdparm` supports a large number of options, so you should read its man page for details. The more common options include the following:

`-d [0|1]` *x*86 EIDE devices can be run in either *Programmed Input/Output (PIO)* mode or in *Direct Memory Access (DMA)* mode. In the former, the CPU directly supervises all data transfers, whereas in the latter, the CPU steps back and lets the controller transfer data directly to and from memory. Therefore, DMA mode produces lower CPU loads for disk accesses. Using `-d1` enables DMA mode. This option is generally used in conjunction with `-X` (described shortly). This option doesn't work on all systems; Linux requires explicit support for the DMA mode of a specific EIDE chipset if you're to use this feature.

`-p` ***`mode`*** This parameter sets the PIO mode, which in most cases varies from 0–5. Higher PIO modes correspond to better performance.

`-S` ***`timeout`*** This option sets an energy-saving option: the time a drive will wait without any accesses before it enters a low-power state. It takes a few seconds for a drive to recover from such a state, so many desktops leave ***`timeout`*** at `0`, which disables this feature. On laptop computers, though, you may want to set ***`timeout`*** to something else. Values between `1` and `240` are multiples of 5 seconds (for instance, `10` means a 50-second delay); `241`–`251` mean 1–11 units of 30 minutes; `252` is a 21-minute timeout; `253` is a drive-specific timeout; and `255` is a 21-minute and 15-second timeout.

`-T` This parameter performs a test of cached disk reads. In effect, this is a measure of memory and other non-disk system performance because the disk isn't accessed.

`-t` This parameter performs a test of uncached disk reads. You can use it to see if your hard disk is performing as you expect it to. (New hard disks in 2001 should return values of well over 10MBps, and usually over 20MBps; anything less than this indicates either an old hard disk or a suboptimal disk configuration.)

-v This option displays assorted disk settings.

-X *transfermode* This option sets the DMA transfer mode used by a disk. Example values of *transfermode* include 34 (for DMA mode 2) and 66 (for UltraDMA mode 2).

Many hdparm parameters can cause serious filesystem corruption if used inappropriately. Precisely what's appropriate varies from one system to another. For instance, using -X66 may be fine on one system, but it could cause filesystem damage on another. You can use the -t parameter to test a disk's performance, and then you can try experimenting with hdparm settings only if your disk performance is poor.

Suppose that you suspect your hard disk is performing poorly. You could test it as follows:

```
# hdparm -t /dev/hda

/dev/hda:
 Timing buffered disk reads:  64 MB in 12.24 seconds =
↳5.23  MB/sec
```

Indeed, this test reveals a rather anemic disk performance by modern standards. You might be able to improve matters by enabling DMA mode transfers, using an appropriate transfer mode, and then retesting, thus:

```
# hdparm -d1 -X34 /dev/hda

/dev/hda:
 setting using_dma to 1 (on)
 setting xfermode to 34 (multiword DMA mode2)
 using_dma    =  1 (on)
# hdparm -t /dev/hda

/dev/hda:
  Timing buffered disk reads:  64 MB in  4.49 seconds =
↳14.25 MB/sec
```

In most cases, such dangerous experiments won't be required, because most systems auto-configure themselves in a way that produces optimal (or at least reasonable) performance. It's best to perform such experiments *only* if an initial test with `hdparm -t` reveals poor performance. If you're still not satisfied, examine your Linux driver availability for your EIDE controller and the capacity of the controller to handle the hard disk. (A speedy modern hard disk can outstrip a controller that's a few years old.)

You can check the specifications for your hard disk to determine how well it *should* be performing. Look at the internal data transfer rate, which should be buried on a specifications sheet for your drive. By real-world standards, this value will be optimistic. `hdparm` should probably return a value of about 75–90 percent of the theoretical maximum.

SCSI Problems

There's an old joke that configuring a Small Computer Systems Interface (SCSI) chain is nine parts science and one part voodoo. In reality, this isn't true, but SCSI configuration can be tricky once you get beyond two or three SCSI devices. Common sources of problems include the following:

Termination Both ends of a SCSI chain must be terminated with a special resistor pack. Most SCSI devices have these built in, and adding or removing termination is a matter of setting a jumper or switch. To complicate matters, though, there are several different *types* of termination, and different varieties of SCSI require different termination types. Using the wrong sort of terminator can produce data transfer errors and unreliable operation. Terminating devices that don't fall on either end of the chain can also cause unreliable operation. Remember that the SCSI host adapter itself is a SCSI device. If it's at the end of a chain, it should be terminated, but if it's in the middle of a chain, it should not be. Most host adapters include BIOS utilities that let you enable or disable termination.

SCSI IDs SCSI devices are identified by ID numbers—0–7 for 8-bit (Narrow) SCSI, 0–15 for 16-bit (Wide) SCSI. If two devices share a single number, chances are that only one will show up, or one device may appear to occupy *all* the SCSI IDs. In either case, performance is likely to be slow and unreliable.

Cable lengths Maximum SCSI cable lengths range from 1.5 to 12 meters, depending upon the SCSI variety. Exceeding cable length limits typically results in data transfer errors, and hence filesystem corruption.

Cable quality Cheap SCSI cables can cause data errors, just as can incorrect termination or cables that are too long. Unfortunately, good SCSI cables can be quite pricey—$50 or more is not uncommon.

Forked chains Many modern SCSI host adapters include three connectors—typically one external connector and two internal connectors (for both Wide and Narrow internal devices). SCSI chains, however, should be one-dimensional—each device should be connected to the next one on the chain, with the SCSI host adapter itself counting as a SCSI device. Therefore, you should *not* use more than two connectors on a SCSI host adapter. Failing to heed this advice will produce data errors, much like other problems.

Most SCSI problems can be traced to one of these issues, and especially to termination and cabling problems. Because of this, useful troubleshooting techniques involve simplifying the SCSI chain. For instance, suppose you've got a chain with two SCSI hard disks, a CD-ROM drive, and a tape drive. If you only need one hard disk to boot, you should try removing all of the other devices to make as short a chain as possible. If that works, swap in a longer cable and start adding devices back to the chain. By doing this, you may find that the problem is related to the length of the cable or to a particular device.

Linux doesn't include a driver that works with all SCSI host adapters, unlike the situation for EIDE controllers. Therefore, your kernel *must* include support for your particular model SCSI host adapter. Most distributions ship with support for most SCSI host adapters, but you may find yourself unsupported if you've got a particularly exotic host adapter. In such a situation, you'll need to locate drivers or switch host adapters.

You can use the `hdparm` utility, described earlier, to test the performance of your SCSI drives. `hdparm` can *not* be used, however, to adjust SCSI drive performance. In Linux, SCSI drives operate at maximum performance at all times; there are no configurable transfer modes or any way to switch between PIO and DMA modes. (All good SCSI host adapters use DMA mode exclusively, but some very cheap ones use PIO mode only.)

Problems with Peripherals

In a computer context, a *peripheral* is a device that connects to and is controlled by a computer. Devices like keyboards, mice, monitors, and scanners are clear examples. Many devices that reside inside the computer's case are also peripherals, however. These include hard drives, CD-ROM drives, and tape backup devices. Most of these internal peripherals *could be* attached externally, given appropriate hardware.

Because the realm of peripherals is so broad, diagnosing problems with them also covers a lot of territory. As a general rule, though, peripheral problems can be broken down into three general classes: problems with the peripheral device itself, problems with the cables connecting the peripheral to the computer, and problems with the computer interface for the peripheral.

Peripheral Device Problems

One of the first steps you should take when diagnosing problems with peripheral devices is to determine whether the problem is related to drivers for the device or to the device itself. The upcoming section, "Identifying Supported and Unsupported Hardware," should help you decide whether the device *should* work in Linux. Printers, scanners, cameras, and more exotic external devices are particularly likely to require special drivers that might or might not exist in Linux. Keyboards, mice, monitors, external RS-232 modems, and EIDE and SCSI devices are almost always supported in Linux.

One useful test to perform is to try the device under another OS. Because most peripherals come with Windows drivers, installing those drivers and trying the device in Windows should give you some clue to help you decide whether the source of the problem is defective hardware or drivers. If you dual-boot a computer into Windows and the device doesn't work, you can't be sure that the problem is in the device, though; it could be in the cable or computer interface to the device. If you move the peripheral to another computer and it does work, the problem could also be in the cable or interface on the Linux computer.

Coincidences happen, so you can't conclude much *with certainty* by moving a device to another computer or OS. For instance, if you move a malfunctioning device to another computer and it still doesn't work, it could be that the software configuration on *both* computers is in error.

Peripheral Cable Problems

Cable problems are usually fairly easy to test—you can replace a cable without too much difficulty in most cases. SCSI cables, though, can be quite expensive, so you may be reluctant to buy a new cable just for test purposes. A few devices, such as mice and most keyboards, come with built-in cables. Fortunately, this latter class of device is usually quite inexpensive, so if a problem develops in a cable, you can probably replace the entire affected device.

Most peripheral cables cannot be attached to the computer backward. Unfortunately, some particularly cheap ribbon cables (used for SCSI, EIDE, and floppy devices inside the computer) lack the notch that serves to prevent backward installation. If you have such a cable, look for a colored stripe along one edge of the cable, and look for pin numbers printed on the connectors on the devices to which the cable attaches. Align the cable so that the colored stripe is associated with pin 1 on both ends, and it should work. If a cable is installed backward, the device will simply not work.

Floppy drive cables are unusual in that they include a twist—a section of cable that's cut and twisted to change the mapping of pins. You should attach your first floppy drive *after* this twist. If you attach your first drive before the twist, your drive identifiers will be confused. On a single-floppy system, your only floppy drive will be identified as `/dev/fd1` rather than `/dev/fd0`. Also, floppy cables normally include two types of connectors for the floppy drives. One form attaches to old 5.25-inch drives, and the other connects to 3.5-inch drives. You can't connect the drive to the wrong type of connector, but you should be aware of this difference so that you're not confused by it, or by the presence of five connectors on a typical floppy cable (one for the motherboard, two for the first floppy drive, and two for the second floppy drive). At most, three of these connectors will be used.

Peripheral Interface Problems

There are a handful of interfaces used for most peripherals. In addition to the EIDE and SCSI interfaces discussed earlier in this chapter, common interfaces include the following:

Floppy *x*86 computers include a floppy interface that can control up to two floppy drives. These interfaces are very mature, so the Linux drivers seldom cause problems. One configuration detail to which you may need to attend is enabling the port in your computer's BIOS setup screen. If this

is not enabled, Linux might not detect the floppy. If the BIOS configuration is correct and Linux can't use the floppy, it may be that the floppy controller is defective. As a device that's built into a motherboard, it can be difficult to replace a floppy controller, but old 486 and earlier systems often used floppy controllers on separate cards, so if you can find such an antique you may be able to make use of it.

Monitor The monitor port is part of the video card. Software problems with it usually relate to XFree86 (discussed in Chapter 2, "Installing Linux"). If the hardware is defective, there's a good chance that you won't even be able to see your BIOS startup messages.

Keyboard *x*86 computers have a keyboard port that uses either a large 8-pin DIN connector or a small mini-DIN connector. These are electrically compatible, so you can use an adapter if you have an incompatible keyboard. As with the floppy port, the keyboard port is highly standardized. In fact, there isn't even a kernel configuration option for it; the driver is always included in the kernel. A bad keyboard connector may turn up in the BIOS POST, but that isn't guaranteed. If the keyboard doesn't work in Linux, try booting a DOS floppy or using the BIOS setup utility to see if the keyboard works in a non-Linux environment.

PS/2 mouse Most *x*86 computers sold since the mid-1990s have used mice that connect through the PS/2 port. (The USB port is increasingly taking over this role, though.) These mice are standardized, although there are variants for features like scroll wheels. The Linux drivers for PS/2 mice are mature and seldom pose problems, but they do need to be included in your kernel or compiled as modules. (All major distributions include these drivers in their standard kernels or module sets.) The PS/2 port can be disabled in the BIOS, so if you're having problems, you may want to check this detail. If a PS/2 port is physically bad, you may want to replace the mouse with a model that interfaces via the RS-232 serial or USB port.

Parallel The parallel port is most commonly used for printers, but it can also handle some scanners, cameras, and external removable-media drives. Linux's parallel port support is mature, but it requires two drivers: one for the low-level parallel port hardware and one for the device being driven. These drivers are included in all major Linux distributions' standard driver sets. Like many other motherboard-based ports, most BIOSes allow you to disable the parallel port, so you may want to check this detail if you're having problems. If necessary, you can buy an ISA or PCI add-on parallel port to replace one that's gone bad on a motherboard.

RS-232 serial Most *x*86 systems include two RS-232 serial ports, but some have just one. These ports are used to connect to older mice, external modems, and various other devices. These ports are highly standardized, and the Linux drivers for them are mature and reliable. Driver problems are therefore unlikely. You may want to check the BIOS if you can't seem to get an RS-232 serial device to work.

USB The Universal Serial Bus (USB) port is a high-speed serial port that's much more flexible than the old RS-232 serial port. Some computers use USB keyboards and mice, and many other devices can connect in this way. If you're using a kernel numbered 2.2.17 or earlier, its USB support is very limited. For better USB support, upgrade to 2.2.18 or a 2.4.*x* or later kernel. Linux requires support for both the underlying USB hardware (which comes in two varieties, Open Host Controller Interface [OHCI] and Universal Host Controller Interface [UHCI]) and for each USB peripheral. Linux distributions sold in 2001 include such support, but not all USB devices are supported. Many motherboards include the option to disable USB support, so be sure it's enabled in the BIOS.

IEEE-1394 The latest high-speed external interface is *IEEE-1394*, aka *FireWire*. This interface is much faster than USB, and it is considered both an alternative and a successor to SCSI for some purposes. Although rare in 2001, IEEE-1394 is likely to grow in importance. Linux's IEEE-1394 support is limited, but it is likely to expand in the future. Check `http://linux1394.sourceforge.net` for more information. IEEE-1394 interfaces are rare on motherboards in 2001, so you may need to buy an appropriate PCI card to handle these devices.

Network Network ports are handled by Linux's network drivers and a network stack, as discussed in Chapter 5. Network interface card drivers are far from standardized, but Linux includes support for the vast majority of Ethernet cards and many cards of other types. If you have a particularly new card, you may need to replace it to get a model with Linux support. Identifying defective hardware may require booting into another OS or moving the card to another computer.

Most of these interfaces, as noted, are highly standardized, so Linux drivers shouldn't be incompatible with your hardware. Network, IEEE-1394, and to some extent USB interfaces are not so standardized, though, and so they sometimes cause problems. There's also the potential for driver incompatibility with many expansion card devices, like SCSI host adapters, sound cards, and video capture boards.

Identifying Supported and Unsupported Hardware

Over the years, Linux has acquired an extensive collection of drivers for a wide variety of hardware. Nonetheless, Linux doesn't support every device. Figuring out which devices are supported and which aren't can be a challenge at times because Linux drivers are usually written for a device's chipset, not for a specific device by brand and model number. For instance, it's not obvious that the Linux Tulip driver works with the Linksys LNE100TX.

To identify what hardware is supported and what isn't, you may want to consult the hardware compatibility lists maintained by various distributions. For instance, `http://hardware.redhat.com` and `http://www.suse.com/us/support/hardware` are good resources. The Linux Hardware Compatibility HOWTO (`http://www.linuxdoc.org/HOWTO/Hardware-HOWTO`) can also be an excellent resource.

Hardware compatibility varies very little from one distribution to another. The only differences result from one distribution including a non-standard driver that another doesn't include, or from peculiarities of configuration that result in conflicts between devices. Therefore, if a device is listed as supported in one distribution, that device will almost certainly work in any other distribution.

You should also check with the hardware's manufacturer if you can't find drivers or aren't sure which drivers to use. Some manufacturers include Linux drivers (usually just the standard kernel drivers) or links to information about Linux compatibility with their products on their Web pages.

Manufacturers sometimes change their products' design without changing their names. Therefore, the presence of a product on a compatibility database, or even compatibility information on the manufacturer's Web site, may not mean that the device will work. Pay careful attention to details like a board's revision number when you are searching for compatibility information.

Using Linux with a Laptop

Laptop computers (also known as *notebook computers*) present certain challenges for Linux, beyond the usual driver issues. Some of these

challenges derive from the fact that Linux, as a Unix clone, was designed for systems that would not have to deal with power management or swapping out PC Cards. The most frustrating laptop problems, however, frequently deal with installation. These problems mostly boil down to the difficulty of getting Linux's XFree86 configuration to work with the finicky *liquid crystal displays (LCDs)* used on laptops.

Special Laptop Installation Issues

Laptops are essentially miniaturized desktop computers. A laptop includes a motherboard, a CPU, RAM, a keyboard, a mouse (usually in the form of a touch pad or TrackPoint), a hard disk, a CD-ROM drive, a floppy drive, and a display. From a software point of view, these devices all work like their counterparts on a desktop computer. There are some limitations that are more common on laptops, though. These include the following:

Modem Most modern laptops include built-in modems. Unfortunately, most of these on *x*86 laptops are software modems that require special drivers. Such drivers are rare in Linux. You can check `http://www.linmodems.org` for information on these devices. In most cases, you'll have to install the modem drivers after you install Linux. If your built-in modem doesn't have Linux drivers at all, you can use a separate PC Card or external RS-232 or USB modem. If you buy a PC Card or USB modem, be sure you get one that's supported by Linux.

Networking All modern Macintosh and some high-end *x*86 laptops include built-in Ethernet support. Before buying the laptop, be sure Linux includes drivers for the Ethernet chipset. If your computer doesn't have Ethernet support and you need to use it on a network, you can buy a PC Card or USB Ethernet adapter. (PC Card adapters are preferable.) If the card is inserted when you install Linux, it may be detected by the installer and can be configured as described in Chapter 2. Once the card is configured, most distributions will automatically detect whether or not the PC Card Ethernet adapter is inserted at system boot time, and then they will configure networking appropriately.

Display The single biggest problem with running Linux on laptops relates to the display. Specifically, the LCDs used on laptops tend to accept only a very narrow range of horizontal and vertical refresh rates. (These are discussed in Chapter 2.) Therefore, it may be difficult or

impossible to get a GUI installer running, and when it comes time to configure X, you may or may not be able to find a working configuration. Even if a system works, it may not work at the optimum resolution. Because LCDs are built from a fixed number of discrete pixels, they work best at a single resolution (usually 800 × 600 or 1024 × 768 for laptops). Running in a lower resolution usually produces a chunky display. Another factor is that, because laptops' display circuitry is not removable, whatever video chipset a laptop uses *must* have Linux support if you expect to use the laptop with X.

Before you buy a laptop or attempt to install Linux on one, check the Linux on Laptops Web page (`http://www.linux-laptop.net`). This Web site includes links to many user-maintained sites concerning running Linux on specific laptop models. You can obtain useful tips and configuration procedures, and you might even pick up some working `XF86Config` files, from these pages. The main Linux on Laptops site also includes links to information on useful laptop utility programs and other information.

If you have trouble installing Linux on a laptop, you might do well to try another distribution, or try using a distribution's text-mode installation routine, if it has one. Sometimes video or other problems simply prevent a distribution from installing. For instance, I've tried installing various versions of five distributions on my own laptop. One installed fine using its GUI installation tools. Two didn't have GUI installation routines, and installed fine using text-mode tools. One distribution didn't install at all in an older version, but a newer version worked in text mode only (the GUI installer failed). The fifth distribution wouldn't install at all.

Understanding Power Management

Laptop computers are often run on battery power. Most laptop batteries can run for only a couple of hours. Therefore, modern laptops include various power management tools that can dramatically reduce the computer's need for power, thus extending battery life. The 2.4.*x* Linux kernels include two sets of power management tools: *Advanced Power Management (APM)* and *Advanced Configuration and Power Interface (ACPI)*. Both require underlying support in the computer's BIOS. As of the early 2.4.*x* kernels, APM is

mature, but ACPI is new and experimental. For this reason, it's often best to use APM, even if your hardware supports ACPI. Most Linux distributions' kernels include APM support.

Although primarily intended for laptops, power management tools can be used on desktop systems, as well. In fact, this is how Linux powers off a computer when it shuts down.

To use APM features effectively, you need some way to tell the computer when to enter power-conserving states. This task is accomplished with the `apmd` package (`http://www.worldvisions.ca/~apenwarr/apmd`), which ships with most Linux distributions and may be installed automatically. The main `apmd` program is a daemon, so it should be started when the computer boots. Once running, it monitors the system's battery status and, if the battery's charge gets too low, `apmd` kicks the system into a suspend mode in which most functions are shut down and only the system's RAM is maintained. `apmd` will also suspend the hard disk if it's gone unused for a long enough time. (You can use the `hdparm` utility, described earlier, to control hard disk power management more directly.)

If you want to manually control APM features, you can do so with the `apm` utility. Typing this command manually presents basic power management information, such as how much battery power is left. The `-s` and `-S` parameters cause the system to go into suspend and standby modes, respectively. Suspend mode shuts off power to most devices, leaving only the CPU and memory operating, and those at minimum power. Standby mode leaves more devices powered up, so the system can recover more quickly; but there's less power savings in this mode. A fully charged laptop can usually last several hours in standby mode and a day or more in suspend mode. Many laptops include a key sequence that will force the system into suspend or standby mode. In most cases, `apmd` will detect such a keystroke and honor the request. Consult your laptop's documentation for details.

Using PC Card Devices

Because laptops don't have ISA or PCI slots, manufacturers developed a standard for expansion cards that allows you to easily insert and remove many of the types of devices that would go in an ISA or PCI slot on a desktop computer. This standard was originally named after the industry group that

developed the standard, the *Personal Computer Memory Card International Association (PCMCIA)*. To reduce the number of acronyms, though, this standard has since been renamed *PC Card*. Many Linux utilities still use the old name.

There are PC Card adapters for desktop systems, so PC Card utilities sometimes find use on these systems. PC Card devices are much more common on laptops, though.

There are several different varieties of PC Card hardware. There are three different sizes of PC Cards: Type I, Type II, and Type III, with each type being thicker than the preceding one. Type I cards are often used for memory expansion. Type II cards are the most common type, and they are used for Ethernet cards, modems, and the like. Type III cards are rare, and they are used for hard disks or other devices with internal moving components. Electronic standards include PCMCIA 1.0, PCMCIA 2.0, PCMCIA 2.1, and PC Card, with the last of these being the most advanced. You can learn more about all of these at the PC Card Web site, `http://www.pc-card.com`.

Unlike support for most hardware, PC Card support doesn't come with the Linux kernel. Instead, you must acquire and install an auxiliary driver package. This package is hosted at `http://pcmcia-cs.sourceforge.net`. Fortunately, most Linux distributions include these PC Card drivers, so there's no need to go looking for them unless you need support for a particularly new device or you upgrade your kernel by manually compiling it yourself.

PC Cards are designed to be inserted and removed at will. Unfortunately, Linux's driver model doesn't work well with such hot swapping. Therefore, the PC Card driver set includes a feature known as *Card Services*, which helps you smoothly install and remove drivers from the kernel and also helps the kernel cope with potential problems. (For instance, automatically starting or stopping network services when an Ethernet PC Card is installed or removed.) Card Services are controlled through configuration files in `/etc/pcmcia`. There are scripts in this directory for different types of services, such as `network` and `ide`. If your distribution's maintainers paid proper attention to PC Card devices, these scripts should require no modifications to work correctly. In some cases, though, you'll need to edit these scripts to have them do the right thing. Details of doing this are very distribution- and device-specific. The "Basic Shell Scripting" section of Chapter 6 may help you understand these scripts if you need to modify them.

Summary

Configuring hardware in Linux requires a wide range of skills. Some configurations, like setting up swap space, creating and managing printer queues, and managing kernel modules, are handled differently in Linux than in other OSs. Printer configuration is particularly unusual in Linux because it relies upon the presence of either a PostScript printer or Ghostscript, a PostScript interpreter that runs under Linux.

When you add new hardware, you must locate Linux drivers for the devices. These may come in the form of kernel modules, or they may reside in non-kernel packages, such as Ghostscript. You physically install the hardware much as you would in any other OS, though, and you should take precautions for both your and the hardware's safety.

Sometimes, problems arise with new hardware. Common problems include defective or overheated motherboards, CPUs, and RAM; misconfigured or defective EIDE devices; and misconfigured or defective SCSI devices. Other devices can also cause problems, especially if the hardware is exotic or uses a new design. One particularly tricky type of hardware is a laptop computer. Laptop displays, power management, and PC Card devices all pose challenges, but not insurmountable ones.

Exam Essentials

Identify when swap space needs to be increased. The output of the `free` command shows how much memory Linux is using—both RAM and swap space. When the amount of used swap space approaches available swap space, it's necessary to increase swap space or RAM.

Describe the role of Ghostscript in Linux printing. Ghostscript serves as an on-computer PostScript interpreter for non-PostScript printers, allowing programs that expect to print to PostScript printers to be used with less expensive printers.

Describe the role of `lpd` in Linux printing. The line printer daemon (`lpd`) accepts local and remote print jobs, maintains the local print queue, calls smart filters, and passes data to the printer port in an orderly fashion.

Contrast drivers in the kernel to driver modules. The Linux kernel is responsible for all drivers. Drivers stored in the Linux kernel are always loaded and increase the size of the kernel file, but driver modules may be loaded and removed dynamically, reducing the memory load at the cost of configuration complexity.

Identify some possible sources for Linux drivers. Linux drivers may be obtained as part of the Linux kernel, from hardware manufacturers, from third-party development efforts, and from Web sites devoted to particular types of hardware.

Summarize how laptop installation and use differs from desktop installation and use. Installing Linux on a laptop requires careful attention to the laptops's non-replaceable hardware, and especially to the display, which can be finicky. Using the system may require using PC Card devices and power management, both of which are unused or less important on desktop systems.

Commands in This Chapter

Command	Description
`free`	Displays information on total RAM and swap space use
`mkswap`	Initializes a file or partition for use as swap space
`swapon`	Activates use of swap space
`swapoff`	Deactivates use of swap space
`lpr`	Submits a print job to a print queue
`lpq`	Displays information on jobs in a print queue
`lprm`	Deletes jobs from a print queue
`lpc`	Monitors and controls a print queue
`pnpdump`	Creates a prototype `isapnp` configuration file based on installed hardware
`isapnp`	Configures ISA PnP devices
`depmod`	Locates module dependencies
`insmod`	Inserts a single module into the kernel
`modprobe`	Inserts a module and all the modules upon which it depends (a module stack)
`rmmod`	Removes a module or a stack of modules from the kernel
`hdparm`	Sets disk driver parameters and tests disk performance
`apm`	Controls APM features in Linux

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

- Advanced Configuration and Power Interface (ACPI)
- Advanced Power Management (APM)
- baud rate
- Card Services
- daemon
- electrostatic discharge (ESD)
- FireWire
- fragmented
- IEEE-1394
- jumper
- kernel module
- laptop computer
- liquid crystal display (LCD)
- notebook computer
- PC Card
- peripheral
- Personal Computer Memory Card International Association (PCMCIA)
- plug-and-play (PnP)
- PostScript
- power-on self-test (POST)
- print queue
- Printer Control Language (PCL)
- printer driver
- smart filter
- software modem
- spool directory
- swap file
- swap partition
- swap space

Review Questions

1. Where may a swap file be located?
   - **A.** Only on the root (/) Linux filesystem
   - **B.** On local read/write Linux filesystems
   - **C.** On NFS or ext2 filesystems
   - **D.** On any partition with more than 512MB of free disk space
2. In which of the following situations would it be *most* reasonable to create a new swap partition?
   - **A.** Your heavily used server is nearly out of swap space and needs no routine maintenance.
   - **B.** A workstation user has been using memory-hungry programs that exceed memory capacity and needs a quick fix.
   - **C.** You're adding a new hard disk to a multiuser system and expect several new users in the next month or so.
   - **D.** A system has been experiencing slow performance because of excessive swapping.
3. Which of the following is generally true of Linux programs that print?
   - **A.** They send data directly to the printer port.
   - **B.** They produce PostScript output for printing.
   - **C.** They include extensive collections of printer drivers.
   - **D.** They can only print with the help of add-on commercial programs.

4. Which of the following describes the function of a smart filter?

   A. It detects the type of a file and passes it through programs to make it printable on a given model of printer.

   B. It detects information in print jobs that might be confidential, as a measure against industrial espionage.

   C. It sends e-mail to the person who submitted the print job, obviating the need to wait around the printer for a printout.

   D. It detects and deletes prank print jobs that are likely to have been created by miscreants trying to waste your paper and ink.

5. Which of the following is an advantage of GUI printer configuration tools over manual configuration?

   A. GUI tools allow you to enter options not possible with text-based tools.

   B. GUI tools include the ability to detect ink cartridge capacity in inkjets.

   C. GUI tools let you configure non-PostScript printers to accept PostScript output.

   D. GUI tools hide the details of smart filter configuration, which vary across distributions.

6. What information about print jobs does the `lpq` command display? (Choose all that apply.)

   A. The name of the application that submitted the job

   B. A numerical job ID that can be used to manipulate the job

   C. The amount of ink or toner left in the printer

   D. The username of the person who submitted the job

7. Which devices require drivers that are *not* part of the standard Linux 2.4.*x* kernels? (Choose all that apply.)

   A. CDC-ACM USB modems

   B. Scanners

   C. Non-PostScript printers

   D. Most Ethernet cards

8. In what ways may you set driver options in Linux? (Choose all that apply.)

   A. By passing the option with the `insmod` command

   B. By using the `append` option in `lilo.conf`

   C. By using the `option` command in `modules.dep`

   D. By using the `option` command in `modules.conf`

9. What is the purpose of the `/lib/modules/`*`x.y.z`*`/modules.dep` file?

   A. To tell the kernel what module to load to access a particular class of device

   B. To tell `modprobe` what modules to load in conjunction with a selected module

   C. To indicate which drivers should be compiled as modules when rebuilding the kernel

   D. To keep track of which modules are loaded at any given moment

10. What information does `lsmod` provide?

    A. The date a module was compiled

    B. A list of currently installed modules

    C. The IRQs used by installed modules

    D. The number of minutes until a module is automatically uninstalled

11. What is the purpose of the POST?

A. To shut off power after a system shutdown

B. To perform basic hardware tests at power up

C. To hand off control from LILO to the kernel

D. To test a printer's PostScript capabilities

12. Why should you be cautious when using `hdparm`?

A. `hdparm` can set hardware options that are not supported by some hardware, thus causing data corruption.

B. Because `hdparm` modifies partition tables, an error can result in loss of one or more partitions and all their data.

C. By changing hardware device file mappings, you can become confused about which drive is `/dev/hda` and which is `/dev/hdb`.

D. `hdparm` can cause Linux to treat an ext2fs partition as if it were FAT, resulting in serious data corruption.

13. A SCSI chain on a single-channel SCSI card is behaving unreliably so you examine it. You find that devices are attached to all three connectors on the SCSI host adapter, for a total of five devices. The device at the end of each cable is terminated, the cables are of high quality, and no two devices share a SCSI ID number. Which of the following is the most likely cause of the problems?

A. None of the devices should be terminated.

B. Only one of the devices should be terminated.

C. Only two of the host adapter's connectors should be used.

D. There should be only four devices attached to the host adapter.

14. You're having problems with a digital camera under Linux. You move the camera (including its cable) to another computer that runs Windows, but the camera doesn't work under Windows, either, even when you install the Windows software that came with the camera. What can you conclude?

A. The problem is almost certainly related to the Linux drivers or camera software.

B. The problem is very likely related to the cable or the camera hardware.

C. The problem probably resides in the computer's interface hardware.

D. The problem is definitely *not* related to the camera's hardware.

15. Which of the following devices are highly standardized in *x*86 systems and so have mature Linux drivers that don't vary from one model to another? (Choose all that apply.)

A. Parallel ports

B. Floppy ports

C. SCSI host adapters

D. Ethernet adapters

16. You've purchased a new PCI sound card for a computer, but when you open the computer, you find it has only three PCI slots, and they're all used by other devices. The computer has two free ISA slots. What options do you have if you need to add sound support for this computer? (Choose all that apply.)

A. Insert the PCI card into an ISA slot.

B. Purchase an ISA sound card for the computer.

C. Purchase a PCI-to-ISA adapter so you can use the PCI card in an ISA slot.

D. Remove a PCI card that's less important than the sound card, to make room for the latter.

17. Why is it best to unplug a computer from the wall or surge protector when performing work on it?

A. If a computer is plugged in, you're more likely to damage it with an electrostatic discharge.

B. Modern computers have live circuits even when turned off. The current in these circuits can injure you.

C. Unplugging the computer reduces the chance that an electrostatic charge will build up in the system, thus damaging it.

D. External surge protectors can damage equipment if that equipment is powered off.

18. What solution might you attempt if a computer routinely generates kernel oopses on warm days but not on cool days?

A. Replace a 4500rpm hard disk with a 7200rpm model.

B. Upgrade the heat sink and fan on the CPU.

C. Upgrade to a more recent kernel.

D. Nothing; kernel oopses are normal.

19. Which components continue to receive power when a laptop computer has entered suspend mode? (Choose all that apply.)

A. The CPU

B. The hard disk

C. The display

D. RAM

20. Which of the following is a challenge of PC Card devices, from a Linux point of view?

    A. PC Card devices draw more power than Linux can support, leading to unreliable operation if APM support isn't enabled.

    B. Linux wasn't designed to expect most devices to appear and disappear randomly, as they do when a user inserts or removes a PC card device.

    C. Supporting PC Card devices requires adding a new type of device hierarchy, which conflicts with existing device types.

    D. The only way to support PC Card devices is to treat them like floppies, which makes using communication devices difficult.

Answers to Review Questions

1. B. A swap file may be located on local read/write filesystems. This includes, but is not limited to, the root filesystem. Swap space may *not* exist on NFS mounts (which are very slow compared to local disk partitions in any event). The amount of free disk space on the partition is irrelevant, so long as it's sufficient to support the swap file size.

2. C. It's easy to create a swap partition when adding a new disk, and in option C, the new user load might increase the need for memory and swap space, so adding a new swap partition is prudent. In options A and B, adding a swap partition would require downtime while juggling the partitions, and so it would disrupt use of the systems. Adding a swap file makes more sense in those cases. In option D, adding swap space won't speed performance much (unless it's on a faster disk than the current swap space); a memory upgrade is in order to reduce reliance on swap space.

3. B. PostScript is the de facto printing standard for Unix and Linux programs. Linux programs generally *do not* send data directly to the printer port; on a multitasking, multiuser system, this would produce chaos because of competing print jobs. Although a few programs include printer driver collections, most forego this in favor of generating PostScript. Printing utilities come standard with Linux; add-on commercial utilities aren't required.

4. A. The smart filter makes a print queue "smart" in that it can accept different file types (plain text, PostScript, graphics, etc.) and print them all correctly. It does not detect confidential information or prank print jobs. The `lpr` program can be given a parameter to e-mail a user when the job finishes, but the smart filter doesn't do this.

5. D. Linux distributions ship with different smart filter configurations, each of which can be tedious to configure in different ways. Although GUI tools also differ, they're somewhat easier to figure out and have similar options to one another. GUI tools are *not* more flexible than text-based tools; after all, the GUI tools simply manipulate the underlying textual configuration files. Both GUI and text-based configurations can invoke smart filters to print PostScript on non-PostScript printers.

6. B, D. The job ID and job owner are both displayed by `lpq`. Unless the application embeds its own name in the filename, that information won't be present. Most printers lack Linux utilities to query ink or toner status; certainly `lpq` can't do this.

7. B, C. Scanners and non-PostScript printers both require non-kernel drivers (as part of SANE and Ghostscript, respectively). Drivers for CDC-ACM USB modems and most Ethernet cards come standard with the 2.4.*x* kernels, although USB support is new with the 2.2.18 and 2.4.*x* kernels.

8. B, D. The `append` option in `/etc/lilo.conf` sends an option to a driver built into the kernel at boot time. The `option` command is part of the `/etc/modules.conf` file's feature set, not of the `/lib/modules/x.y.z/modules.dep` file's feature set. `insmod` has no mechanism for passing options to drivers, aside from options set in `modules.conf`.

9. B. `modules.dep` contains module dependency information—which other modules each one requires in order to operate. `modprobe` uses this information to load prerequisite modules whenever requested to load one module.

10. B. `lsmod` shows a list of currently installed modules, their sizes, and the modules that depend upon them. It does not provide information on compilation dates, IRQs, or an estimate of time before the module is removed. (You can obtain IRQ use information from the `/proc/interrupts` pseudo-file, though.)

11. B. POST stands for "power-on self-test." It's a BIOS routine that checks for basic functionality of core system components, such as RAM integrity and the presence of a keyboard. Most computers provide an encoded beep if the POST fails.

12. A. `hdparm` manipulates low-level options in EIDE hard disk controllers, such as the use of DMA or PIO modes. If a controller is buggy or doesn't support a specified mode, the result can be data corruption or lost access to hard disks. The utility has nothing to do with partition tables, device file mappings, or filesystems per se.

13. C. SCSI chains must be one-dimensional—each after the other along a straight line. By using all three connectors on a SCSI host adapter, the configuration described creates a Y-shaped fork in the SCSI chain, which is very likely to cause data transfer errors. The device at each end of the SCSI chain should be terminated.

14. B. Because the cable and camera are the only constants in both tests, they're the most likely source of the problem. This isn't absolutely certain, though; software or interface hardware problems could exist on both test systems, thus misleading you in your diagnosis.

15. A, B. Both parallel and floppy ports are standardized on *x*86 hardware. SCSI host adapters and Ethernet adapters both come in many incompatible varieties. Linux includes drivers for most models of both types of device, but you must match the driver to the chipset used on each device.

16. B, D. Expansion slots are a limited resource that you must budget. If you run out of space, you must compromise by using another connection type or prioritizing the devices you have installed. It's impossible to insert a PCI card in an ISA slot or vice-versa. There are no adapters to make such uses possible.

17. B. Modern computers use a motherboard-mediated power circuit, and so they carry some current even when plugged in. You can get an electrical shock from certain circuits if you accidentally touch them even when the power's off.

18. B. Temperature-related problems can often be overcome by improving ventilation within the computer. Because kernel oopses are often caused by overheating CPUs, upgrading the heat sink and fan can often improve matters. Although kernel oopses can sometimes be caused by kernel bugs, the temperature-sensitive nature of the problem suggests that option C won't have any effect. Kernel oopses definitely are *not* normal. Hard disks that spin faster are likely to generate more heat than those that spin slower, so option A will most likely have no positive effect on the problem, and may make it worse.

19. A, D. Suspend mode is a low-power mode intended to save battery power on a laptop. It therefore shuts down components that don't need to be active when the system is not being actively used, such as the display, hard disk, and CD-ROM drive. The CPU and RAM continue to receive power, albeit at minimal levels, to maintain the working configuration.

20. B. Linux expects most devices, like Ethernet cards and hard disks, to remain available until Linux unloads the driver. PC Cards can be physically ejected by the user. This requires an extra software layer (Card Services) that helps the kernel adjust to the sudden loss of a device or its reappearance.

Chapter 9

Troubleshooting

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ 6.1 Identify and locate the problem by determining whether the problem is hardware, operating system, application software, configuration or the user.
- ✓ 6.2 Describe troubleshooting best practices (i.e., methodology).
- ✓ 6.3 Examine and edit configuration files based on symptoms of a problem using system utilities.
- ✓ 6.4 Examine, start, and stop processes based on the signs and symptoms of a problem.
- ✓ 6.5 Use system status tools to examine system resources and statuses (e.g., `fsck`, `setserial`).
- ✓ 6.6 Use system boot disk(s) and root disk on workstation and server to diagnose and rescue file system.
- ✓ 6.7 Inspect and determine cause of errors from system log files.
- ✓ 6.8 Use disk utilities to solve file system problems (e.g., `mount`, `umount`).
- ✓ 6.9 Resolve problems based on user feedback (e.g., rights, unable to login to the system, unable to print, unable to receive or transmit mail).
- ✓ 6.10 Recognize common errors (e.g., package dependencies, library errors, version conflicts).
- ✓ 6.11 Take appropriate action on boot errors (e.g., LILO, bootstrap).
- ✓ 6.12 Identify backup and restore errors.
- ✓ 6.13 Identify application failure on server (e.g., Web page, `telnet`, `ftp`, pop3, `snmp`).

- ✓ **6.14 Identify and use troubleshooting commands (e.g., `locate`, `find`, `grep`, `|`, `<`, `>`, `>>`, `cat`, `tail`).**
- ✓ **6.15 Locate troubleshooting resources and update as allowable (e.g., Web, man pages, howtos, infopages, LUGs).**
- ✓ **6.16 Use network utilities to identify network and connectivity problems (e.g., `ping`, `route`, `traceroute`, `netstat`, `lsof`).**

In the best of all possible worlds, everything works correctly all of the time. Unfortunately, we don't live in the best of all possible worlds; in our world, problems occasionally occur. Therefore, it's necessary that you understand something of how to troubleshoot a Linux system. This task begins with diagnosis: You must be able to localize the problem as belonging to any of several broad classes, then identify and diagnose it more precisely. After you've done this, you can move on to solving the problem. You can often do this by adjusting common configuration files that have been misconfigured. Other problems require working around buggy software or locating missing software. No matter what the problem and the solution, it's important that you understand various commands that can help you along the way.

Localizing the Problem

Usually, the first troubleshooting step you must take is to localize the problem—that is, to narrow down the general field in which the problem occurs. Likely problem areas include hardware, kernel, application software, configuration, and those that are user-created. Sometimes you won't be able to narrow the field to just one of these areas, but eliminating just one or two can be a huge help. You can generally narrow the field using the general symptoms of a problem, as described in this section.

Symptoms of Hardware Problems

Hardware problems can affect just about any aspect of system operation—the boot process, network operations, disk input/output, overall system reliability, and so on. Chapter 8, "Hardware Issues," discusses hardware problems in more detail, so you should consult it when you need to perform detailed hardware troubleshooting and problem-solving tasks.

Hardware problems can manifest themselves in either consistent or inconsistent forms. A consistent problem is one that's easily reproduced; for instance, if you can't bring up a network interface using the procedures described in Chapter 5, "Networking." An inconsistent problem is one that doesn't always occur, even when all conditions are identical; for instance, if the network interface comes up sometimes, but not other times. Although other problem classes can sometimes produce inconsistent problems, hardware malfunctions are more likely to produce these problems.

Before you declare that a problem is inconsistent and therefore begin focusing on hardware issues, you should be *sure* that the problem truly is inconsistent. Consistent problems sometimes appear to be inconsistent because some critical environmental condition has changed, such as the username used to issue a command or whether some program other than the one that's malfunctioning is running. Consider again the example of the network interface that won't come up—the interface might come up only after loading the appropriate kernel module. If you don't realize this but perform tests with and without that kernel module loaded, you might mistakenly believe the problem to be inconsistent.

A physical manifestation is another symptom of a hardware problem. For instance, you might hear different sounds coming from the computer than you typically do. A hard disk that doesn't spin up won't create a normal disk sound—it may be silent, or it may produce some other noise as the motors strain against some problem. Some software problems can create such symptoms as well, though, particularly when the software is supposed to cause hardware actions. A problem with tape backup software might prevent the tape drive from operating, for instance.

One type of physical manifestation that can be important in modern computers is heat. CPUs, hard disks, and other computer components all generate heat, and this heat can be damaging if it's not dissipated. The Lm_sensors package (`http://www.netroedge.com/~lm78`) can help you detect heat problems, particularly with your CPU. This program monitors the temperature that is reported by sensors that exist on most modern motherboards so

that you can see if the temperature has risen too high, and if it's correlated with a problem.

One particularly reliable sign of a hardware problem is the kernel oops, which indicates a processing error in the kernel. In some cases, kernel oopses translate into crashed programs, but in extreme cases they can crash the computer as a whole. Kernel oopses generate an error message containing the word `oops`, which is displayed on the text-mode console and (if it doesn't crash the system) logged in `/var/log/messages` or some other log file. Kernel oopses are almost always caused by either a kernel bug or a hardware problem (usually in the CPU or RAM, but sometimes in other components, like a hard disk controller). If you're running an experimental kernel, or even an old kernel in a stable series, you might try replacing your kernel with the latest stable kernel to see if that fixes the problem. If it doesn't, chances are good that the cause is in your hardware.

If a system had been functioning correctly but spontaneously develops system-wide problems or problems with a specific hardware device, the most likely explanations are a hardware problem or a configuration problem. Software, including the kernel, doesn't usually go bad spontaneously, although an upgrade might end up installing a version that contains a new bug. (One exception is known as *software rot*, which is rare and is usually caused by a filesystem error that causes corruption in a program or critical data file.)

Symptoms of Kernel Problems

One of the primary duties of the Linux kernel is to function as an interface between the hardware and the bulk of the software you run on the computer. Because of this, the kernel has unusually privileged access to the system's hardware. A bug in the kernel can manifest itself in ways that can be difficult to distinguish from hardware problems, such as system crashes, kernel oopses, unreliable operation of hardware, or an inability to use hardware.

Kernel bugs are less likely than are hardware problems to appear inconsistently, but such a manifestation is possible. (In truth, the bug would be consistent, but it would cause different symptoms depending upon some low-level factor you can't observe, like the value in a particular hardware register.) Kernel bugs are also less likely to produce physical symptoms, like changes in a computer's sound.

Fortunately, Linux kernel bugs are very rare, at least if you're using a stable release (one with an even second number, like 2.2.18 or 2.4.3). These stable kernels have been extensively tested and seldom cause problems. If you need to use a development kernel (one with an odd second number) to gain access to a particularly new driver or feature, you may run into kernel problems, however. Solving these problems may require upgrading to a more recent development kernel or downgrading to an older one.

Symptoms of Application Software Problems

Application software doesn't run with the sort of low-level access to the hardware that the kernel possesses, so problems with such programs are typically much less serious. These problems are usually restricted to just one program, as opposed to hardware and kernel problems, which often manifest themselves in many different programs. Typical application problems include the following:

Programs that fail to start Programs sometimes don't start at all. This can be caused by a bug in the program or an unmet dependency, as discussed in more detail in the upcoming section, "Package Dependencies and Conflicts."

Program crashes A program may start, but then crash. As with a failure to start, this can be related to a bug in the program or a failed dependency. Hardware and kernel problems also sometimes manifest themselves in this way.

Programs that consume too many system resources Sometimes a program runs out of control and begins consuming an inordinate amount of system resources—typically too much CPU time, RAM, or disk space. This is usually caused by poor design or a bug in the program. If the program has stopped responding but is still consuming resources, it may have hung, in which case, you may need to kill it. These conditions are discussed in the upcoming section, "Stopping, Starting, or Restarting Processes," and in Chapter 7, "Managing Partitions and Processes."

Programs that misbehave in program-specific ways Programs often malfunction in ways that are very program-specific. They may produce bad displays, corrupt data, and so on. Such problems can sometimes be caused by configuration errors, as well.

If an application develops problems even though you've not upgraded it or changed its configuration files, it may be a symptom of a break-in. Crackers sometimes replace critical applications, and such replacements may behave in a visibly different way than the originals. Software rot is another possible cause of such problems.

Application problems are often best dealt with by replacing the software, either with a newer (or sometimes an older) version of the software, or with a competing package. You might want to check troubleshooting resources (as discussed soon, in "Using Troubleshooting Resources") to locate information on the specific problem you're experiencing and learn whether newer or older versions of the package suffer from the same problem.

Symptoms of Configuration Problems

Both individual programs and Linux as a whole use configuration files, which may be set up incorrectly or in ways that are inappropriate for your system or your goals. When this is the case, the program or computer will misbehave in ways that may be difficult to distinguish from application problems, or sometimes even from kernel or hardware problems. Programs may crash, hardware may become unavailable, and in extreme cases, the computer may not boot or may crash some time after booting. (Configuration problems that cause the system to crash once booted are extremely rare, though.)

System configuration problems typically affect some subsystem, such as the ability to access a partition, start networking, use X, or print. Previous chapters of this book cover the relevant subsystems and contain information on proper configuration, and so they may help you determine what's going wrong in these cases. The upcoming section, "Configuration File Problems," also provides pointers.

User applications also utilize configuration files, and so they too can be affected by this class of problem. These files may be either system-wide (typically stored in `/etc`) or stored in individual users' home directories. If just one user has problems with a program, there's a good chance that the user's configuration file is the culprit. If all users have a problem, it's more likely associated with a system-wide configuration file or an application problem (a bug).

Symptoms of User Problems

Some problems aren't really with the computer; they're with the user—or more precisely, with user expectations. If a user expects a Linux system to function exactly like a Windows system, that user may come to you with "problem reports" that merely reflect the differences between the two systems. For instance, a user who ejects a floppy disk without first unmounting it may find that files on that floppy disk are corrupt or missing. This isn't a bug in Linux; it's a difference between how Linux and Windows handle disks.

If you are inexperienced with Linux, it can be difficult to separate user problems from other problem types. The best approach is to learn more about how Linux works, particularly in whatever area seems to be causing the problem. This book serves as a good starting point, and if you've read chapters in sequence, by now you should have a good idea of Linux's basic operating model.

Some user problems really are problems, but they fall into the category of configuration problems—a user may have misconfigured an application, which will then misbehave. The usual solution is to reconfigure the application, either by using GUI tools in the application or by editing the program's configuration file with a text editor. In extreme cases, you may need to delete the user's configuration file, which should restore the program to its default configuration, which the user will then have to adjust to restore any customizations.

Problem Identification

Ideally, you'll be able to use the preceding advice to localize a problem to just one or two areas, at least as a first approximation. Once you've done this, you can use log files, tools, and the symptoms of the problem the users report (or you experience) to help further identify the nature of the problem.

Sometimes, the nature of problem solving may require you to revise your original assessment of the problem. Initially, you might believe that a problem is related to the kernel, but later you may decide that it's a hardware issue, for instance. Even the most experienced diagnosticians occasionally make such mistakes.

Using Log Files to Identify Problems

The "Monitoring Log Files" section of Chapter 4, "Users and Security," introduced log files. In brief, Linux maintains various mechanisms that are used to record important information on the activity of the kernel and various servers and system utilities in log files. Most of these log files reside in the `/var/log` directory, and the most important log files on most systems are `/var/log/messages`, `/var/log/secure`, and `/var/log/syslog`. Chapter 4 describes how to locate both system log files and the log files for specific servers, for those servers that maintain their own logs.

You can use log files to monitor system loads (for instance, to determine how many pages a Web server has served), to check for intrusion attempts, to verify the correct functioning of a system, and to note errors generated by certain types of programs. To one extent or another, all of these functions can be used to identify problems. Here are a few examples of information that can be useful when you are troubleshooting:

Verifying heavy loads If a server is running sluggishly, log files may contain clues in the form of a large number of entries from the server. If a server has experienced a massive increase in the number of clients it handles or the size of the files it transfers, you may need to increase the server's capacity to restore good performance. Most non-server programs don't log their activities, though, so you probably won't be able to diagnose similar load problems caused by increasing workstation demands in this way. You'll likely have an idea that workstation load has increased in a more direct way, though, because the workstation users should know that they're running more or more resource-intensive programs.

Sometimes the logging action itself can contribute substantially to a server's CPU and disk input/output requirements. If a server is behaving sluggishly, try reducing its logging level (so that it records less information).

Intrusion detection Some system problems are related to the presence of an intruder. Crackers frequently modify your system files or utilities, thus affecting your system's performance or reliability. Their actions are sometimes reflected in log files. Even the *absence* of entries can sometimes be a clue—crackers often delete log files, or at least remove entries for a

period. You might not notice such log file discrepancies unless you examine the log files soon after a break-in occurs, however.

Normal system functioning If a system is misbehaving, the presence of and information in routine log file entries can sometimes help you pin down the problem, or at least eliminate possibilities. For instance, suppose your system is working as a DHCP server for your network, dishing out IP addresses to other systems, as described in Chapter 5. If your clients aren't receiving IP addresses, you can check the log file on the server. If that file indicates that the DHCP server has received requests and given leases in response, you can focus your problem-solving efforts on the clients.

Missing entries If you know that a program should be logging information, but you can't locate it, this may be evidence that the program is misconfigured or is not starting properly. In some cases, missing entries may indicate problems outside of the computer you're examining. For instance, suppose you configure Samba to log access attempts. If you can't access the Samba server from another system, you can check for Samba log file entries. If those entries aren't present, it could mean that Samba isn't running, that it's misconfigured, or that some network problem (such as a misconfigured router or firewall) is blocking access.

Error messages The most direct evidence of a problem in a log file is usually an error message. A log file entry might indicate an authentication failure, for instance, which should help you focus your troubleshooting efforts. (The user might or might not receive as informative a message as is recorded in the log file.) To improve this capacity, you can configure many servers and utilities to log more information than usual; consult the program's documentation for details.

Log files are most useful when you are diagnosing software problems with the kernel, servers, user login tools, and miscellaneous other low-level utilities. Information routinely recorded in log files includes kernel startup messages, kernel module operations, user logins, cron actions, filesystem mounting and unmounting, and actions performed by many servers. This information can reflect hardware, kernel, application, configuration, and even user problems.

Real World Scenario

Using *dmesg* for System Diagnosis

The dmesg command can be particularly useful for diagnosing certain types of hardware and kernel problems. This command displays the contents of the *kernel ring buffer*, which is a data structure that contains recent kernel messages. Many of these messages are logged to log files, but dmesg displays just the kernel messages. Immediately after you start the computer, you will see the messages in the kernel ring buffer scroll past on the screen at high speed as the computer boots. These messages contain potentially important information on your system's hardware and drivers—most of the information that drivers write to the kernel ring buffer concerns whether they are loading successfully, and what devices they're controlling (such as hard disks handled by EIDE or SCSI controllers).

For instance, suppose your computer has two network cards, but only one works. When you examine the output of `dmesg` just after booting (say, by typing **`dmesg | less`**), it should reveal information on the working card, and possibly on the one that's not working, as well. If there's no entry for the missing card, then chances are Linux hasn't detected the card because the driver is missing. If there is an entry for the card, then chances are some other aspect of network configuration is incorrect. You can search for specific information by using `grep`, as in **`dmesg | grep eth0`** to find lines that refer to `eth0`. This is most effective if you know that the entries for which you're looking contain certain strings.

The output of `dmesg` immediately after booting is so important that some distributions send the output of the command to a special log file (such as `/var/log/boot.messages`). If your distribution doesn't do this, you can do it yourself by putting a line like `dmesg > /var/log/boot.messages` in your `/etc/rc.d/rc.local`, `/etc/rc.d/boot.local`, or other late startup script. As the system operates normally, the kernel ring buffer will accumulate additional messages, which will eventually displace the boot messages, so storing them at bootup can be important.

Using System Status Tools to Identify Problems

Linux includes many configuration and information utilities that can help you diagnose a problem. Typically, these utilities return information on the current configuration, or they allow you to change that configuration. These tools are useful to the extent that you understand how the underlying system *should* be configured, but they don't normally return information that explicitly states that some feature is misconfigured. Examples of these tools include the following:

`setserial` This program displays or sets options for the RS-232 serial ports (typically `/dev/ttyS0` and `/dev/ttyS1`). If you type **`setserial -a /dev/`*`device`***, the utility displays an extended report, including the type of hardware, the port's speed, and the hardware resources used by the ports.

`ifconfig` This command, discussed in Chapter 5, is used to obtain information on or configure a network interface. You can use it to learn whether the device has an IP address, what interrupt request (IRQ) it uses, and so on.

`route` This command, also discussed in Chapter 5, displays or sets information on the computer's routing table, which it uses to send information to particular IP addresses. This information can be very important in diagnosing many network problems.

`df` If you suspect your hard disk is filling up, you can use this command, which is discussed in Chapter 7, to display the disk space that's used and available on all your partitions.

`fsck` Filesystem corruption can be a very serious problem, and Linux includes a tool to diagnose and correct it: `fsck`. This tool examines a partition and, optionally, fixes it. It's discussed in Chapter 7 and in the upcoming section, "Filesystem Problems."

`lpq` Printing problems can have several causes, and `lpq` will help you to eliminate some possibilities. This command, discussed in Chapter 8, displays all the jobs that are waiting to be printed. If a job you submit disappears after it has entered the queue, then chances are the printer is losing the job. If the job remains queued, then Linux can't find the printer to send the job on its way.

top This tool, discussed in Chapter 7, displays information about processes running on the computer. It's particularly helpful in spotting CPU-hogging processes because `top` displays processes sorted by the amount of CPU time they consume; the CPU hogs float to the top of the display.

These and other diagnostic tools are discussed throughout this chapter and earlier in the book. As a general rule, any tool that returns information on the system's status can be a useful diagnostic tool.

The `/proc` filesystem is another useful source of diagnostic information. This directory contains files and subdirectories that host information on the computer's configuration. Writing to certain files can change how the computer operates, and reading from files (using commands like `cat` or `less`) allows you to examine the system's configuration. Relevant `/proc` filesystem files have been discussed throughout this book.

Evaluating User Complaints

The symptoms of a problem, as experienced by users, can be an important source of diagnostic information. If you're trying to interpret others' problem reports, the challenge is often in extracting the true symptoms from inexperienced users' interpretations of them. Users may be imprecise or omit information that's actually critical to understanding a problem, such as the exact error message reported by a program. It's often helpful to investigate the problem yourself, to try to reproduce it and learn more about it.

If you have administrative privileges on a system, you can temporarily take on a user's identity by using `su`. First, log in as `root` or use `su` to acquire `root` privileges. Then, type **`su - username`**, where *`username`* is the username of the user who reports the problem. When you do this, you'll acquire that user's identity without having to provide a password. (Only `root` has this privilege.) You can then investigate the problem using the targeted user's configuration files, which are sometimes necessary for reproducing the problem.

As noted earlier, some user problems are caused by lack of understanding. For instance, if a user who's used to DOS types **`DIR`** at a Linux command prompt, Linux responds with a `command not found` error message. You'll have to educate your users about the Linux way of doing things, or you will need to customize your environment to conform to the users' expectations. For instance, you could create a script called `dir` or `DIR` that calls `ls` (or `ls -l`), so that users get something akin to the output they expect when typing these commands.

Some common problems users report include the following:

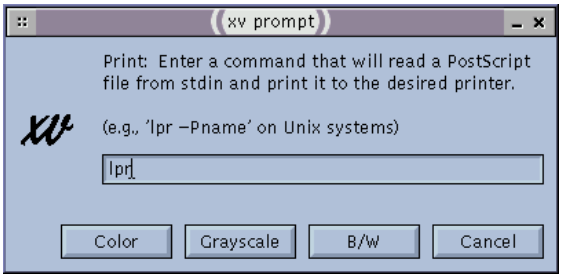
Login problems Users sometimes forget their passwords, or type them incorrectly. Remind users that their Linux passwords are case-sensitive. You might want to check if the password has expired, as described in Chapter 4. If it has, you can reenable it, ideally just for a day or two so that the user can change the password. If necessary, you can use `passwd` to give the user a new password.

File permission problems The Linux file permission system can sometimes be difficult for new users to understand. This isn't normally much of a problem if users work primarily in their own directories, but it can become an issue if users must collaborate on shared files. You'll have to educate users on the important Linux ownership and permission features, as described in Chapter 4, if this is the case.

Removable media problems Linux handles removable media very differently than does Windows, and this fact often causes consternation for new users. Many distributions configure their desktop environments to make Linux *appear* more like Windows, but Linux must still mount the removable media, unmount them when it's done, and so on. Be sure to point out any media unmounting tools provided by the desktop environment, or teach users to use the `umount` command. (You may also need to modify `/etc/fstab` to allow users to mount and unmount removable media.) The floppy disk is particularly likely to cause problems because users can eject a floppy on an *x*86 system without first unmounting it, and this can result in filesystem corruption. Linux can lock other media so that they can't be ejected, which is more likely to produce an immediate complaint or query but less likely to produce filesystem corruption on the media.

Printing problems If you've configured printing correctly, as described in Chapter 8, your users should be able to print, as well. Some users will experience problems, though, particularly when faced with text-based programs or commands that don't list printer destinations, such as the XV printing dialog box shown in Figure 9.1. You'll have to educate users about your system's printer names and how to list them. Be sure to note that there should *not* be a space between `-P` and the printer name if you use the BSD printing system.

FIGURE 9.1 Some Linux printing dialog boxes require the user to know the printer's name, which can be intimidating.



E-mail problems As with printing, e-mail should work for all users if you configure it correctly, as described in Chapter 5. This is particularly true if you use a local mail queue and run an SMTP mail server. If each user has a remote POP or IMAP account, you may need to instruct all your users in how to configure their mail clients to read mail.

Program errors Just about any user program may crash, corrupt data, or otherwise misbehave. Sometimes this will force you to replace the program, or advise users to try alternatives. You'll have to troubleshoot each of these on a case-by-case basis, though.

Shutdown problems Like most OSs released since the mid-1990s, Linux should be shut down using an explicit shutdown command. As described in Chapter 6, "Managing Files and Services," this command is `shutdown`, and it can normally only be run by `root`. Many distributions, however, include a shutdown option on their GUI login screens so that ordinary users can shut down the system. If workstation users should be able to do this, they must be told to use this option, rather than simply hitting the power switch on the computer. Shutting down improperly usually results

in a lengthy startup process, and it can result in serious filesystem corruption and data loss. If your users say they routinely see such symptoms, it's likely that they're not shutting down their systems correctly, or that there's a problem in the shutdown process.

Diagnosing Software Problems

There are several specific classes of software problems that deserve explicit discussion because they're more common than others, unusually serious, or particularly tricky to diagnose and fix. Some of these problems are very specific, such as filesystem problems and backup and restore difficulties. Others are more general, such as configuration file and server problems.

Filesystem Problems

Linux stores data on hard disks with the help of one or more filesystems, which are essentially complex data structures that provide a way for Linux to remember where on the disk a given file exists. In some sense, a filesystem is like a table of contents or an index in a book; it relates a filename to a specific location on the disk, just like the table of contents or index allows you to find information within a book. Unlike a book, though, most filesystems are dynamic structures, and in the process of handling data, they sometimes acquire errors. Linux provides the `fsck` program to locate, and optionally correct, these errors. This command is described in more detail in Chapter 7, so you should consult that chapter if you experience problems with your disk.

You should be alert to the recurrence of problems, should `fsck` report any. `fsck` often reports and corrects errors after a system crash or improper shutdown, but these errors are much rarer on routine runs of `fsck` (for instance, when `fsck` is forced because a disk has been mounted the specified maximum number of times). If you begin to see errors on a routine basis, it could be a symptom of a more serious problem, such as a driver bug, disk controller problem, or a disk that's starting to go bad.

A few filesystem problems can be corrected by remounting the filesystem. This is particularly likely for DOS and Windows FAT filesystems. Because DOS and Windows 9*x*/Me don't support Linux-style ownership and permissions, the FAT filesystem that they use doesn't support these characteristics. As a result, Linux fakes these features when mounting FAT filesystems. By default, Linux gives ownership of all files to the user who issues the `mount` command, and it gives only the owner write access to the partition. This can be a problem if you mount the partition automatically via `/etc/fstab` because this results in ownership by `root` with no other users able to write to the partition. You can overcome this problem by using the `umask=`*`value`*, `uid=`*`UID`*, and `gid=`*`GID`* options to `mount`, as described in Chapter 7. You can include these parameters in `/etc/fstab` to use them automatically when you reboot, and if you need to apply them immediately, you can unmount the partition and then remount it with these options.

If you try to unmount a filesystem but get a `device is busy` error, this means that some process has an open file on the device. It may not be obvious what process this is, particularly if the computer in question hosts many users. One tool that can be very helpful in this situation is `lsof`, which lists all open files. You can pipe the output of `lsof` through `grep`, searching on the name of the mount point directory. For instance, the following command locates open files on `/mnt/floppy`:

```
$ lsof | grep "/mnt/floppy"
less  27343  rodsmith  8r  REG  2,0  2215  174253 /mnt/
↳floppy/readme.txt
```

The first column lists the command (`less` in this case), the second column lists the process ID (PID), the third column shows the username, subsequent columns list additional information, and the final column shows the name of the file that's open (`/mnt/floppy/readme.txt` in this case). Once you've located the processes that are preventing you from unmounting the filesystem, you may be able to shut them down manually or close whatever files are open in the target directory. In a worst-case scenario, you can use `kill` to terminate these processes, but this could cause problems for whoever is using these programs.

Shells frequently prevent you from unmounting a filesystem. If you've used `cd` to move into the mounted filesystem, you won't be able to unmount it. Rather than exiting from the shell, you can use `cd` to move into another directory.

Configuration File Problems

Many problems in Linux can be traced to entries in configuration files. These files control many aspects of how a Linux system behaves, so an incorrect or non-optimal setting can cause problems that are major or minor, as well as blatant or subtle. There are at least as many potential problems as there are configuration files—some files control more than one aspect of a Linux system's operation. Chapter 6 covers many of Linux's configuration files in broad strokes, and other chapters cover some of them in greater detail. Some of the files that more commonly cause problems include the following:

`/etc/lilo.conf` Unlike most configuration files, this one isn't read at every system startup; instead, it controls the operation of the `lilo` utility, which writes Linux's boot loader, as described in Chapter 3, "Software Management." LILO problems turn up at boot time, as discussed in "Handling LILO Boot Errors," later in this chapter.

`/etc/inittab` This file controls the initial startup of the computer. The most common problem that's related to this file is if the system fails to start X, or starts it when you don't want it started. Other problems might include difficulties with text-mode local or remote logins and very serious system startup failures that aren't kernel-related. Chapter 6 discusses this file in detail.

`/etc/modules.conf` This file controls the automatic loading of kernel modules. If you can't use a hardware device without manually loading appropriate kernel modules, edit this file to work around the problem. Chapter 6 discusses this file's format.

`/etc/fstab` Linux automatically mounts partitions according to specifications in this file. You can also configure certain filesystems to be mountable by users by editing this file. Therefore, if a partition isn't mounting correctly, editing this file may correct the problem. Chapters 6 and 7 discuss `/etc/fstab` and its options.

`/etc/passwd` This file stores user account information. (An auxiliary file, `/etc/shadow`, stores passwords on most Linux systems.) You can correct many user account problems by editing these files, but text-mode and GUI utilities provide interfaces to these files that are easier to use and less error-prone. Chapter 4 discusses these issues in more detail.

`/etc/hosts` There are several different ways to translate between IP addresses and hostnames, one of which is via entries in this file. Normally, this file's contents are largely irrelevant, but in a few cases, the system's boot process or even the launching of certain programs may be delayed if IP addresses and hostnames aren't properly mapped. The solution is to create entries in this file for `localhost` (with an IP address of 127.0.0.1) and your network hostname (with its associated IP address). Place the IP address first on the line, followed by the hostname.

`/etc/printcap` On distributions that use the BSD or LPRng printing systems (which is most Linux distributions in 2001), the `/etc/printcap` file controls printer definitions. This is the first place to look if printing doesn't work at all, although you may eventually need to look elsewhere. Chapter 8 covers printer configuration in more detail.

`/etc/X11/XF86Config` This file (which sometimes resides in `/etc` rather than `/etc/X11`, and is sometimes called `XF86Config-4`) controls XFree86 configuration. If X won't start or you're dissatisfied with your resolution, color depth, or refresh rate, this is the first place to look. (If X doesn't start when the computer boots, but it does start when you log in and type **`startx`**, look to `/etc/inittab` instead.) Chapter 2, "Installing Linux," discusses X configuration in more detail.

If X starts but you have problems with your desktop environment, adjusting XF86Config won't do any good. Desktop environment and window manager configuration can be adjusted on a system-wide or user-by-user basis, as discussed in Chapter 2.

Crontab files The cron utility, discussed in Chapter 7, uses several different configuration files, such as `/etc/crontab`, files referred to by `/etc/crontab`, and files for individual users in `/var/spool/cron`. If something strange is happening at a regular time, these are the places to begin looking because the problem may be caused by an errant cron job.

Many individual programs can be misconfigured as well, usually by configuration files that may be global (typically stored in `/etc`) or user-specific (in the user's home directory). If a program is misbehaving, particularly just for some users, you should read the program's documentation to learn where its configuration files reside. You can then adjust the files as required, or in the case of personalized files, delete them from the affected users' directories. This action should restore the default values.

Server Software Problems

Network servers can be tricky to debug because they're non-interactive—you don't click an option and see the program produce an error message. In most cases, you can learn a great deal about a server problem by examining the system log files, as discussed earlier, in "Using Log Files to Identify Problems."

You can use the `tail` command to examine the last few lines of a log file just after a problem occurs. For instance, if a server isn't responding, you can try connecting to the server from another machine, then you can type **`tail /var/log/messages`** to see if the server has logged any messages concerning the connection attempt. (You may need to use a different log file for some servers.)

Some common server problems include the following:

Failure to start Servers may fail to start up. This problem may be caused by an incorrect super server configuration (in `/etc/inetd.conf` or `/etc/xinetd.d/`*`servername`*), or by an incorrect SysV startup script configuration. The usual problem with the latter is a startup script name for a given runlevel that causes the server to stop rather than start. Both issues are discussed in more detail in Chapter 6.

Failure to respond Sometimes a server is running but doesn't respond to queries. This may happen because a firewall, TCP Wrappers, `xinetd` configuration, or other access control mechanism is blocking the client request. It's also possible that the server itself implements such a block, so you should check the server's configuration file. Chapter 5 covers all these issues.

Slow responses Sometimes a server will respond, but it will do so very slowly. If you're receiving such complaints, you should first try to track down which clients are experiencing slow responses; the problem may be caused by slow routers or overloaded Internet backbones, for instance. It could also be that the server's network connection, CPU, RAM, or hard disk is inadequate for the task. Finally, some servers are deliberately slow at certain tasks. For instance, some mail servers pause for several seconds if you enter an invalid destination address; this is a way of slowing down spammers who try to test a system to see if they can use it as an open relay.

Unexpected responses If a server doesn't generate the replies that you expect, that's usually an indication that the server's not been configured correctly. For instance, you might not be able to perform an anonymous login to an FTP server, in which case you should examine the FTP server's configuration file. If you don't see the Web pages you expect from a Web server, the cause could be in the configuration file, or you may have put the Web pages in the wrong location.

Crashing server A server program may respond but then crash or disconnect users. This problem is fairly rare in major servers, but it does still occur sometimes, particularly in unusual servers. The cause can be just about anything—a bug, an incorrect configuration, or even a hardware failure.

Ultimately, troubleshooting a server is much like troubleshooting an ordinary application, except that you'll find most error messages in the server's log file rather than displayed directly on the screen.

Backup and Restore Problems

Backing up a computer, as described in Chapter 7, is an important undertaking for the safety of your system. Without a backup, you may lose data, or at least experience costly downtime, should any of several things happen. These include a system compromise, hardware theft, hardware failure, and even human error. Unfortunately, backups are not themselves immune to problems. You should be prepared for these, or at least be familiar with some problem causes and solutions.

The single worst time to discover a problem with your backup procedure is when you need the backup in a time-critical emergency situation. Therefore, you should test your backups and your emergency recovery procedures as best as you can before an emergency occurs.

Because most backup systems for Linux use tapes, this section focuses upon tape backup procedures. Some of these problems and solutions apply to other media, such as CD-R or CD-RW devices, but others don't. One of the failings of magnetic media in general is that they can degrade with time, particularly if they're heavily used. For this reason, you should probably replace tapes after they've been used about 100 times, or at the first sign they

may be becoming unreliable, such as unexplained failures during the verification phase of a backup procedure.

Some backup and restore errors occur when backing up a system—for instance, you might be unable to access the tape drive at all. Other problems occur at restoration time, such as errors when recovering data. The most common problems in both categories include the following:

Driver problems Backup hardware, like all other hardware, requires driver support. Most EIDE/ATAPI and SCSI tape devices require support for the underlying controller or host adapter and support for tape devices. These drivers, if compiled as modules, must be loaded when you perform a backup. If the drivers aren't present, you'll be unable to access the tape device.

Tape drive access errors As described in Chapter 7, Linux uses device files like `/dev/ht0` and `/dev/nst0` to provide access to a tape drive. You should be able to access these devices for both reading and writing as the user who performs the backup. This user is normally `root`, so access shouldn't be a problem. If you get a `no such device` error when you try to access the tape device, you've probably entered the wrong device filename, or your drivers may not be loaded.

File access errors In order to back up a computer, the user who runs the backup must have full read access to all the files. For this reason, you normally run a backup as `root`. (A non-`root` user can back up most system files, but perhaps not other users' files, and certainly not highly protected files like `/etc/shadow`.) If you only want to back up your own user files, you can use an ordinary account, but that account must have full read/write access to the tape device file. Restoring files requires full read/write access to all affected directories—again, `root` is the usual choice for performing this task.

Media errors In many ways, the worst nightmare for backups is if the tape develops errors. The problems might even occur after verification has succeeded, particularly if the tape has been in storage for a while. To reduce the risk of this problem, you should keep tapes stored at room temperature (*do not* leave them in a hot car, for instance). Keep at least two backups of the same computer. As described in Chapter 7, don't use `gzip` or `bzip2` compression in conjunction with `tar`, because in the event of an error, these will render all of the backup after the error unusable. Most

tape drives include compression features that don't suffer from this problem; with them or with uncompressed backups, a single error is likely to damage just a few files (possibly as few as one).

Files not found One common problem when using `tar` or similar utilities from the command line is an inability to restore specific files. Unless overridden with the `-P` (`--absolute-paths`) parameter, `tar` stores files *without* the leading `/`. Therefore, you *should not* include the leading `/` when restoring specific files or directories. (This also means you should change to the root directory when doing restores.) If you don't know the exact name of a file, you can recover it by using `tar`'s `-t` (`--list`) command, but this can take some time. Some more sophisticated backup packages include an index of files on a tape, along with point-and-click means of selecting these files if you need to restore just some, which can help reduce the chance that you'll mistype a filename and therefore seek through an entire backup without finding the file.

To head off restore-time problems, it's important that you verify your backups. This can usually be done using a backup-time verify option that automatically performs a check immediately after backing up. (This may return errors for some files that have legitimately changed between the backup and verify passes.) Many tape drives include a separate read head that verifies data immediately after it's written. This feature can save time, but it's still wise to at least occasionally run a verify pass even with such drives, in case data are corrupt when they reach the drive. (I once encountered precisely this problem—a three-way interaction of a tape drive, a Zip drive, and the SCSI host adapter caused data corruption on backups.)

After restoring data, you may want to verify the information against any summary information you have. For instance, if you restore an entire computer, typing **`rpm -Va`** will verify restored RPM packages against the RPM database. (Some packages will have legitimate deviations, such as changed configuration files.) If you've installed Tripwire, it can be used for the same purpose, although Tripwire will probably not check as many files.

Diagnosing Network Problems

Linux systems often operate on networks, and networking problems involve issues that don't arise with local problems. Network cabling,

interactions with other systems, router problems, and more can all crop up. Unfortunately, it's not always obvious when a network problem is related to your local configuration and when it's something that affects more systems. Even if a problem only occurs on your Linux system, it's possible that the cause lies elsewhere. For instance, a cable might be bad, or a server might contain a bug that manifests itself only with Linux clients. Your problem-solving task begins when you review your configuration; after you have done this, you can use more advanced diagnostic tools to localize the problem.

Reviewing Your Network Configuration

Many network problems are the result of an incorrect network configuration. Therefore, it's best to review these settings before proceeding with more involved troubleshooting. A few commands and procedures are particularly helpful in verifying that your network settings are correct. If you configured your IP address using static numbers provided to you by a network administrator, you should also double-check that those numbers are correct; if you wrote them down incorrectly, or if the administrator gave you the wrong numbers, there's a good chance that your network configuration won't work.

If you use DHCP to configure your network settings, you should do whatever you can to verify that DHCP is functioning correctly. Try using `ps` to verify the existence of a `dhcpcd`, `dhclient`, or `pump` process, depending upon which your distribution uses. For instance, you might type the following command:

```
$ ps ax | grep dhc
  340 ?        S      0:00 /sbin/dhcpcd eth0
```

This output confirms that `dhcpcd` is running and is bound to `eth0`. Some DHCP clients provide debugging information in log files or some way of querying the system for information on leases. For instance, `pump` has a `-s` option that provides information on the lease—**`pump -s`** returns the IP address, netmask, DHCP server address, and so on. You may not be able to confirm that these values are valid, but this command at least lets you know that they've been assigned to the computer and that because of this the DHCP client worked to a minimal degree—or that it did not.

Whether you use a static IP address or a dynamic system like DHCP or PPP, you can verify basic interface functioning with `ifconfig`, as described in Chapter 5. Typing **`ifconfig eth0`**, for instance, returns information on

the `eth0` (first Ethernet) interface. Check that the IP address and network mask are what they should be, or at least that they exist if they're assigned via DHCP or PPP.

If your computer can communicate with local systems but not computers on other networks, it's possible that you've specified the wrong gateway (aka router) address. You can check this detail with the `route` command, thus:

```
$ route -n
Kernel IP routing table
Destination Gateway   Genmask       Flags Metric Ref Use Iface
10.19.1.0   0.0.0.0   255.255.255.0 U     0      0     0 eth0
127.0.0.0   0.0.0.0   255.0.0.0     U     0      0     0 lo
0.0.0.0     10.19.1.1 0.0.0.0       UG    0      0     0 eth0
```

This final line, with a Destination of 0.0.0.0 (called `default` if you omit the `-n` option to `route`), shows the gateway system—10.19.1.1 in this example. Verify this address against what it should be. You may also want to check that there aren't any extraneous routes. A system with one network card is likely to list three routes—one for the local network (the 10.19.1.0 route in the preceding example), one for the localhost network (127.0.0.0), and one for the default route. A system with multiple network cards will have more routes, and one with no Internet connectivity (just a local network connection) might have just two, without a default route. On rare occasion, a system with just one network card may have more than three routes, but such configurations are rare.

If you can access computers via IP address but not by name, chances are your name server configuration is at fault. Check `/etc/resolv.conf`. It should contain one or more lines like the following:

```
nameserver 10.19.1.171
```

Each of these lines should contain the name of one DNS server computer, as provided by your network administrator or dynamic IP address system (DHCP or PPP). Verify that these addresses are correct, and if they're not, edit the file appropriately. If there are any stray addresses listed, remove them. `/etc/resolv.conf` may hold up to three `nameserver` lines.

DNS servers, like other servers, occasionally develop problems. DNS problems might therefore be a result of network problems outside of your control. If you believe this is the case, contact the administrator responsible for the DNS servers to report the problem.

Localizing the Source of the Problem

In addition to `ifconfig`, `route`, and diagnostic features of some DHCP clients, Linux provides several useful network diagnostic tools. The most basic of these is `ping`, which is described in the "Network Diagnostic Tools" section of Chapter 5. This tool tests basic connectivity between your computer and another that you specify. If you can ping another computer, chances are you can use other networking protocols to reach it, as well.

Be sure to test basic connectivity using both IP addresses and hostnames. If the former works but not the latter, your system may have a DNS configuration problem, as described earlier, or your network's DNS servers may not be working correctly.

When you are debugging basic connectivity problems, try pinging several different systems:

- Pinging the `localhost` address (127.0.0.1) verifies that the most basic Linux networking tools are operating. Because this address corresponds to Linux itself and isn't associated with any specific network hardware, it should always work.
- You should be able to ping your own IP address on your network interface hardware. (Use `ifconfig` to obtain this address, if you don't know what it should be.) If this test fails, it most probably indicates a hardware failure, driver problem, or failure to configure the IP address.
- Pinging other computers on your local network tests some of the low-level network hardware (such as the wires leading to the network connector); your network cabling; and any hubs, switches, and similar devices on your local network.

- You should be able to ping systems on the Internet at large, if your network is connected to the Internet. This tests the proper functioning of your router and others on the Internet.

Some computers are configured to ignore pings. As a result, it's possible that you'll be able to ping some computers but not others.

If you can ping some hosts but not others, you can use the `traceroute` command to localize the problem. This command sends three packets to each router between you and the destination system, and it records the time it takes to receive a response. This is much like what `ping` does, except that `traceroute` performs this test for each system between you and the destination. The result is that you should be able to spot the source of a breakdown, either because it doesn't return all three packets or because it shows a huge increase in the round-trip time. For instance, here's the result of running `traceroute`:

```
$ traceroute -n 128.197.153.100
traceroute to 128.197.153.100 (128.197.153.100), 30 hops
↳max, 40 byte packets
 1  66.92.68.1  21.187 ms  16.525 ms  16.385 ms
 2  63.251.141.219  14.609 ms  16.609 ms  16.662 ms
 3  63.251.128.6  16.551 ms  16.629 ms  16.661 ms
 4  63.145.1.93  16.667 ms  15.842 ms  17.348 ms
 5  63.145.1.78  16.591 ms  15.839 ms  15.892 ms
 6  128.197.254.62  16.638 ms  17.333 ms  15.866 ms
 7  128.197.153.100  16.617 ms  16.699 ms  16.706 ms
```

Omitting the `-n` parameter causes `traceroute` to display each intermediate router by name rather than by IP address. This example shows no problems. If the times (the values expressed in milliseconds [ms]) increase suddenly between a pair of routers, or if one or more doesn't respond (which `traceroute` indicates by showing an asterisk rather than a response time), then that router may be having problems. It may be overloaded, or it may be connected in some inefficient way to the preceding router. The same is true for the final destination system.

You're not likely to obtain useful information from `traceroute` on any connection that doesn't involve routers. For these local connections, `ping` will tell you whether basic connectivity exists; `traceroute` doesn't add any new information.

Another useful diagnostic tool is `netstat`, which can display many different types of network configuration information. When fed the appropriate information, `netstat` can be used in place of `ifconfig` or `route`, or it can be used to display information on the programs that are bound to specific ports. Chapter 5 discusses `netstat` in more detail.

Handling LILO Boot Errors

One of the first pieces of code a Linux system runs is the Linux Loader (LILO). LILO is the boot loader that takes over from the BIOS, loads the Linux kernel into memory, and turns the computer over to the kernel. Unfortunately, this task is much more difficult than it sounds at first. The *x*86 BIOS was designed in days when 40MB hard disks were huge and 32-bit programs were fantasies (at least, on desktop computers). As the technologies that go into computers have evolved, so has the BIOS. The result works remarkably well, given its peculiar mix of requirements; but the combination of the BIOS, the extremely variable *x*86 hardware market, and LILO's own needs means that LILO sometimes doesn't behave properly. These problems can be extremely frustrating because they mean that the system won't boot. LILO's error codes are also extremely cryptic. Understanding these problems and knowing how to work around them can help you turn a cryptic error message into a booting system. Before proceeding with this section, you should review "Configuring Boot Loaders" in Chapter 3.

When installed on non-*x*86 architectures, Linux doesn't use LILO. These systems use other boot loaders, some of which are similar to LILO. (SuSE on PowerPC calls its boot loader LILO, but it's really derived from a PowerPC boot loader called `ybin`.) Details of non-*x*86 boot loaders differ from those of LILO, so you should consult documentation for your platform's boot loader.

LILO Boot Error Codes

When LILO can't boot a system, it displays an error code. Unfortunately, these codes are notoriously cryptic. Table 9.1 summarizes the LILO error codes, so you can parse them and stand some chance of correcting the underlying problem.

TABLE 9.1 LILO Error Codes, Their Meanings, and Possible Resolutions

Error Code	Meaning	Possible Resolution
(none)	LILO hasn't loaded.	LILO may not have been installed, or it may have been installed in the wrong location. Check the boot line of `/etc/lilo.conf`, adjust it as necessary, and type **`lilo`** to reinstall LILO. If LILO's installed on a partition, you may have to make it the default boot partition using DOS's `FDISK` or Linux's `fdisk`.
`L` *`error-code`*	The main part of LILO has booted, but it can't locate the second-stage boot loader, `/boot/boot.b`.	The *`error-code`* is a 2-digit code delivered by the BIOS. These codes are detailed in the LILO documentation. Most indicate a hardware error, such as a bad hard disk. Some indicate a disk geometry problem, in which the BIOS and LILO don't agree on how to treat disk addresses. Such problems can often be overcome by changing disk geometry options in the BIOS or by adding or removing such options in `/etc/lilo.conf`.
`LI`	The main part of LILO has booted and loaded the second-stage boot loader, `/boot/boot.b`, but this second-stage loader won't run.	As with the `L` *`error-code`* condition, these problems frequently indicate disk geometry mismatches, which may require you to adjust geometry settings in the BIOS or `/etc/lilo.conf`.
`LI1010...`	LILO has booted, but it can't locate your kernel image.	This problem is usually caused by installing a new kernel over an old one and failing to rerun `lilo`.

TABLE 9.1 LILO Error Codes, Their Meanings, and Possible Resolutions *(continued)*

Error Code	Meaning	Possible Resolution
LIL	The second-stage loader, /boot/boot.b, has loaded and run, but it can't read the information it needs to work.	This problem is usually caused by failing hardware or a disk geometry mismatch. If it is the latter, you may be able to fix it by adjusting BIOS disk geometry options or by setting or removing such options in /etc/lilo.conf.
LIL?	The second-stage boot loader, /boot/boot.b, has been loaded at an incorrect address.	This problem can be caused by moving /boot/boot.b without rerunning lilo or by a disk geometry mismatch.
LIL-	The disk descriptor table, /boot/map, is corrupt.	This problem can be caused by moving /boot/map without rerunning lilo or by a disk geometry mismatch.
LILO	The program has loaded and run correctly.	If Linux fails to boot at this point, the problem is most likely with the kernel, its drivers, or system configuration files.

Several of these problems relate to LILO's inability to read files that have changed or been moved. One critical thing to remember about LILO is that it does *not* automatically recover data from changed configuration files; you *must* type **lilo** after making any changes in order for those changes to take effect. In this context, "changes" means both edits to configuration files and moving or replacing files, such as kernel files. If you fail to do this and encounter problems as a result, boot Linux in some other way, as described shortly, and then type **lilo** to correct matters.

Real World Scenario

Radical Surgery to Correct Disk Geometry Problems

There are several different ways to address data on a disk. Two popular methods are logical block addressing (LBA) mode and cylinder/head/sector (CHS) mode. The former uses a single number to specify a disk address, and the latter uses a triplet of numbers. The *x*86 BIOS and EIDE drives have traditionally used CHS mode, although some BIOS extensions and SCSI disks use LBA mode. Translating between the two and between different types of CHS addressing causes many problems on *x*86 systems.

When a disk is moved from one computer to another, it's not uncommon to find that the old computer assigned one CHS geometry but the new computer wants to use a different geometry. The result of this mismatch can be an inability to boot with LILO, and sometimes even data loss. In such cases, the best solutions often involve backing up the data, wiping the disk's partition table and boot sector, and starting from scratch. To do this, you can use `dd` to fill the disk's boot sector with zeroes:

```
# dd if=/dev/zero of=/dev/hda bs=512 count=1
```

You should substitute the correct device identifier for `/dev/hda`, of course. *This procedure will wipe out all partitions on the disk!* If you don't need to keep data when you are moving a disk from one computer to another, it's not a bad idea to perform this procedure as a matter of course.

Non-LILO Boot Techniques

If LILO fails, you may need some other way to boot the computer. One option may be to use a separate emergency disk set, as described in "Fixing LILO from an Emergency Boot System" and "Using an Emergency Disk Set." In many cases, though, a simpler solution is to use an alternative method of booting your current system. Options for doing this include the following:

`LOADLIN` This is a DOS program that boots Linux. `LOADLIN` comes with most Linux distributions, usually in a directory on the installation CD-ROM called `dosutils`. To use `LOADLIN`, you need a DOS boot floppy or

partition, a copy of LOADLIN.EXE, and a copy of your Linux kernel. Boot DOS and type **LOADLIN *VMLINUZ* root=/dev/*rootdevice* ro**, where *VMLINUZ* is the name of the kernel and /dev/*rootdevice* is the name of your root partition, such as /dev/hda3 or /dev/sda7.

Raw kernel on a floppy If written directly to a floppy, a raw Linux kernel is capable of booting a computer. Copy the kernel to the floppy with the command **dd if=*vmlinuz* of=/dev/fd0**, where *vmlinuz* is the kernel filename. To boot, insert the floppy disk in a system configured to boot from the floppy and turn on the power. For this to work, though, the kernel needs to be configured with information on the location of your root partition. This should happen automatically if you build your own kernel, but if not, you need to issue the command **rdev /dev/fd0 /dev/*rootdevice***, where /dev/*rootdevice* is your root device partition. Disks that hold a raw kernel don't have filesystems, so they look like they're unformatted in DOS or Windows, and can't be mounted in Linux.

LILO on a floppy You can install LILO to a floppy. This will result in a quicker boot than using LOADLIN from a floppy or a raw kernel on a floppy since the kernel will still reside on the computer. This approach won't get around problems caused by moving kernels or other files in /boot, however. To install LILO on a floppy, edit /etc/lilo.conf so that the boot line reads boot=/dev/fd0; then type **lilo**. (You'll need to edit this back if you want to subsequently install LILO on a hard disk.)

As a general rule, LOADLIN is the most flexible of these techniques since you can create or modify a boot disk using DOS—say, by using a default kernel from a Linux installation CD-ROM. A raw kernel floppy is an easy way to boot because, once configured, you don't need to remember the boot partition; but by the same token, if you change your system's configuration and forget to update the emergency disk, you can't boot the changed configuration using that disk. Using LILO on a floppy is the least useful in getting around a malfunctioning LILO on disk, but it can be useful in some situations. Some people prefer this approach to a regular disk-based LILO, in fact, particularly for dual-boot computers; these people configure the hard disk to boot directly to the non-Linux OS, and then they insert the LILO boot floppy when booting Linux.

Fixing LILO from an Emergency Boot System

If you have an emergency disk set, as described shortly, you may need to recover a LILO system on hard disk using the emergency set. One possible approach is to create a `LOADLIN` boot procedure using the hard disk's kernel, and then boot using it. A more direct approach, however, is to use the emergency system's copy of LILO to do the job. To do this, you must configure `/etc/lilo.conf` in some specific ways:

- You must mount the regular disk root (`/`) partition somewhere in the emergency system—say, at `/mnt/std`. If your `/boot` directory resides on its own partition, you should mount it instead of or in addition to the root partition.
- You must adjust all references to `/boot` or other files so that they point to the regular partition. For instance, change `/boot/vmlinuz` to `/mnt/std/boot/vmlinuz`.
- Configure the kernel images and other boot options as you would normally. For instance, the `boot` and `root` options should point to your regular hard disk or partitions on it.

With any luck, when you type **`lilo`**, the boot loader will install itself normally. This isn't guaranteed, though. If your emergency system uses a different version of LILO than you have installed on your regular system, the version mismatch may cause LILO installation to fail. In such cases, it's usually easiest to boot using `LOADLIN` or the like and reinstall **`lilo`** using the regular system tools.

Using an Emergency Disk Set

It's sometimes necessary to perform recovery operations on a computer that won't boot for reasons other than a LILO problem. For instance, the `/etc/fstab` file that maps partitions to filesystems might be corrupt, or changes to a startup script might cause the system to lock up or shut down before the computer has finished booting. You'll also need to do this if your hard disk fails and you need to replace it and recover your system from a backup. In these cases, you need to be able to boot a Linux system without using the main partition. This is the role of an emergency disk. Such disks come in many forms, from single-floppy distributions to complete Linux

installations stored on high-capacity removable disks or CD-ROMs. Knowing how to locate or create such a disk is a vital troubleshooting skill, and knowing what tools are on such disks is important when the time comes to use one.

Locating a Ready-Made Emergency Disk

There are several sources of ready-made Linux emergency disks. These can often be downloaded from the Internet, and they require little or no configuration to use. Examples include the following:

Your distribution's installation media Most distributions include some sort of emergency disk system. These are sometimes accessed as an option when you boot the installer. For instance, in Red Hat Linux, you type **`linux rescue`** at the installer's `lilo:` prompt. Some distributions include separate boot images for this purpose; consult your documentation for details.

Tom's Root/Boot Disk This distribution's official name, tomsrtbt, is short for "Tom's floppy which has a root filesystem and is also bootable," but an intermediate-length name for it is "Tom's Root/Boot Disk." It is a complete, if small, Linux system that fits on a single 3.5-inch floppy disk. It comes in packages for both Linux and DOS, so you can create the system from DOS even if that's all you have available in an emergency. Read more about it at `http://www.toms.net/rb`.

μLinux This is another single-floppy distribution that's similar in concept to Tom's Root/Boot Disk . You can learn more at its Web site, `http://mulinux.nevalabs.org`.

ZipSlack Slackware (`http://www.slackware.com`) produces a version of its distribution that installs in slightly under 100MB, and can boot from a FAT partition. This distribution can fit on a small DOS partition or a removable disk like a Zip or LS-120 disk. It's more complete than a single-floppy distribution, and it can be more easily customized with commercial backup tools, custom kernel drivers, and so on.

Demo Linux Demo Linux, `http://www.demolinux.org`, is a complete Linux distribution on CD-ROM. To use it, you'll need to create a CD-R from a 650MB file (instructions for doing this are on the Demo Linux Web site), so you should have a fast Internet connection or a lot of patience. Demo Linux is unusually complete; you can even run X using it on most video hardware.

SuSE Evaluation SuSE (`http://www.suse.com`) makes an evaluation version of its OS available. The SuSE evaluation CD-ROM is similar to Demo Linux in many ways, such as its size and support for GUI operation.

This is only a sampling of tiny Linux distributions; you can find more at `http://www.linux.org/dist/english.html`; the last half of this Web page lists small and specialty distributions. The Tom's Root/Boot Disk page also provides links to some other micro-distributions.

Creating a Custom Emergency Disk

Chances are one of the standard emergency disks will suit your purposes. If not, you can create a custom disk. This task varies in difficulty depending upon how you approach it. The simplest method is generally to modify an existing emergency disk to suit your purposes. ZipSlack can be particularly good for this because it's very much like a normal Linux distribution (albeit one missing many creature comforts, like X). You can recompile its kernel (or compile a kernel for it on a regular distribution), add tools you need, and so on. On a 100MB Zip disk, ZipSlack has a small amount of space for additions, but you can strip away some programs if you don't need them. If you have a larger disk from which to run it, such as a 250MB Zip disk, you have more freedom to expand the installation.

If you modify a tiny distribution like Tom's Root/Boot Disk, you'll need to be very careful in how you proceed. These distributions often use program files that have been very carefully optimized for space, so replacing them or adding more programs can exceed the space available on the disk. CD-based emergency systems aren't easily modified because of their read-only nature; but if you mount such a system in a normal distribution, you can copy its files to your own hard disk, customize them, and burn a new CD-R based on your changes.

Although creating a custom emergency disk entails additional effort, sometimes it's necessary. This is most likely to be the case when your system has unusual hardware that requires special drivers, or uses filesystems, networking features, or other configurations that aren't supported in the common emergency disks.

Emergency Disk Recovery Tools

No matter what type of recovery disk you use, it should have certain tools. For the most part, these are standard Linux programs; but in some cases you

may need a more exotic tool. Examples of programs you should have on any emergency recovery disk include the following:

Drivers You'll need drivers for any necessary hardware and filesystems you intend to support. These can be a challenge for a single-floppy distribution because they consume precious disk space and different computers' needs aren't identical. This challenge is particularly acute if you use an unusual SCSI adapter and SCSI hard disks or some other unusual but vital hardware.

A text editor Most emergency systems include at least Vi (discussed in Chapter 7). Vi's small size makes it a good choice for this role. Larger systems often include other editors, as well.

Disk utilities Your emergency system should include utilities that can be used to prepare a disk to support a Linux system, such as `fdisk` and `mkfs`. A copy of `fsck` can repair damaged filesystems (but not failing hardware). If you use unusual filesystems, be sure you have the necessary support utilities for them.

Backup software Most emergency systems include `tar` at a minimum, and often `restore` (the restore-time companion to `dump`), `cpio`, and perhaps others. If you use commercial backup software, you'll need to add it to your emergency system. (If the software doesn't fit on your emergency system's main disk, you may be able to store it on a separate floppy and mount that floppy independently.)

Network software If you intend to access the network (say, to restore data via a network backup system), you'll need network hardware drivers and one or more network client or server packages. These can quickly consume the space available on a floppy rescue disk, but some are smaller than others. Mounting an NFS export, for instance, requires only the normal `mount` command and appropriate kernel NFS support.

Some recovery disk methods require unusual features to access the recovery disk itself. For instance, if you're using a parallel-port Zip drive along with ZipSlack, you must boot from a kernel stored on a boot floppy. That kernel must include support for parallel-port Zip drives. As of the 2.4.3 kernel, this works for parallel-port Zip drives but not for USB Zip drives. This may be fixed in the future, of course.

Stopping, Starting, or Restarting Processes

Sometimes programs misbehave themselves. In such cases, you may need to kill the process or restart it. Programs also sometimes crash, in which case you may need to start the program up again. Some of these issues have been covered elsewhere, such as in Chapter 7, but there are some special troubleshooting considerations when managing processes.

When to Stop, Start, or Restart a Process

As discussed in Chapter 7, one common reason for stopping a process is if that process has hung—that is, if it's become unresponsive to its normal forms of input. Hung processes do nobody any good, and they can do harm by chewing up CPU time unnecessarily. You should try to be as sure as possible that a process has hung before killing it, though. Some programs become unresponsive for brief periods as they process data, for instance. You'll have to be familiar with the normal patterns of your programs to make the determination of whether or not a process is truly hung.

You might also want to kill a process if it's gone wildly out of control—for instance, if a program is generating huge output files that you don't want. Such processes can threaten others, by consuming limited resources like disk space, CPU time, and memory.

You can kill most processes with the `kill` command, as discussed in Chapter 7. For some services, it's best to kill them by using their SysV startup scripts, as described in Chapter 6. The reason is that some programs use *lock files*, which are files that indicate the program is using some resource that should not be shared. If you kill such a program with `kill`, the lock file may be left behind, which will cause the program to fail the next time you try to start it. You'll often see an error message (possibly in the system's log files) concerning the presence of a lock file. If you see such a message, consult the program's documentation to learn where the lock file is stored. You can usually delete such files manually to work around the problem.

Restarting a program involves stopping it and then starting it again. You might do this to force the program to reexamine its configuration files—but many servers include other means of doing this. For instance, many servers respond to the `SIGHUP` signal by rereading their configuration files, so you

can pass this signal with `kill`, as described in Chapter 7, to enact changes to the server's configuration without interrupting its service even briefly. Some servers' SysV startup files include a `restart` option that automatically stops a server and starts it up again. Consult a program's documentation to learn what options it supports, or examine the SysV startup script to learn its capabilities.

Working Around Recurring Problems

If you find yourself regularly stopping or restarting processes because they've hung or crashed, the program is most likely buggy. Check with the program's maintainer to see if there is an updated version of the program. Another possibility is a hardware problem. Subtle problems with CPUs, motherboards, and RAM, as well as not-so-subtle problems with some other components, sometimes manifest themselves in the form of specific programs that crash regularly. These crashes are sometimes accompanied by kernel oopses. Sometimes these or other hardware errors can cause the computer to crash entirely.

If you can't replace a buggy program with an updated version, you may need to work around it. One possibility is to replace it with another program that does a similar job. If the program is truly unique, say because it performs some obscure task, you might try fixing it yourself. Describing how to do this is well beyond the scope of this book, however; consult a text on Linux programming for more information. Short of replacing the program or fixing its bug, you may be able to work around the problem by changing how you use the software. If particular command sequences cause it to fail, don't use them. If a specific data file format doesn't work, try using another. If a process that should be running at all times crashes regularly and you simply can't seem to get it working again, try creating a cron job (Chapter 7) that restarts the process if it has crashed. You can run this script as often as is necessary to get acceptable performance. If the program is a server, you might launch it from `inetd` or `xinetd` rather than using a SysV startup script.

Real World Scenario

Working Around a Buggy Router

Networking hardware relies upon the *Address Resolution Protocol (ARP)* to translate between TCP/IP addresses and networking hardware's Media Access Control (MAC) addresses. To do this, all the computers on a local network segment maintain a cache of ARP solutions. If a device needs to contact a computer whose name isn't in the ARP cache, the device can send a query to the network to locate the correct MAC address and create such an entry. Unfortunately, this process occasionally goes wrong, particularly when certain types of hardware sit between the two computers that must communicate. Specifically, one computer (let's assume it's a router) may lose its ARP entry for another computer (let's say it's your Linux box) and be unable to generate a new ARP entry when it needs to send data to the Linux box. The result is that incoming network traffic may not reach a Linux computer unless the Linux system has sent data recently.

When I encountered this problem on a network on which I administered just one computer, the solution was to create a cron job that used `ping` to send a single packet to the router every ten minutes. This solution, although in some sense unaesthetic, was quite effective, and illustrates the utility of cron jobs in working around certain types of problems.

Package Dependencies and Conflicts

Chapter 3 discusses package installation. Although this process often proceeds smoothly, there are times when it doesn't. The usual sources of problems relate to unsatisfied dependencies or conflicts between packages. The RPM and Debian package management systems are intended to help you locate and resolve such problems, but on occasion (particularly when mixing packages from different vendors), they can actually cause problems. In either event, it pays to recognize these errors and know how to resolve them.

Although dependency and conflict problems are often discussed in terms of RPM or Debian package requirements, they also occur with tarballs. These more primitive packages lack the means to automatically detect these problems, though.

Real and Imagined Package Dependency Problems

Package dependencies and conflicts can arise for a variety of reasons, including the following:

Missing libraries or support programs One of the most common dependency problems is caused by a missing support package. For instance, all K Desktop Environment (KDE) programs rely upon Qt, a widget set upon which these programs are built. If Qt isn't installed, you won't be able to install any KDE packages using RPMs or Debian packages. Libraries—support code that can be used by many different programs as if it were part of the program itself—are particularly common sources of problems in this respect.

Incompatible libraries or support programs Even if a library or support program is installed on your system, it may be the wrong version. For instance, if a program requires Qt 2.2, the presence of Qt 1.4 won't do much good. Fortunately, Linux library naming conventions allow you to install multiple versions of a library, in case you have programs with competing requirements.

Duplicate files or features Conflicts arise when one package includes files that are already installed and that belong to another package. Occasionally broad features can conflict, as well, as in two Web server packages. Feature conflicts are usually accompanied by name conflicts. Conflicts are most common when mixing packages intended for different distributions because distributions may split files up across packages in different ways.

Mismatched names RPM and Debian package management systems give names to their packages. These names don't always match across distributions. For this reason, if one package checks for another package by name, the first package may not install on another distribution, even if the

appropriate package is installed, because that target package has a different name. This problem was a common one when Red Hat packages were installed on some non-Red Hat distributions in 1999 and 2000 because Red Hat referred to the critical glibc library in a way that some others didn't.

Some of these problems are very real and serious. Missing libraries, for instance, must be installed. (Sometimes, though, a seemingly missing library isn't quite as missing as it seems, as discussed shortly, in "Forcing the Installation.") Others, like mismatched package names, are artifacts of the packaging system. Unfortunately, it's not always easy to tell into which category a conflict fits. When using a package management system, you may be able to use the error message returned by the package system, along with your own experience with and knowledge of specific packages, to make a judgment. For instance, if RPM reports that you're missing a slew of libraries with which you're unfamiliar, you'll probably have to track down at least one package—unless you know you've installed the libraries in some other way, in which case you may want to force the installation.

When installing tarballs, you won't get any error messages during installation; you'll only see problems when you try to run the program. These messages may relay an inability to locate a library or run a file, or they may simply cause the program to crash or otherwise misbehave. Conflicts can be particularly insidious with tarballs because you won't be warned about conflicts, so installing a package can break an existing one, and you might not notice the damage for some time. You can use the `--keep-old-files` qualifier to keep `tar` from overwriting existing files, though.

Workarounds to Package Dependency Problems

When you encounter a package dependency or conflict, what can you do about it? There are several approaches to these problems. Some of these approaches work well in some situations but not others, so you should review the possibilities carefully. The options include forcing the installation, modifying your system to meet the dependency, rebuilding the problem package from source code, and finding another version of the problem package.

Forcing the Installation

One approach is to ignore the issue. Although this sounds risky, in some cases involving failed RPM or Debian dependencies, it's appropriate. For

instance, if the dependency is on a package that you installed by compiling the source code yourself, you can safely ignore the dependency. When using `rpm`, you can tell the program to ignore failed dependencies by using the `--nodeps` parameter, thus:

```
# rpm -i apackage.rpm --nodeps
```

You can force installation over some other errors, such as conflicts with existing packages, by using the `--force` parameter:

```
# rpm -i apackage.rpm --force
```

Do *not* use `--nodeps` or `--force` as a matter of course. Ignoring the dependency checks can lead you into trouble, so you should use these options only when you need to do so. In the case of conflicts, the error messages you get when you first try to install without `--force` will tell you which packages' files you'll be replacing, so be sure you back them up or are prepared to reinstall the package in case of trouble.

If you're using `dpkg`, you can use the `--ignore-depend=`*package*, `--force-depends`, and `--force-conflicts` parameters to overcome dependency and conflict problems in Debian-based systems. Because there's less deviation in package names and requirements among Debian-based systems, though, these options are less often needed on such systems.

Upgrading or Replacing the Depended-Upon Package

Officially, the proper way to overcome a package dependency problem is to install, upgrade, or replace the depended-upon package. If a program requires, say, Qt 1.44 or greater, you should upgrade an older version (such as 1.40) to 1.44. To perform such an upgrade, you'll need to track down and install the appropriate package. This usually isn't too difficult if the new package you want comes from a Linux distribution; the appropriate depended-upon package should come with the same distribution.

One problem with this approach is that packages intended for different distributions sometimes have differing requirements. If you run Distribution A and install a package that was built for Distribution B, the package will express dependencies in terms of Distribution B's files and versions. The

appropriate versions may not be available in a form intended for Distribution A, and by installing Distribution B's versions, you can sometimes cause conflicts with other Distribution A packages. Even if you install the upgraded package and it works, you could run into problems in the future when it comes time to install some other program or upgrade the distribution as a whole—the upgrade installer might not recognize Distribution B's package or might not be able to upgrade to its own newer version.

Rebuilding the Problem Package

Some dependencies result from the libraries and other support utilities installed on the computer that compiled the package, not from requirements in the underlying source code. If the software is recompiled on a system that has different packages, the dependencies will change. Therefore, rebuilding a package from source code can overcome at least some dependencies.

If you use an RPM-based system, the command to rebuild a package is straightforward: You call `rpm` with the name of the source package and use `--rebuild`, as follows:

```
# rpm --rebuild packagename-version.src.rpm
```

Of course, to do this you must have the source RPM for the package. This can usually be obtained from the same location as the binary RPM. When you execute this command, `rpm` extracts the source code and executes whatever commands are required to build a new package—or sometimes several new packages. (One source RPM can build multiple binary RPMs.) The compilation process can take anywhere from a few seconds to several hours, depending upon the size of the package and the speed of your computer. The result should be one or more new binary RPMs in `/usr/src/distname/RPMS/arch`, where *distname* is a code for your distribution (such as `RedHat` for Red Hat or `OpenLinux` for Caldera) and *arch* is your CPU architecture (such as `i386` or `i586` for *x*86 or `ppc` for PowerPC). You can move these RPMs to any convenient location and install them just like any others.

Source packages are also available for Debian systems, but aside from sites devoted to Debian and related distributions, Debian source packages are rare. The sites that do have these packages provide them in forms that typically install easily on appropriate Debian or related systems. For this reason, it's less likely that you'll rebuild a Debian package from source.

You can also recompile a package from a source tarball. Doing this requires that you read the documentation that comes with the source package. Details differ from one package to another, but you must typically run half a dozen or fewer commands to configure the compilation scripts for your system, compile the software, and install the software. You may also need to edit some configuration files manually. This process is best undertaken if you have some experience with the process—but of course, everybody who's done it had to start with one package, so you may want to try it some time simply to gain the experience.

However you do it, recompiling a program from source code requires more software than installing a precompiled program does. At the very least, you need whatever compiler is appropriate for the software (usually the GNU C Compiler [GCC]). Especially for X-based programs, you're also likely to need *header files* for the libraries the program uses. These are special files needed to compile a program to use the libraries, and they may not be installed on an ordinary system. Header file packages frequently contain the word `devel` in their names, as in `qt-devel` or `ncurses-devel`. You can install these packages just like any other. The version you install should match the version of the associated library, though; if this isn't the case, programs that use the library might not compile properly, or they might not work once they've been compiled.

Rebuilding a package from source code, even via RPM, doesn't always work. Part of the reason for this is that necessary header file packages might be missing. Similarly, compilers or other development tools might be missing or be the wrong version. Sometimes, the underlying source code relies upon features that aren't present in the version of the library you're using. When these problems occur, you'll see error messages when you are preparing or compiling the package. These error messages may be cryptic, or they may refer specifically to a version conflict or the like. In such cases, you may need to use some other method of working around the dependency problem.

Locating Another Version of the Problem Package

Frequently, the simplest way to fix a dependency problem or package conflict is to use a different version of the package you want to install. This could be a newer or older official version (4.2.3 rather than 4.4.7, say), or it might be the same official version but built for your distribution rather than for another distribution. Sites like RPM Find (`http://www.rpmfind.net`) or Debian's package listing (`http://www.debian.org/distrib/packages`) can be very useful in tracking down alternative versions of a package. Your

own distribution's Web or FTP site can also be a good place to locate packages, as well.

If the package you're trying to install requires newer libraries than you've got, an older version may work with your existing libraries.

The main problem with locating another version of the package is that sometimes you really need the version that's not installing correctly. It might have features that you need, or it might fix important bugs. On occasion, other versions might not be available, or you might be unable to locate another version of the package in your preferred package format. (You may be able to convert from another format by using the `alien` tool described in Chapter 3.)

Startup Script Problems

One particularly common problem when trying to install servers from one distribution in another is in getting SysV startup scripts working. Although all major Linux distributions use SysV startup scripts, these scripts are not always transportable across distributions. Different distributions frequently implement support routines in unique ways, so these scripts may be incompatible. The result is that the server you installed may not start up, even if the links to the startup scripts are correct, as described in Chapter 6. The "Startup Scripts" section of that chapter includes a discussion of ways around this problem. Possibilities include modifying the startup script that came with the server, building a new script based on another one from your distribution, and starting the server through a local startup script like `/etc/rc.d/rc.local` or `/etc/rc.d/boot.local`.

Startup script problems only affect servers and other programs that are started automatically when the computer boots; they don't affect typical user applications or libraries.

Using Common Troubleshooting Commands

Certain general-purpose commands and shell tools are very useful in troubleshooting certain types of problems. These commands can be used in other contexts, as well, and in fact, they aren't usually thought of as exclusively troubleshooting in nature. This section describes some of these commands, breaking them down into file-location, file-examination, and redirection tools.

File-Location Commands

You use file-location commands to locate a file on your computer. Most frequently, these commands help you locate a file by name, or sometimes by other criteria, such as modification date. These commands can search a directory tree (including root, which scans the entire system) for a file matching the specified criteria in any subdirectory.

find

The `find` utility implements a brute-force approach to finding files. This program finds files by searching through the specified directory tree, checking filenames, file creation dates, and so on to locate the files that match the specified criteria. Because of this method of operation, `find` tends to be slow, but it's very flexible and is very likely to succeed, assuming the file for which you're searching exists. The `find` syntax is as follows:

```
find [path...] [expression...]
```

You can specify one or more paths in which `find` should operate; the program will restrict its operations to these paths. The *`expression`* is a way of specifying what you want to find. The `find` man page includes information on these expressions, but some of the more common include the following:

`-name` ***`pattern`*** You can search for a filename that matches the specified *`pattern`* using this expression. If *`pattern`* is an ordinary filename, `find` matches that name exactly. You can use wildcards if you enclose *`pattern`* in quotes, and `find` will locate files that match the wildcard filename.

-perm *mode* If you need to find files that have certain permissions, you can do so by using the -perm expression. *mode* may be expressed either symbolically or in octal form. If you precede *mode* with a +, find locates files in which *any* of the specified permission bits are set. If you precede mode with a -, find locates files in which *all* the specified permission bits are set.

-size *n* You can search for a file of a given size with this expression. Normally, *n* is specified in 512-byte blocks, but you can modify this by trailing the value with a letter code, such as c for bytes or k for kilobytes.

-gid *GID* This expression searches for files whose group ID (GID) is set to *GID*.

-uid *UID* This expression searches for files owned by the user whose user ID (UID) is *UID*.

-maxdepth *levels* If you want to search a directory and, perhaps, some limited number of subdirectories, you can use the -maxdepth expression to limit the search.

There are many variant and additional options; find is a very powerful command. As an example of its use, consider the task of finding all C source code files, which normally have names that end in .c, in all users' home directories. If these home directories reside in /home, you might issue the following command:

```
# find /home -name "*.c"
```

The result will be a listing of all the files that match the search criteria.

Ordinary users may use find, but it doesn't overcome Linux's file permission features. If you lack permission to list a directory's contents, find will return that directory name and the error message, Permission denied.

locate

The locate utility works much like find if you want to find a file by name, but it differs in two important ways:

- locate is far less sophisticated in its search options. You normally use it to search only on filenames, and the program returns all files that

contain the specified string. For instance, when searching for `rpm`, `locate` will return other programs, like `gnorpm` and `rpm2cpio`.

- `locate` works from a database file that it maintains. Most distributions include a cron job that calls `locate` with options that cause it to update its database periodically, such as once a night or once a week. (You can also use the `updatedb` command to do this task at any time.) For this reason, `locate` may not find recent files, or it may return the names of files that no longer exist. If the database update utilities omit certain directories, files in them won't be returned by a `locate` query.

Because `locate` works from a database, it's typically much faster than is `find`, particularly on system-wide searches. It's likely to return many false alarms, though, especially if you want to find a file with a short name. To use it, type **`locate` *`search-string`***, where *`search-string`* is the string that appears in the filename.

Some Linux distributions use `slocate` rather than `locate`. `slocate` includes security features to prevent users from seeing the names of files in directories they should not be able to access. On most systems that use `slocate`, the `locate` command is a link to `slocate`, so `locate` implements `slocate`'s security features.

whereis

The `whereis` program searches for files in a restricted set of locations, such as standard binary file directories, library directories, and man page directories. This tool does *not* search user directories or many other locations that are easily searched by `find` or `locate`. `whereis` is a quick way to find program executables and related files like documentation or configuration files.

`whereis` returns filenames that begin with whatever you type as a search criterion, even if those files contain extensions. This feature often turns up configuration files in `/etc`, man pages, and similar files. To use the program, type the name of the program you want to locate. For instance, the following command locates `ls`:

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.bz2
```

The result shows both the `ls` executable (`/bin/ls`) and the `ls` man page. `whereis` accepts several parameters that modify its behavior in various ways. These are detailed in the program's man page.

File-Examination Commands

Locating files by name, owner, or other surface characteristics is very convenient, but sometimes you need to locate files based on their contents, or quickly examine files without loading them into a text editor. Naturally, Linux provides tools to perform these tasks.

grep

`grep` is an extremely useful command. It searches for files that contain a specified string and returns the name of the file and (if it's a text file) a line of context for that string. The basic `grep` syntax is as follows:

```
grep [options] pattern [files]
```

Like `find`, `grep` supports a large number of options. Some of the more common options include the following:

`-c` or `--count` Instead of displaying context lines, `grep` displays the number of lines that match the specified pattern.

`-f` *`file`* or `--file=`*`file`* Takes pattern input from the specified file, rather than from the command line.

`-i` or `--ignore-case` Causes a case-insensitive search, rather than the default case-sensitive search.

`-r` or `--recursive` Searches in the specified directory and all subdirectories, rather than simply the specified directory.

The *`pattern`* is a *regular expression*, which can be a complex specification that can match many different strings. Alphabetic and numeric characters are interpreted in a literal way in a regular expression, but some others have special meaning. For instance, if you enclose a series of letters or numbers in square braces (`[]`), the system matches any one of those characters. For instance, suppose you want to locate all the files in `/etc` that contain the strings `tty1` or `tty2`. You could enter the following command:

```
# grep tty[12] /etc/*
```

You can use `grep` in conjunction with commands that produce a lot of output in order to sift through that output for the material that's important to you. (Several examples throughout this book have used this technique.) For instance, suppose you want to find the process ID (PID) of a running xterm. You can use a pipe (described shortly, in "Redirection and Pipes") to send the result of a `ps` command (described in Chapter 7) through `grep`, thus:

```
# ps ax | grep xterm
```

The result is a list of all running processes called `xterm`, along with their PIDs. You can even do this in series, using `grep` to further restrict the output on some other criterion, which can be useful if the initial pass still produces too much output.

cat

The program `cat` has nothing to do with feline pets. Rather, it's short for the word "concatenate," and it's a tool for combining files, one after the other, and sending them to *standard output* (that is, your screen, xterm, or remote login session). One common use for `cat` is to forego the multifile aspect of the command and display a single file. For instance, the following command displays the contents of `/etc/fstab`:

```
$ cat /etc/fstab
```

This can be a good way to quickly view a short file. It's much less effective for large files, though, because the top of the file will scroll off the top of the display. For very long files, it may also take a long time to scroll through the entire file.

Another use of `cat` is to quickly combine two files into one. This is best achieved in conjunction with the redirection operator (>), which is described shortly. For instance, suppose you want to combine `/etc/fstab` with `/etc/fstab-addition`. You might issue the following command:

```
# cat /etc/fstab fstab-addition > fstab-plus
```

You could then examine the resulting file, `fstab-plus`. If `fstab-addition` contains a new entry you wanted to add to `/etc/fstab`, copying `fstab-plus` over the old `/etc/fstab` will accomplish the job. In fact, `cat` can even serve as a quick-and-dirty way to create a text file, thus:

```
$ cat - > text.txt
```

The - character from which cat is reading is a shorthand for *standard input*—normally your keyboard. Anything you type after this point will be entered into `text.txt`, until you press Ctrl+D. This keystroke terminates the `cat` program, at which point `text.txt` will contain your desired text. This can be a particularly useful trick if you're using an extremely spare emergency system and need to quickly create a configuration file.

more and *less*

A program that's used in many OSs to allow users to view information in a controlled way is known as `more`. Typing **`more` *`filename`*** results in a screen-by-screen display of *`filename`*'s contents. You can press the Enter key to move down one line of text, or the spacebar to move forward by one screen. This can be a convenient way to view configuration or other text files.

Although `more` is useful, the original program has many limitations. For instance, there's no way to page *backward* through a file or search for text within the file. These needs spawned a better version of `more`, which is known as `less` in a twist of humor. In addition to paging forward, `less` allows you to type in various keystrokes to do other things. Some of these are modeled after the keystrokes used in the Emacs editor, such as Ctrl+V to move forward by a screen and Esc-V to move backwards by a screen. You can also search for text by typing **`/`** followed by the search pattern. Typing **`q`** exits from `less`. You can learn more from the `less` man page.

Most Linux systems use `less` to display man pages, so you can practice the `less` commands while viewing the `less` man page.

tail

Sometimes, you want to view the last few lines of a file, but not the beginning of the file. For instance, you might want to check a log file to see if an action you've just performed has created an entry. Because programs log actions at the ends of log files, a way to quickly check the end of the file is convenient. This was the purpose for which `tail` was written. It displays the last 10 lines of a file (or if you include the `-n` *`num`* parameter, the last *`num`* lines). For instance, to view the last 20 lines of `/var/log/messages`, you could type the following command:

```
# tail -n 20 /var/log/messages
```

Redirection and Pipes

Several of the preceding examples have used *redirection* and *pipes* (aka *pipelines*). These are mechanisms that you can use to redirect the input to a process or the output from a process. Redirection passes input to or from a file, and a pipe allows you to tie two or more programs together so that one uses the output of another as input.

Normally, the standard output of a program goes to the display you used to launch it. The output redirection operator, >, changes this, sending standard output to a file that you specify. For instance, suppose you want to capture the output of `ifconfig` in a file called `iface.txt`. You could use the following command to do this:

```
$ ifconfig > iface.txt
```

This operator wipes out the current `iface.txt` file, if it exists. If you want to append information rather than overwrite it, you can use the >> operator instead of >.

You can replace standard input by using the input redirection operator, <. This is most useful when you must routinely provide the same information to a program time after time. You can create a file with that information and pass it to the program with the input redirection operator, thus:

```
$ superscript < script-input.txt
```

To have one program take another's output as input, you use a pipe, which is represented by a vertical bar (|). An earlier example illustrated this process: The output of `ps` may contain too much information to be quickly parsed, so you can pass its output through `grep` to locate just the information you want, thus:

```
# ps ax | grep xterm
```

This command searches for the string `xterm` in the `ps` output, and displays all the lines that match. The output of `ps` goes into `grep`, and `grep`'s output appears on your screen. (You could use another pipe or redirect `grep`'s output, if you prefer.)

Using Troubleshooting Resources

Chances are you're not the first person to experience any given problem. If you can tap the knowledge of somebody who's been down the same troubled path before, you may be able to traverse that path more quickly. This is the motivation behind the existence of various troubleshooting resources. Some of these are online, others come with your distribution, and a few exist in non-electronic forms. These resources include the following:

Local documentation Most programs ship with documentation in any of several different forms. Typically, installation and detailed use documentation appears in a subdirectory of `/usr/doc` or `/usr/share/doc` named after the program. Most programs also ship with Linux man and info pages, which can be accessed by typing **`man`** or **`info`** followed by the program's name or the name of a configuration file. These man and info pages are typically much briefer than a conventional manual is, so they're most useful if you want to look up the name of a parameter or some other minor detail.

HOWTOs Many Linux users have contributed Linux documentation in the form of *HOWTO documents*. These are archived at `http://www.linuxdoc.org`, among other places. Most Linux distributions ship with some or all of these, typically in `/usr/doc/HOWTO` or `/usr/share/doc/HOWTO`. (There are also mini-HOWTOs, which are like HOWTOs, but shorter.) HOWTOs are more tutorial in nature than are man or info pages. They aren't usually comprehensive, but they do provide enough information to get you started doing something. Some HOWTOs are tied to topics rather than tools; they usually aren't manuals for specific programs, but are guides to performing some task, hence the name. Some are excellent, but others are outdated or confusing. All in all, if you need to learn how to accomplish some goal, it's worth looking for a HOWTO on the topic.

Program Web pages Most Linux programs have associated Web pages, and these Web pages frequently include documentation. This resource is particularly helpful if you want to evaluate competing products or read up on one before installing it. Program information included in package files often points you to a program's Web page. For instance, type **`rpm -qpi`** ***`packagename`*****`.rpm`** to find information on *`packagename`* in an RPM system.

Usenet news groups Usenet news is a forum in which individuals post messages for all to read. Usenet posts are similar to e-mail messages in many ways, except that they're public. Most ISPs and organizations such as universities maintain Usenet news servers, and you as an individual can access these using Linux programs like `tin` (`http://www.tin.org`) or Pan (`http://pan.rebelbase.com`). The Google Groups Web site (`http://groups.google.com`) maintains a database of newsgroup postings, so you can search for help in the form of previous queries about your problem. You should probably do this before posting a new cry for help; a quick search often turns up an answer much more quickly than does a new posting.

Printed documentation Commercial programs often come with printed documentation, and there are even books available for many open source programs. Smaller programs are often mentioned in books on the topic in general. Therefore, a trip to your bookstore or library can be quite worthwhile, particularly if you need to learn a topic in depth.

User groups Computer user groups have long been sources of information. These are groups of people who meet in person periodically (often once a month) to attend group-sponsored presentations and exchange information. Attending a *Linux user group (LUG)* can provide you with contacts that can be very useful in solving problems, or in learning about Linux generally. You can find a LUG near you by browsing to `http://www.linux.org/users/index.html`. User groups also often advertise in local computer publications or post notices in local computer stores.

By taking advantage of any or all of these resources, you can learn a great deal about Linux in a short period of time or solve almost any Linux problem. Not all resources are appropriate for solving all problems, though. For instance, attending a LUG may not do you much good if the next meeting is in two weeks and you need to solve a problem *today*. (You might be able to call somebody you met at a LUG, though.)

Even if you don't make extensive use of any given information resource, you should at least take some time to familiarize yourself with the *type* of information to be found in each of these sources. Knowing what information a resource can provide can be very valuable if and when you need to locate that type of information.

Summary

Troubleshooting involves many different tasks. You must normally begin by tracking down the cause of the problem—is it software, hardware, or user-induced? What specific component is involved? What causes are likely given the symptoms? This last question is particularly tricky to answer because answering it requires knowledge of proper and improper functioning of the malfunctioning component. Because of this, in order to be able to effectively troubleshoot any problem on a Linux computer, you need extensive knowledge of every feature of Linux and the hardware on which it runs.

This chapter discusses many common or important problems in Linux, including boot errors, startup and shutdown problems and processes, and package dependency issues. Each of these has its own set of symptoms and solutions.

When you are working through a problem, there are several skills and resources that are important for you to have. One of these is the ability to boot an emergency system. These can be tiny floppy-based distributions, dedicated Linux systems on larger removable media, or even emergency installations on a hard disk. Such tools can help you get a system that won't even boot working again. Likewise, many Linux commands, such as `find`, `grep`, and `less`, are extremely useful in debugging a Linux system. Finally, knowing where to go to find help in your troubleshooting efforts is important because even the most knowledgeable Linux administrator is likely to be stumped now and then.

Exam Essentials

Summarize some ways you can localize a problem. Knowing whether a problem appears consistently or inconsistently, whether it affects the entire computer or just one or two programs, and whether it affects all users or just some can be important clues to help you track down a problem.

Describe how the LILO boot process can go awry. If the BIOS can't locate LILO or if LILO can't locate the kernel, the boot process will fail. Either of these conditions may occur for various reasons, such as installation of LILO in the wrong location, a mismatch between CHS geometries between the BIOS and LILO, or corruption of the kernel.

Summarize where you can get emergency Linux disk sets and how they're used. Emergency disk sets come with most distributions or can be obtained from third parties (including other distribution maintainers). To use them, you will need to boot from the emergency medium (often a floppy disk), and occasionally you will need to insert a second medium (like a Zip disk) to complete the boot process, after which you can use them like a regular text-mode Linux login.

Explain when you might need to restart a process. Processes sometimes become unresponsive or otherwise misbehave, thus requiring a restart. At other times you must restart a process to get it to reload a changed configuration file.

Summarize methods of working around package dependency problems. You can sometimes force a package to install by using override switches, but it's usually better to upgrade or replace the new or depended-upon packages. Sometimes rebuilding a package from source code will work around the problem, as well.

Describe where you can go to find help when you can't solve a problem. Help exists in many locations, including official documentation (documentation files in `/usr/doc` or `/usr/share/doc`, man pages, and info pages), online (program Web pages, HOWTOs, and Usenet newsgroups), and in non-electronic form (books, magazines, and people).

Commands in This Chapter

Command	Description
`setserial`	Displays or adjusts information on a computer's serial port interface
`lsof`	Displays open files on a computer
`traceroute`	Displays latencies for every router between you and a target computer

Command	Description
`find`	Locates files that match any of many search criteria, such as name, owner, and permissions
`locate`	Locates files in a system-wide database based on name
`whereis`	Locates files in common binary, documentation, and configuration directories
`grep`	Locates files that include a specified search string
`cat`	Concatenates multiple files; often used to display a complete file on the screen
`more`	Displays a file a screen at a time
`less`	An improved version of `more`
`tail`	Displays the last few lines of a file

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

Address Resolution Protocol (ARP)
header file
HOWTO document
kernel ring buffer
Linux user group (LUG)
lock file
pipeline
pipes
redirection
regular expression
software rot
standard input
standard output

Review Questions

1. When are you likely to encounter software rot?
   A. When your computer has been infected by a virus
   B. When you've installed a Trojan Horse
   C. When a program accidentally deletes a device file
   D. When disk corruption damages a program file

2. A multiuser system hosts a program called `analyze`, which is used by most of the system's users. One user comes to you to report that `analyze` is crashing frequently, but no other user has this problem. Which of the following is most likely to be true?
   A. An overheating CPU is causing `analyze` to crash.
   B. The user's configuration file or data set is causing `analyze` to crash.
   C. You must change the user's password to correct the problem.
   D. A kernel or device driver bug is causing `analyze` to crash.

3. You've installed a new SCSI scanner, but you can't get it to work. Which of the following commands is most likely to provide some clues as to what's wrong?
   A. **`dmesg | less`**
   B. **`fdisk /dev/sda`**
   C. **`scandiag`**
   D. **`lsof | less`**

4. Which of the following commands will help you identify a process that's run out of control and is consuming an inordinate amount of CPU time?
   A. **`grep`**
   B. **`fsck`**
   C. **`top`**
   D. **`cat /proc/cpuinfo`**

5. A user reports that files he's copied to floppy disk from Linux are corrupt or can't be found when the floppy disk is read on another computer. Which of the following problems is most likely to cause this symptom?

   A. The user ejected the disk without first unmounting it.

   B. The user's `.floppy` configuration file is corrupt.

   C. The CD-ROM drive is malfunctioning, interfering with floppy transfers.

   D. The computer's network drivers are buggy and causing data corruption.

6. How can you work around the problem of open files that prevent you from unmounting a filesystem?

   A. Check the contents of `/etc/mtab`.

   B. Type **`netstat`** to identify open files.

   C. Type **`lsof`** to identify open files.

   D. Type **`closefiles -a`** to close all open files.

7. Whenever you restart a Linux system, you need to manually mount the `/usr/local` directory. What configuration file might you edit to automate this process?

   A. `/etc/inittab`

   B. `/etc/mtab`

   C. `/usr/localtab`

   D. `/etc/fstab`

8. You're operating a Web server, which had previously been working well. Recently, though, you've noticed that it's responding more slowly to external requests. Which of the following might be causing this problem? (Choose all that apply.)

   A. The CPU may be going bad or overheating.

   B. The Web server's SysV startup script may be corrupt.

   C. Traffic to the Web site may have picked up.

   D. Runaway processes may be slowing down the server.

9. Which of the following should you do to avoid nasty surprises when restoring a tape backup in an emergency situation? (Choose all that apply.)

   A. Verify every backup as soon as it's made.

   B. Use only tapes that have been tested at least a hundred times.

   C. Test your emergency restore procedure before it's needed.

   D. Compress your `tar` backups with `gzip` rather than `bzip2`.

10. What type of troubleshooting information can the `traceroute` command provide?

    A. Information on unresponsive or slow routers between you and any site on the Internet

    B. The contents of your computer's routing table, including its default route

    C. Whether or not your hostname is set correctly

    D. The mapping between a computer's IP address and its hardware address

11. Your computer won't boot, and the LILO error code seems to indicate a mismatch in cylinder/head/sector (CHS) geometry between the BIOS and Linux. In a worst-case scenario, what might you need to do to correct this problem?

    A. Use Linux's `dd` to wipe out the master boot record (MBR) and recreate all the disk's partitions.

    B. Use the `w` command in Linux's `fdisk` to rewrite the disk's partition table.

    C. Delete the computer's BIOS so that it no longer interferes with LILO.

    D. Replace the hard disk with a model that's more compatible with LILO.

12. If LILO doesn't work and you want to boot Linux, which of the following techniques might you use? (Choose all that apply.)

    A. `LOADLIN`

    B. System Commander

    C. DOS's `SYS` command

    D. A raw kernel on a floppy disk

13. What media can be used for emergency Linux boots? (Choose all that apply.)

    A. Tapes

    B. Floppies

    C. CD-ROMs

    D. Zip disks

14. Why might you want to modify an emergency recovery disk set?

    A. To include custom or commercial packages that aren't on conventional emergency disk sets

    B. To remove unnecessary tools like `fdisk` and `fsck`, making space for other tools

    C. To ensure that the disk set includes the latest versions of Netscape and StarOffice

    D. To add drivers for Web cameras, scanners, or sound cards

15. What is used to signal that a program is using a resource so that other programs don't attempt to use that same resource?

    A. A lock file

    B. A pipe

    C. Redirection

    D. A signal

16. Which of the following dependency problems is *least* likely to be caused by a fundamental problem, and *most* likely to be caused by an artificial naming mismatch?

A. Two packages both include files called `/usr/bin/bigprog`.

B. A package complains that it needs `biglib-1.4`, but you've installed `biglib-1.4plus`.

C. Your system has the `biglib-1.4` package installed, but a program requires `biglib-1.9`.

D. A package complains that it needs `biglib-devel-1.4`, but you've installed `biglib-1.4`.

17. Which of the following are disadvantages of rebuilding a package to work around a dependency problem? (Choose all that apply.)

A. This approach won't work if you use a CPU architecture other than the one the developer used.

B. Rebuilding a package often requires development header files that you may not have.

C. Rebuilding a package always loses dependency information.

D. Rebuilding a package takes more time than installing a conventional binary package.

18. Which of the following file-location commands is likely to take the *most* time to find a file that might be located anywhere on the computer?

A. `find`

B. `locate`

C. `whereis`

D. They're all equal in speed.

19. Which of the following commands is an improved version of `more`?

A. `grep`

B. `tail`

C. `cat`

D. `less`

20. Which of the following statements is a fair comparison of man pages to HOWTO documents?

A. Man pages require Internet access to read; HOWTOs do not.

B. Man pages are a type of printed documentation; HOWTOs are electronic.

C. Man pages describe software from a user's point of view; HOWTOs are programmers' documents.

D. Man pages are brief reference documents; HOWTOs are more tutorial in nature.

Answers to Review Questions

1. D. Software rot is a condition in which a program file (or possibly a configuration file, library, or other support file) becomes corrupt because of a filesystem error or some other problem. This condition has nothing to do with viruses or Trojans, nor does it refer to programs behaving in an incorrect way such as deleting a device file.

2. B. Because only one user is experiencing problems, chances are good that the problem is related to this user's configuration, method of using the program, or data fed into the program. Hardware and kernel problems would almost certainly influence other users, and probably other programs. The user's password is probably not involved because the password is not normally used by programs once a user has logged in.

3. A. The `dmesg` utility displays system startup messages, which should include information on SCSI devices detected by the SCSI host adapter. (Piping `dmesg` through `less` allows you to view the output a page at a time.)

4. C. The `top` utility displays information on the processes that are running at any given time, sorted by order of CPU time used. If a process that normally consumes little CPU time shoots to the top of the list, it may have hung or be misbehaving in some way.

5. A. Linux caches all disk transfers, including those to floppy disks, so ejecting a floppy without first unmounting it can cause data corruption. There is no `.floppy` configuration file required for floppy access in Linux. CD-ROM and network hardware and drivers are unlikely to influence writing a file to a floppy disk.

6. C. The `lsof` program displays a list of all open files. (You may want to pipe its output through `grep` to make its output more manageable). Once you've found the open files on the partition you want to unmount, you can close them.

7. D. `/etc/fstab` defines the partitions that a Linux system mounts automatically when the system starts. Adding an entry for `/usr/local`, or editing an existing entry, will make this directory mount automatically.

8. C, D. Servers can become slow because of increased demand for the server's services or because local tools are consuming more of the server computer's resources. A bad or overheating CPU is likely to manifest in server crashes rather than reduced server performance. Likewise, a corrupt startup script is likely to cause the server to not start at all.

9. A, C. Testing—both in the form of testing emergency restore procedures generally and in the form of testing your backup media when you make backups—can help detect and head off problems in an emergency situation. Because tapes degrade with use, any tape that's been used (or tested) a hundred times is more likely than a new one to have errors. In the event of an error at restore, `tar` archives compressed with *either* `gzip` or `bzip2` will restore only to the point of the error. Without these forms of compression, more of an archive will be recoverable.

10. A. `traceroute` returns three round-trip latencies between your computer and each router that lies on the path between your system and a target system. This information can help you identify network bottlenecks.

11. A. Erasing the MBR and creating a new one often works around CHS geometry problems, but at a potentially serious cost: All existing partitions will be lost. You'll then have to re-create these partitions and restore all data from a backup, or reinstall Linux from scratch.

12. A, D. `LOADLIN` is a DOS utility that can boot a kernel. Linux kernels can also boot if placed on a floppy disk without a filesystem. Both these methods will require some preparation *before* LILO fails. System Commander is a boot loader that can't boot Linux without the help of LILO or GRUB. DOS's `SYS` command will rewrite critical DOS boot files, but won't do anything to help if LILO fails.

13. B, C, D. Just about any medium that's bootable or that can be used in conjunction with another bootable medium can be used for an emergency Linux installation. Larger media, such as CD-ROMs, support more sophisticated emergency recovery systems. Tapes can't be used because Linux can't store and use a normal Linux filesystem on a tape.

14. A. If you use a commercial or custom package for backups, unusual filesystems, or the like, you'll want to include support for these tools on your emergency recovery disk. You should not remove `fdisk` or `fsck`, both of which are likely to be important in some recovery situations. Netscape, StarOffice, Web cameras, scanners, and sound cards are all unnecessary to emergency system operation, and so they won't exist on most emergency systems. (Very large ones, such as CD-ROM–based systems, may include these tools, but they aren't required for emergency system operation.)

15. A. For some resources, such as RS-232 serial ports, programs create special files, known as lock files, to signal to other programs that the resource is being used. When the original program finishes, it deletes the lock file so that other programs may use the resource. If this fails to happen for some reason, you may need to manually remove the lock file.

16. B. The package name `biglib-1.4plus` is a bit odd, and it probably indicates that somebody built it as an expanded version of the `biglib-1.4` package. It's probably compatible with the standard `biglib-1.4` package, but this isn't certain. The other options are all much more likely to be serious and real problems.

17. B, D. Rebuilding a package takes time and may not work if you lack important header files or other development tools. It will often work on CPUs other than the one the program's author used, and if you use an RPM or Debian source package, the result will be an RPM or Debian package, complete with dependency information.

18. A. `find` operates by searching all files in a directory tree, and so it is likely to take a long time to search all a computer's directories. `locate` uses a precompiled database, and `whereis` searches a limited set of directories, so these commands will take less time.

19. D. `less`, like `more`, displays a text file a page at a time. `less` also includes the ability to page backwards in the text file, search its contents, and more.

20. D. Man pages are intended to give you quick information on commands, configuration files, or the like. HOWTOs are intended as introductions to packages or broad topics.

Glossary

1024-cylinder limit The *x*86 BIOS has traditionally been unable to read past the 1024th cylinder in a cylinder/head/sector (CHS) addressing scheme, which has limited the size of hard disks—first to 504MB, then to just under 8GB. On a computer with an old BIOS, the 1024-cylinder limit prevents the system from booting a kernel from higher than this limit, although Linux itself uses addressing schemes that aren't bothered by this limit. BIOSes made since the late 1990s also include ways around the limit, if the software understands those mechanisms. See also *cylinder/head/sector (CHS) addressing*.

absolute directory name A directory name that begins with a slash (/), indicating that it's to be interpreted starting from the root (/) directory.

account Stored information and a reserved directory that allows one individual to use a computer. The term is often used and thought of as if it were a distinct virtual component of a computer that a person can use, as in "Sam logged into his account." or "Miranda's account isn't working."

ACPI See *Advanced Configuration and Power Interface (ACPI)*.

Address Resolution Protocol (ARP) A protocol used to obtain a network hardware address based on an IP address.

Advanced Configuration and Power Interface (ACPI) A power management protocol. Linux includes limited ACPI support.

Advanced Power Management (APM) A power management protocol. Linux includes better APM support than ACPI support.

anonymous FTP site A server that allows access via the FTP protocol, using a username of `anonymous` and any password. (Conventionally, you give your e-mail address as the password.) Anonymous FTP servers are commonly used for distributing files on the Internet.

APM See *Advanced Power Management (APM)*.

AppleTalk A network protocol stack used by Apple with its Macintosh computers. AppleTalk is used primarily on local networks for file and printer sharing.

ARP See *Address Resolution Protocol (ARP)*.

bad block A sector on a disk that's unable to reliably store data. Bad blocks inevitably develop on magnetic media over time, but when this begins to happen, the disk tends to deteriorate rapidly.

Basic Input/Output System (BIOS) A low-level software component included on a computer's motherboard in read-only memory (ROM) form. The CPU runs BIOS code when it first starts up, and the BIOS is responsible for locating and booting an OS or OS loader.

baud rate A measure of data transmission speed, commonly used over serial lines, corresponding to the number of signal elements transmitted per second. This term is often used as a synonym for "bits per second," but many modems encode more than one bit per signal element, so the two aren't always synonymous.

Berkeley Standard Distribution (BSD) Originally a set of add-on utilities for the original AT&T Unix release. BSD later became an independent family of Unix-like OSs. BSD also refers to specific components or ways of doing things derived from these.

binary 1. The base-2 numbering system. 2. A program or file that contains data other than plain text, such as graphics or program data. 3. The version of a program that the computer runs, as opposed to the source code version of the program.

binary package A file that contains a compiled and ready-to-run Linux program, including necessary configuration files, documentation, and other support files.

BIOS See *Basic Input/Output System (BIOS)*.

bit A binary digit (0 or 1).

boot loader A program that directs the boot process. The BIOS calls the boot loader, which loads the Linux kernel or redirects the boot process to another boot loader.

boot sector The first sector of a disk or partition. The boot sector for a bootable disk or partition includes boot loader code, although this code may be absent from non-bootable disks or partitions. See also *boot loader*.

broadband 1. High-speed (greater than 200Kbps) Internet connections delivered to homes and small businesses. 2. Networking technologies that allow simultaneous transmission of data, voice, and video.

broadcast A type of network access in which one computer sends a message to many computers (typically all the computers on the sender's local network segment).

BSD See *Berkeley Standard Distribution (BSD)*.

build number A number identifying minor changes made to a binary package by its maintainer, rather than changes implemented by the program's author, which are reflected in the version number.

bus A data transfer mechanism within the computer, such as the SCSI bus or the memory bus.

byte An 8-bit number, typically represented as falling between 0 and 255.

C library (libc) Standard programming routines used by many programs written in the C programming language. The most common Linux C library is also referred to as libc or GNU libc (glibc).

cache memory A fast form of memory that's used to temporarily hold a subset of a larger but slower memory store. When they are properly implemented, caches can improve system performance. Hard disks include RAM as cache for data on disk, and computers can implement their own disk caches. Modern CPUs include a form of cache for RAM, and some motherboards include the same.

Card Services A package that helps integrate PC Card (aka PCMCIA) devices into Linux.

central processing unit (CPU) The main chip on a computer, which handles the bulk of its computational tasks.

checksum A simple file integrity check in which the values of individual bits or bytes are summed up and compared to a stored value for a reference version of the file.

child process A relative term referring to a process that another one has created. For instance, when you launch a program from a bash shell, the program process is a child process of the bash shell process.

chipset One or more chips that implement the main features of a motherboard or add-in board for a computer. The chipset is *not* the CPU, though; the chipset provides more specialized functions, such as the ability to control a hard disk or produce a video display.

CHS addressing See *cylinder/head/sector (CHS) addressing*.

CHS mode See *cylinder/head/sector (CHS) mode*.

CHS translation See *cylinder/head/sector (CHS) translation*.

CIFS See *Common Internet Filesystem (CIFS)*.

client 1. A program that initiates data transfer requests using networking protocols. 2. A computer that runs one or more client programs.

command prompt One or more characters displayed by a shell or other program to indicate that you're to type a command. Many Linux distributions use a dollar sign (`$`) as a command prompt for ordinary users, or a pound sign (`#`) as a command prompt for `root`.

Common Internet Filesystem (CIFS) Name for an updated version of the Server Message Block (SMB) file sharing protocols. CIFS is implemented in Linux via the Samba suite.

compiler A program that converts human-readable source code for a program into a binary format that the computer runs.

Complementary Metal Oxide Semiconductor (CMOS) setup utility A part of the BIOS that gives the user the ability to control key chipset features, such as enabling or disabling built-in ports.

conditional expression A construct of computer programming and scripting languages used to express a condition, such as the equality of two variables or the presence of a file on a disk. Conditional expressions allow a program or script to take one action in one case and another action in the other case.

console 1. The monitor and keyboard attached directly to the computer. 2. Any command prompt, such as an xterm window.

Coordinated Universal Time (UTC) See *Greenwich Mean Time*.

copyleft Slang term referring to the GNU GPL or certain other open source licenses.

core dump A record of a crashed process's memory, stored in a disk file at the time of the process's crash. Core dumps can sometimes be used to debug the cause of the crash.

CPU See *central processing unit (CPU)*.

cracker An individual who breaks into computers. Crackers may do this out of curiosity, malice, for profit, or for other reasons.

crash An event in which a program terminates abruptly and abnormally. Crashes are typically the result of programming errors.

creating a filesystem Writing low-level filesystem (meaning 1) data structures to a disk. This is sometimes also called high-level formatting. See also *filesystem*.

crippleware Software that's distributed without some important feature, or that works for only a limited period of time, as a demonstration to produce sales of the full product.

cron job A program or script that's run at a regular interval by the cron daemon. See also *system cron job* and *user cron job*.

cylinder/head/sector (CHS) addressing A method of hard disk addressing in which a triplet of numbers (a cylinder, a head, and a sector) are used to identify a specific sector. CHS addressing contrasts with linear block addressing (LBA).

cylinder/head/sector (CHS) mode See *cylinder/head/sector (CHS) addressing*.

cylinder/head/sector (CHS) translation Modifying one CHS addressing scheme into another. CHS translation is commonly used by BIOSes from the mid-to-late 1990s to allow the systems to use hard disks between 504MB and 8GB in capacity.

daemon A program that runs constantly, providing background services. Linux servers are typically implemented as daemons, although there are a few non-server daemons.

Data Display Channel (DDC) A protocol that allows a computer to query a monitor for its maximum horizontal and vertical refresh rates and other vital statistics.

DDC See *Data Display Channel (DDC)*.

Debian package A package file format that originated with the Debian distribution, but is now used on several other distributions. Debian packages feature excellent dependency tracking and easy installation and removal procedures.

default route The route that network packets take if a more specific route doesn't direct them in some other way. The default route typically involves a gateway or router system that can further redirect the packets.

demoware A demonstration version of a commercial program. See also *crippleware*.

dependency A requirement of one software package that another one be installed. For instance, most Linux programs include a dependency upon the C library.

desktop computer A computer that sits on a desk and that's used by an individual for productivity tasks. A desktop computer is similar to a workstation, but some people use "desktop" to refer to somewhat lower-powered computers or those without network connections. See also *workstation*.

desktop environment A set of programs that provide a friendly graphical environment for a Linux user.

development kernel A kernel with an odd middle number, such as 2.3.47. These kernels incorporate experimental features and are not as stable as are release kernels. See also *release kernel*.

device file A special file, typically residing in the `/dev` directory, that provides programs with access to hardware. Device files exist for most types of hardware, such as RS-232 serial ports, floppy disks, sound cards, and so on. Many devices sport multiple device files.

DHCP See *Dynamic Host Configuration Protocol (DHCP)*.

DIMM See *dual inline memory module (DIMM)*.

direct memory access (DMA) A means of transferring data between devices (like sound cards or SCSI host adapters) and memory without directly involving the CPU.

distribution A complete collection of a Linux kernel and programs necessary to do work with Linux. Dozens of different Linux distributions exist, each with its own unique characteristics; but they all work in a similar way and can run the same programs, assuming similar vintages of critical support libraries like libc.

DMA See *direct memory access (DMA)*.

DNS See *Domain Name System (DNS)*.

domain name A name associated with an organization or set of computers. Individual computers are assigned names within a domain, and domains can be partitioned into subdomains.

Domain Name System (DNS) A distributed set of computers that run servers to convert between computer names (such as `ns.example.com`) and IP addresses (such as 192.168.45.204). DNS servers are organized hierarchically and refer requests to systems responsible for successively more specific domains.

dot file A Linux or Unix file whose name begins with a dot (`.`). Most Linux shells and programs hide such files from the user, so user configuration files usually come in this form to be unobtrusive in directory listings.

drag bar The window control, typically displayed at the top of the window, that allows users to drag the window around the screen. The drag bar is sometimes also called the title bar because the window's title appears in this area.

DRAM See *dynamic RAM (DRAM)*.

dual inline memory module (DIMM) One of several types of small circuit boards on which memory chips are distributed, for ease of installation in computers. DIMMs are used on some Pentium-level and later computers.

Dynamic Host Configuration Protocol (DHCP) A protocol used on local networks for dissemination of network configuration information. A single DHCP server can maintain information for many DHCP clients, reducing overall configuration effort.

dynamic RAM (DRAM) One of several types of RAM. Plain DRAM is now largely obsolete in desktop computers.

dynamically linked Programs that call upon information in libraries by loading a separate library file when the program runs. This contrasts with statically linked programs. See also *library*.

effective user ID The owner associated with a running process. This may or may not be the same as the user ID of the individual who ran the program.

EIDE See *Enhanced Integrated Device Electronics (EIDE)*.

electrostatic discharge (ESD) A transfer of an electrical charge from one body to another. ESD frequently occurs when a person who's shuffled around on a carpet in dry weather touches a metallic object such as a doorknob or computer component. It can be very damaging to electronic devices such as computers.

Enhanced Integrated Device Electronics (EIDE) A type of interface for hard disks, CD-ROM drives, tape drives, and other mass storage devices.

envelope In networking, the portion of a data packet that directs the transmission and routing of the packet. The envelope includes information such as the source and destination addresses and other housekeeping information.

environment variable A setting that's available from any program running in a session. Environment variables can define features such as the terminal type being used, the path to search for executable programs, and the location of an X server for GUI programs.

ESD See *electrostatic discharge (ESD)*.

Ethernet The most common form of local networking in 2001.

ext2 See *Second Extended Filesystem (ext2 or ext2fs)*.

ext2fs See *Second Extended Filesystem (ext2 or ext2fs)*.

extended INT13 BIOS routines added in the late 1990s to allow *x*86 computers to boot from hard disks larger than 8GB.

extended partition A type of disk partition used on *x*86 systems. Extended partitions are placeholders for one or more logical partitions.

external transfer rate The data transfer rate between one device and another. The external transfer rate is frequently applied to disks and similar devices in reference to the speed of the EIDE or SCSI interface, as opposed to the speed of the drive mechanism itself. In this context, the external transfer rate is almost always faster than the internal transfer rate.

failed dependency A state in which a package's dependencies are not met when attempting to install it, or in which removing a package would cause other installed packages to have unmet dependencies.

FDDI See *Fiber Distributed Data Interface (FDDI)*.

FHS See *Filesystem Hierarchy Standard (FHS)*.

Fiber Distributed Data Interface (FDDI) A type of network hardware that supports up to 100Mbps speeds over fiber-optic cables.

Fibre Channel A type of network hardware that supports up to 1062Mbps speeds over fiber-optic cables.

file access permissions See *file permissions*.

file manager A GUI program that allows users to manipulate files using mouse point-and-click operations.

filename completion A feature of some shells that allows them to complete a command or filename when you press the Tab key.

file owner The account with which a file is most strongly associated. The owner often has permission to do more with a file than other users can do.

file permissions Linux's file access control mechanism. Every file has an owner, a group, and permissions that define how the owner, group members, and all other users (the "world") may access the file. Permissions include read, write, and execute for the owner, group, and world.

file sharing protocol A network protocol that allows one computer to access files stored on a second computer as if the second computer's files were local to the first computer. Examples include SMB/CIFS (used on Windows-dominated networks), NFS (used on Unix-dominated networks), and AppleShare (used on Macintosh-dominated networks).

filesystem 1. The low-level data structures recorded on a disk in order to direct the placement of file data. The filesystem determines characteristics like the maximum partition size, the file naming rules, and what extra data (time stamps, ownership, and so on) may be associated with a file. 2. The overall layout of files and directories on a computer. For instance, a Linux filesystem usually includes a root directory (`/`), several directories falling off of this (`/usr`, `/var`, `/boot`, etc.), subdirectories of these, and so on.

Filesystem Hierarchy Standard (FHS) A standard that defines the names and contents of critical directories in a Linux filesystem (meaning 2).

Filesystem Standard (FSSTND) An early attempt to define the names and contents of critical directories in a Linux filesystem (meaning 2). FSSTND has been supplanted by the FHS.

File Transfer Protocol (FTP) A simple protocol for transferring files over TCP/IP networks.

file type code A code that defines the type of a file—a normal file, a directory, a symbolic link, and so on.

FireWire A name for IEEE-1394 that's favored by Apple.

focus In the context of window managers, focus refers to the window that receives input from the keyboard and mouse. Focus may be changed in various ways, depending upon the window manager and its settings. See also *window manager*.

font server A program that provides font bitmaps to client programs on the same or (sometimes) other computers. The font server may work directly from font bitmaps, or it may generate the bitmaps from outline fonts such as PostScript Type 1 or TrueType fonts.

formatting Writing data structures necessary for a disk to hold data. Formatting may be either high-level or low-level. High-level formatting is synonymous with creating a filesystem. Low-level formatting is seldom necessary on hard disks, but it is more common on floppies. It entails re-creating the low-level data structures that define the locations of individual sectors.

fragmented Term referring to files whose contents are split across several parts of a disk, rather than placed contiguously. File fragmentation tends to degrade disk performance because it increases head movements when reading files.

frame In networking, a data packet associated with network hardware (such as Ethernet), as opposed to the software (such as TCP/IP).

frame buffer A low-level but standardized interface between software and video hardware. X uses a frame buffer interface on many non-*x*86 computers.

free software Term favored by the Free Software Foundation to refer to certain types of open source software. Used in this way, the term should not be confused with freeware.

freeware Any software that may be distributed and used freely, whether or not it's open source. Not to be confused with free software, at least as the term is most commonly used within the open source community. See also *free software*.

FSSTND See *Filesystem Standard (FSSTND)*.

FTP See *File Transfer Protocol (FTP)*.

full duplex A mode of communication in which data can be transferred in two directions at the same time.

gateway A computer that functions as a router between two networks.

General Public License (GPL) A software license developed by the Free Software Foundation. The GPL is one of several open source licenses. It allows anybody to modify a program and distribute modifications, so long as the changed version is also distributed under the terms of the GPL.

`getty` A program that handles the login process from a text-mode console or serial port.

GID See *group ID (GID)*.

gigabit Ethernet A variety of Ethernet that can transfer 1,000 megabits (1 gigabit) per second.

glibc A specific type of C library used on Linux systems since the late 1990s.

GMT See *Greenwich Mean Time (GMT)*.

GNOME See *GNU Network Object Model Environment (GNOME)*.

GNU Recursive acronym for GNU's Not Unix. GNU is a Free Software Foundation (FSF) project to build an entirely open source OS that works like Unix. The term is also used by some non-FSF projects.

GNU/Linux Generic term for a complete Linux OS to distinguish the complete OS from the kernel alone. This term is favored by Debian; most other distributions use "Linux" alone.

GNU Network Object Model Environment (GNOME) A common desktop environment for Linux.

GPL See *General Public License (GPL)*.

graphical user interface (GUI) A method of human/computer interaction characterized by a graphical display, a mouse to move a pointer around the screen, and the ability to perform actions by pointing at objects on the screen and clicking a mouse button.

Greenwich Mean Time (GMT) The time in Greenwich, England, unadjusted for daylight savings. Linux systems use this time internally and adjust to local time by knowing the system's time zone.

group A collection of users. Files are owned by a user and a group, and group members may be given access to files independent of the owner and all other users. This feature may be used to enhance collaborative abilities by giving members of a group read/write access to particular files, while still excluding those who aren't members of the group.

group administrator A person with administrative authority over a group. A group administrator can add or delete members from a group and perform similar administrative tasks.

group ID (GID) A number associated with a particular group. Similar to a user ID (UID).

group owner The group with which a file is most strongly associated, after the file owner.

GUI See *graphical user interface (GUI)*.

hacker 1. An individual who is skilled at using or programming computers and who enjoys using these skills in constructive ways. Many Linux programmers consider themselves hackers in this sense of the term. 2. A cracker (see also *cracker*). This use of the term is more prevalent in the mass media, but it is frowned upon in the Linux community.

half-duplex A type of data transmission in which data can be sent in only one direction at a time.

hard link A directory entry for a file that has another directory entry. All hard links are equally valid ways of accessing a file, and all must be deleted before a file is deleted. See also *soft link*.

hardware address A code that uniquely identifies a single network interface. This address is built into the device itself rather than assigned in Linux.

hash An encryption method in which a file or string is encoded in a manner that cannot be reversed. Hashes are commonly used for password storage and as a more secure variant on checksums, among other things. See also *checksum*.

header files Files that contain interface definitions for software routines contained in a library. Program source code that uses a library must refer to the associated header files.

High-Performance Parallel Interface (HIPPI) A type of network hardware that supports speeds of up to 1600Mbps over fiber-optic cabling.

HIPPI See *High-Performance Parallel Interface (HIPPI)*.

home directory A directory associated with an account, in which the user's files reside.

hostname A computer's human-readable name, such as `persephone.example.com`.

hot swapping Adding or removing hardware while the computer is turned on.

HOWTO documents Linux documentation that describes how to accomplish some task or use a particular program. HOWTOs are usually tutorial in nature. They're archived at `http://www.linuxdoc.org`, and all major distributions ship with them as well.

HTTP See *Hypertext Transfer Protocol (HTTP)*.

hub A type of network hardware that serves as a central exchange point in a network. Each computer has a cable that links to the hub, so all data pass through the hub. Hubs echo all data they receive to all the other computers to which they connect. See also *switch*.

hung Term used to describe a program that's stopped responding to user input, network requests, or other types of input to which it should respond. Hung processes sometimes consume a great deal of CPU time.

Hypertext Transfer Protocol (HTTP) A protocol used for transferring Web pages from a Web server to a Web browser.

IEEE-1394 An external bus technology that's used to connect high-speed external devices such as hard disks, scanners, and video equipment. IEEE-1394 is rare in 2001, but it is likely to become more important in the future. Linux 2.4.*x* includes limited IEEE-1394 support.

IMAP See *Internet Message Access Protocol (IMAP)*.

incremental backup A type of backup in which only files that have changed are backed up. This is used to reduce the time required to back up a computer, at the cost of potentially greater restoration complexity.

Industry Standard Architecture (ISA) The expansion bus used on the original IBM PC. ISA is still available on some computers in 2001. ISA is inferior to PCI in most respects, but it has a huge installed base.

inode A filesystem (meaning 1) data structure that contains critical information on the file, such as its size and location on the disk.

input/output (I/O) A term that describes the acceptance of data from an external source or the sending of data to an external source. In some cases, the "external source" may be internal to the computer, as in I/O between a hard disk and the CPU or memory. In other cases, I/O is more clearly external, as in network I/O.

Integrated Services Digital Network (ISDN) A type of digital telephone service that also supports data transfer at speeds of up to 128Kbps. ISDN never took off in North America, but is moderately popular in Europe.

internal transfer rate The rate of data transfer within a device. This is typically applied to hard disks and similar devices to describe how quickly they can read or write data from their physical media.

internet Any collection of networks linked together by routers. See also *Internet*.

Internet The largest network on Earth, which connects computers from around the globe. When used in this way, the word is always capitalized. See also *internet*.

Internet Message Access Protocol (IMAP) A protocol for exchanging mail messages. The recipient initiates an IMAP session. IMAP differs from POP in that IMAP allows the recipient to leave messages in organized folders on the server; POP requires that the recipient download the messages to organize them.

Internet Packet Exchange (IPX) A protocol that underlies much of Novell networking. Despite the name, this protocol is unrelated to the Internet.

interrupt request (IRQ) A method by which peripherals (SCSI host adapters, sound cards, etc.) signal that they require attention from the CPU. An IRQ also refers to a specific interrupt signal line. The *x*86 architecture supports 16 IRQs, numbered 0–15, but IRQs 2 and 9 are linked, so in practice, there are only 15 IRQs, and many of these are used by basic hardware like floppy disks.

I/O See *input/output (I/O)*.

IP address A computer's numeric TCP/IP address, such as 192.168.45.203.

IPv6 The "next-generation" Internet Protocol. This upgrade to TCP/IP allows for a theoretical maximum of approximately 3.4×10^{38} addresses, as opposed to the 4 billion addresses possible with the IPv4 that's in common use in 2001.

IPX See *Internet Package Exchange (IPX)*.

IRQ See *interrupt request (IRQ)*.

ISA See *Industry Standard Architecture (ISA)*.

ISDN See *Integrated Services Digital Network (ISDN)*.

journaling filesystem A type of filesystem that maintains a record of its operations. Such filesystems can typically recover quickly after a power failure or system crash. See also *filesystem*.

jumper Metal pins on a circuit board that may be shorted via a plastic and metal cap (which is also called a jumper). When a jumper is so set, it alters the behavior of the circuit board. Jumpers were especially common on old ISA cards, and they still exist on some motherboards and many disk devices.

KDE See *K Desktop Environment (KDE)*.

K Desktop Environment (KDE) A common desktop environment on Linux.

kernel The core program of any Linux system. The kernel provides interfaces between the software and the hardware and controls the operation of all other programs. Technically, the kernel is the *only* component that is Linux; everything else, such as shells, X, and libraries, is available on other Unix-like systems.

kernel module A driver or other kernel-level program that may be loaded or unloaded as required.

kernel module autoloader A utility that loads and unloads kernel modules as required by the kernel, obviating the need to manually load and unload kernel modules.

kernel oops A kernel error and its aftermath (such as a program or system crash). These errors are most commonly caused by a bug in the kernel or a hardware problem.

kernel ring buffer A record of recent messages generated by the Linux kernel. Immediately after a Linux system boots, this buffer contains the bootup messages generated by drivers and major kernel subsystems. This buffer may be viewed with the `dmesg` command.

laptop computer A small portable computer, typically somewhat smaller than a briefcase. Laptops use special miniaturized components. Special software requirements of laptops include power management (APM or ACPI) and Card Services. A laptop computer is also known as a notebook computer.

LBA See *linear block addressing (LBA)*.

LCD See *liquid crystal display (LCD)*.

libc See *C library (libc)*.

library A collection of code that's potentially useful to many programs. This code is stored in special files to save disk space and RAM when running programs that use the library.

LILO See *Linux Loader (LILO)*.

linear block addressing (LBA) A method of accessing data on a disk that uses a single sector number to retrieve data from that sector. LBA contrasts with cylinder/head/sector (CHS) addressing. Some sources refer to LBA as *logical* block addressing.

Linux 1. The open source kernel designed by Linus Torvalds as the core of a Unix-like operating system (OS). 2. A complete OS built around Linus Torvald's kernel. See also GNU/Linux.

Linux Loader (LILO) The most popular Linux boot loader. Can boot a Linux kernel or redirect the boot process to another boot loader in a non-Linux partition, thus booting other OSs. See also *boot loader*.

Linux user group (LUG) A group of Linux users who meet on a regular basis (usually monthly) to exchange information, attend presentations, and help each other.

liquid crystal display (LCD) A type of flat-panel display that's common on laptops and is becoming more common on desktop systems. LCDs are lightweight and consume little electricity, but they're more expensive to produce than are conventional monitors.

load average A measure of the demands for CPU time by running programs. A load average of 0 means no demand for CPU time; 1 represents a single program placing constant demand on the CPU; and values higher than 1 represent multiple programs competing for CPU time. The `top` and `uptime` commands both provide load average information.

LocalTalk A type of network hardware common on older Macintosh networks.

lock file A file that's created by an application to indicate that some resource (such as an RS-232 serial port) is in use, in order to prevent conflicts over accessing that resource.

log file A text file maintained by the system as a whole or an individual server, in which important system events are recorded. Log files typically include information on user logins, server access attempts, and automatic routine maintenance.

log rotation A routine maintenance process in which the computer suspends recording data in log files, renames them, and opens new log files. It then deletes old log files. This process keeps log files available for a time, but ultimately it deletes them, preventing them from growing to consume all available disk space.

logical block addressing (LBA) See *linear block addressing (LBA)*.

logical partition A type of *x*86 hard disk partition that has no entry in the primary partition table. Instead, logical partitions are carried within an extended partition.

loop A programming or scripting construct allowing multiple executions of a segment of code. Typically terminated through the use of a conditional expression.

LUG See Linux user group (LUG).

MAC address See *Media Access Control (MAC) address*.

machine name The portion of a hostname that identifies a computer on a network, as opposed to the network as a whole (for instance, `gingko` is the machine name portion of `gingkgo.example.com`). The machine name is sometimes used in reference to the entire hostname.

main memory The main type of RAM in a computer, as opposed to cache memory.

major version number The first number in a program's version number. For instance, if a program's version number is 1.2.3, the major version number is 1.

master One of two EIDE/ATAPI devices on a single EIDE chain. The master device gets a lower Linux device letter than the slave device does.

Master Boot Record (MBR) The first sector of a hard disk. The MBR contains code that the BIOS runs during the boot process, as well as the primary partition table.

MBR See *Master Boot Record (MBR)*.

Media Access Control (MAC) address Low-level address associated with a piece of network hardware. The MAC address is usually stored on the hardware itself, and it is used for local network addressing only. Addressing between networks (such as on the Internet) uses higher-level addresses, such as an IP address.

mode The permissions of a file. In conjunction with the file's owner and group, the mode determines who may access a file and in what ways.

mode lines Definition of the timings required by particular video resolutions running at particular refresh rates.

modem This word is short for "modulator/demodulator." It's a device for transferring digital data over an analog transmission medium. Traditionally, the analog transmission medium has been the normal telephone network, but the word "modem" is increasingly being applied to devices used for broadband Internet access, as well.

module A kernel driver or other kernel component that's stored in a separate file. Linux can load modules on demand or on command, saving RAM when modules aren't in use and reducing the size of the kernel.

motherboard The main circuit board in a computer. The CPU, RAM, and add-on cards typically plug directly into the motherboard, although some designs place some of these components on extender cards. The motherboard is also sometimes referred to as the mainboard or the system board.

mount point A directory in a filesystem (meaning 2) at which a new filesystem (meaning 1) is attached. Mount points are typically empty directories before their host filesystems are mounted.

mounted A state in which a filesystem (meaning 1) is attached to its associated mount point.

NetBEUI A network stack similar to AppleTalk or TCP/IP in broad outline, but used primarily only on local networks.

NetBIOS Networking protocols that are often used in conjunction with NetBEUI or TCP/IP. NetBIOS underlies the SMB/CIFS file sharing protocols used by Microsoft Windows and implemented in Linux by Samba.

netmask See *network mask*.

Network Filesystem (NFS) A file sharing protocol used among Linux and Unix computers.

Network Information Service (NIS) A network protocol that allows computers to share simple database files. Commonly used to provide centralized login authentication and as a substitute for DNS on small networks.

network mask A bit pattern that identifies the portion of an IP address that's an entire network and the part that identifies a computer on that network. The pattern may be expressed as four decimal bytes separated by dots (as in 255.255.255.0) or as the number of network bits following an IP address and a slash (as in 192.168.45.203/24). The network mask is also referred to as the netmask or subnet mask.

NFS See *Network Filesystem (NFS)*.

NIS See *Network Information Service (NIS)*.

non-rewinding tape device A tape device file that does *not* cause the tape to automatically rewind when the job is done. The non-rewinding nature of the device is indicated by the presence of a leading `n` in the device filename, such as `/dev/nst0` or `/dev/nht0`. This file is used for handling multiple backups on a single tape. See also *rewinding tape device*.

notebook computer See *laptop computer*.

open mail relay An SMTP mail server that's configured to relay mail from anywhere to anywhere. Open mail relays are frequently abused by spammers to obfuscate their messages' true origins.

open source A type of software that's freely redistributable, is available with source code, and may be modified by anybody wanting to do so. (There is a formal 9-point definition, which is summarized in Chapter 1, "Planning the Implementation.")

Open System Interconnection (OSI) model A means of describing network stacks, such as TCP/IP, NetBEUI, or AppleTalk. In the OSI model, such stacks are broken down into several layers, each of which communicates directly with the layers above and below it.

OSI model See *Open Systems Interconnection (OSI) model*.

packet A limited amount of data packaged together with an envelope and sent over a network. See also *envelope*.

packet filter A type of firewall that operates on individual network data packets, passing or rejecting packets based on information such as the source and destination addresses and ports.

pager In X, this is a utility, window manager feature, or desktop environment feature that provides several virtual desktops. You can run separate programs in each virtual desktop and switch between them, thus minimizing desktop clutter.

parameter An option passed to a program on a command line, or occasionally as part of a configuration file.

parent process A relative term referring to the process that started another. For instance, if you launch a program from a bash shell, the bash shell process is the new program's parent process.

partition A contiguous part of a hard disk that's set aside to hold a single filesystem (meaning 1). Also used as a verb to describe the process of creating partitions on a hard disk.

partition table The disk data structure that describes the layout of partitions on a hard disk.

path A colon-delimited list of directories in which program files may be found. (Similar lists define the locations of directories, fonts, and other file types.)

payload The portion of a network data packet that contains the actual data to be transmitted, as opposed to the envelope.

PC Card A type of expansion card that's common on laptop computers. This interface is commonly used for Ethernet cards, modems, and storage devices. Also known as PCMCIA.

PCI See *Peripheral Component Interconnect (PCI)*.

PCL See *Printer Control Language (PCL)*.

PCMCIA See *Personal Computer Memory Card International Association (PCMCIA)*.

peripheral A device that connects to and is controlled by a computer. Many peripherals, such as Web cams and keyboards, are external to the computer's main box. Some definitions include devices that reside within the computer's main box, such as hard disks and CD-ROM drives.

Peripheral Component Interconnect (PCI) An expansion bus capable of much higher speeds than the older ISA bus. *x*86 computers sold in 2001 usually include several PCI slots.

permission bit A single bit used to define whether or not a given user or class of users has a particular type of access to a file. For instance, the owner's execute permission bit determines whether the owner can run a file as a program. The permission bits together comprise the file's mode.

Personal Computer Memory Card International Association (PCMCIA) 1. An earlier name for PC Card (but one that's still used by many Linux utilities and documentation). 2. The trade group that developed the PC Card standard.

PIO See *Programmed Input/Output (PIO)*.

pipe A method of executing two programs so that one program's output serves as the second program's input. Piped programs are separated in a Linux shell by a vertical bar (`|`).

pipeline See *pipe*.

plug-and-play (PnP) A term applied to hardware (especially ISA cards, and sometimes PCI cards) to denote that no hardware configuration is required, especially via jumpers. PnP devices often require special configuration via Linux files, however.

Point-to-Point Protocol (PPP) A method of initiating a TCP/IP connection between two computers over an RS-232 serial line or modem.

port number A number that identifies the program from which a data packet comes or to which it's addressed. When a program initiates a network connection, it associates itself with one or more ports, allowing other systems to uniquely address data.

Post Office Protocol (POP) A mail server protocol in which the recipient initiates transfer of messages. Differs from IMAP in that POP doesn't provide any means for the recipient to organize and store messages on the server.

PostScript A programming language used on many high-end printers. PostScript is optimized for displaying text and graphics on the printed page. The Linux program Ghostscript converts from PostScript to bitmapped formats understood by many low-end and mid-range printers.

power-on self-test (POST) A series of hardware checks performed by an *x*86 computer when it boots, to guarantee minimal functionality.

PnP See *plug-and-play (PnP)*.

POST See *power-on self-test (POST)*.

PPP See *Point-to-Point Protocol (PPP)*.

primary boot loader The first boot loader run by the BIOS.

primary partition A type of *x*86 partition that's defined in a data structure contained in the hard disk's partition table in the MBR. An *x*86 computer can host only four primary partitions.

print queue A storage place for files waiting to be printed.

Printer Control Language (PCL) A language developed by Hewlett-Packard for controlling printers. (Many of Hewlett-Packard's competitors now use PCL.) PCL is most commonly found on mid-range laser printers, but some inkjet printers also support the language. There are several PCL variants, the most common ranging from PCL 3 to PCL 6.

printer driver A software component that converts printable data generated by an application into a format that's suitable for a specific model of printer. In Linux, printer drivers usually reside in Ghostscript, but some applications include a selection of printer drivers to print directly to various printers.

process A piece of code that's maintained and run by the Linux kernel separately from other pieces of code. Most processes correspond to programs that are running. One program can be run multiple times, resulting in several processes.

Programmed Input/Output (PIO) A method of data transfer between memory and expansion cards in which the CPU actively performs the transfer. PIO tends to consume much more CPU time than DMA does.

proprietary A protocol, file format, hardware design, program, or other technology that uses features unique to the technology in question. Proprietary technologies are often difficult to handle in an open source environment because they're often poorly documented or because developers must sign nondisclosure agreements before they can obtain the documentation.

protocol stack A collection of drivers, kernel procedures, and other software that implements a standard means of communicating across a network. Two computers must support compatible protocol stacks to communicate. The most popular protocol stack today is TCP/IP.

proxy server A server (typically on a local network) that stands in for remote servers. Proxy servers can buffer transfers, provide security, and provide access controls.

public domain Intellectual property (software, music, books, etc.) that's either outlived its copyright or for which the author has removed the copyright. (No software is yet old enough to have outlived its copyright.) Public domain software may be modified and redistributed by anybody who wants to do so.

pull mail protocol A mail protocol in which the recipient initiates the transfer. Examples include POP and IMAP.

push mail protocol A mail protocol in which the sender initiates the transfer. SMTP is the most common push mail protocol.

RAMbus Dynamic RAM (RDRAM) A type of RAM used in RIMMs.

random access A method of access to a storage device (RAM, hard disk, etc.) in which information may be stored or retrieved in an arbitrary order with little or no speed penalty. See also *sequential access*.

RDRAM See *RAMbus Dynamic RAM (RDRAM)*.

RDRAM Inline Memory Module (RIMM) A small circuit board that holds memory chips configured as RDRAM. Used in some Pentium II and later computers.

Red Hat Package Manager (RPM) A package file format designed by Red Hat but now used on many other distributions, as well. RPM features excellent dependency tracking and easy installation and removal procedures.

redirection A procedure in which a program's standard output is sent to a file rather than to the screen, or in which the program's standard input is obtained from a file rather than from the keyboard. See also *standard input* and *standard output*.

regular expression A method of matching textual information that may vary in important ways but that contains commonalities. The regular expression captures the commonalities and uses various types of wildcards to match variable information.

relative directory name A directory name that's specified relative to the current directory. Relative directory names often include the parent specification (`..`), which indicates the current directory's parent.

release kernel A kernel with an even second number, such as 2.2.17 or 2.4.3. Release kernels should have few bugs, but they sometimes lack drivers for the latest hardware. See also *development kernel*.

release number See *build number*.

remote login server A type of server that allows individuals at distant locations to use a computer. Examples include Telnet, SSH, and XDM.

Request for Comments (RFC) An Internet standards document. RFCs define how protocols like Telnet and SMTP operate, thus allowing tools developed by different companies or individuals to interoperate.

rewinding tape device A tape device file that causes the tape to automatically rewind when the access is complete. The rewinding nature of the device is indicated by the lack of a leading `n` in the device filename, such as `/dev/st0` or `/dev/ht0`. This file is often used for reading or writing the first backup on a tape. See also *non-rewinding tape device*.

RFC See *Request for Comment (RFC)*.

ribbon cable A type of cable in which insulated wires are laid side by side, typically bound together by plastic. The result is a wide but thin multi-conductor cable that resembles a ribbon.

RIMM See *RDRAM Inline Memory Module (RIMM)*.

root directory The directory that forms the base of a Linux filesystem (meaning 2). All other directories are accessible from the root directory, either directly or via intermediate directories.

root filesystem The filesystem (meaning 1) on a Linux system that corresponds to the root directory, and often several directories based on it.

root partition The partition associated with the root filesystem.

router A computer that transfers data between networks. See also *gateway*.

RPM See *Red Hat Package Manager (RPM)*.

runlevel A number associated with a particular set of services that are being run. Changing runlevels changes services or can shut down or restart the computer.

Samba Web Administration Tool (SWAT) A server that allows administrators to configure Samba servers from another computer by using an ordinary Web browser.

script kiddies Individuals with little knowledge or skill, who break into computers using scripts created by others. Such break-ins often leave obvious traces, and script kiddies frequently cause collateral damage that produces system instability.

scripting language Interpreted computer programming language designed for writing small utilities to automate simple but repetitive tasks. Examples include Perl, Python, Tcl, and shell scripting languages like those used by bash and `tcsh`.

SCSI See *Small Computer System Interface (SCSI)*.

Second Extended Filesystem (ext2 or ext2fs) The most common filesystem (meaning 1) in Linux from the mid-1990s through 2001.

secondary boot loader A boot loader that's launched by another boot loader.

Secure Shell (SSH) A remote login protocol and program that uses encryption to ensure that intercepted data packets cannot be used by an interloper. Generally regarded as the successor to Telnet on Linux systems.

Sequenced Packet Exchange (SPX) Part of the Novell networking stack, along with IPX.

sequential access A method of accessing a storage medium that requires reading or writing data in a specific order. The most common example is a tape; to read data at the end of a tape, you must wind past the interceding data. See also *random access*.

server 1. A program that responds to data transfer requests using networking protocols. 2. A computer that runs one or more server programs.

Server Message Block (SMB) A file sharing protocol common on Windows-dominated networks. SMB is implemented in Linux via the Samba suite. Also known as the Common Internet Filesystem (CIFS).

server program See *server*, meaning 1.

set group ID (SGID) A special type of file permission used on program files to make the program run with the permissions of its group. (Normally, the user's group permissions are used.)

set user ID (SUID) A special type of file permission used on program files to make the program run with the permissions of its owner, rather than those of the user who runs the program.

SGID See *set group ID (SGID)*.

shadow password A method of storing encrypted passwords separately from most other account information. This allows the passwords to reside in a file with tighter security options than the rest of the account information, which improves security when compared to storing all the account information in one file with looser permissions.

shareable files Files that might reasonably be shared with another computer. These include users' data files and standard program files.

shareware Software that's freely redistributable but for which the author requests payment.

shell A program that provides users with the ability to run programs, manipulate files, and so on.

shell script A program written in a language that's built into a shell.

signal In reference to processes, a signal is a code that the kernel uses to control the termination of the process or to tell it to perform some task. Signals can be used to kill processes.

SIMM See *Single Inline Memory Module (SIMM)*.

Simple Mail Transfer Protocol (SMTP) The most common push mail protocol on the Internet. SMTP is implemented in Linux by servers like sendmail, Postfix, Exim, and qmail.

Simple Network Management Protocol (SNMP) A protocol for reporting on the status of a computer over a network, or adjusting a computer's settings remotely.

Single Inline Memory Module (SIMM) A small circuit board that holds memory chips for easy installation in a computer. SIMMs come in 30- and 72-pin varieties. They were used on 80386, 80486, many Pentium-level, and a few Pentium II systems. They are still used in many peripherals such as printers.

slave The second of two possible devices on an EIDE chain. The slave device has a higher Linux device number than the master device does.

Small Computer System Interface (SCSI) An interface standard for hard disks, CD-ROM drives, tape drives, scanners, and other devices.

smart filter A program, run as part of a print queue, that determines the type of a file and passes it through appropriate programs to convert it to a format that the printer can handle.

SMB See *Server Message Block (SMB)*.

SMTP See *Simple Mail Transfer Protocol (SMTP)*.

SNMP See *Simple Network Management Protocol (SNMP)*.

soft link A type of file that refers to another file on the computer. When a program tries to access a soft link, Linux passes the contents of the linked-to file to the program. If the linked-to program is deleted, the soft link stops working. Deleting the soft link doesn't affect the original file. Also referred to as a symbolic link. See also *hard link*.

software modem Modems that implement key functionality in software that must be run by the host computer. These modems require special drivers, which are uncommon in Linux.

software rot Deterioration of software caused by filesystem errors, accidental overwriting of software files, and so on.

source package A file that contains complete source code for a program. The package may be compiled into a binary package, which can then be installed on the computer.

source RPM A type of source package that uses the RPM file format.

spam Unsolicited bulk e-mail. (When capitalized, this word is a trademark that refers to a canned meat made by Hormel.)

spawn The action of one process starting another.

spool directory A directory in which print jobs, mail, or other files wait to be processed. Spool directories are maintained by specific programs, such as the printing system or SMTP mail server.

SPX See *Sequenced Packet Exchange (SPX)*.

SSH See *Secure Shell (SSH)*.

stable kernel See *release kernel*.

standard input The default method of delivering input to a program. It normally corresponds to the keyboard at which you type.

standard output The default method of delivering purely text-based information from a program to the user. It normally corresponds to a text-mode screen, xterm window, or the like.

startup script A script that controls part of the Linux boot process.

static files Files that don't change except through direct intervention of the system administrator. Examples include system binary and configuration files. Static files may be stored on partitions that are mounted read-only.

statically linked Programs that incorporate library code into their finished binary executables. Statically linked programs are larger and use more RAM than their dynamically linked counterparts, but they don't depend upon the underlying library existing on the computer.

sticky bit A special file permission bit that's most commonly used on directories. When set, only a file's owner may delete the file, even if the directory in which it resides can be modified by others.

subdomain A subdivision of a domain. A subdomain may contain computers or subdomains of its own.

subnet mask See *network mask*.

SUID See *set user ID (SUID)*.

super server A server that listens for network connections intended for other servers and launches those servers. Examples on Linux are `inetd` and `xinetd`.

superuser A user with extraordinary rights to manipulate critical files on the computer. The superuser's username is normally `root`.

swap file A disk file configured to be used as swap space.

swap partition A disk partition configured to be used as swap space.

swap space Disk space used as an extension to a computer's RAM. Swap space allows a system to run more programs or to process larger data sets than would otherwise be possible.

SWAT See *Samba Web Administration Tool (SWAT)*.

switch A type of network hardware that serves as a central exchange point in a network. Each computer has a cable that links to the switch, so all data pass through the switch. A switch usually sends data only to the computer to which it's addressed. See also hub.

symbolic link See *soft link*.

system cron job A cron job that handles system-wide maintenance tasks, like log rotation or deletion of unused files from `/tmp`. See also *user cron job*.

System V (SysV) A form of AT&T Unix that defined many of the standards used on modern Unixes and Unix clones, such as Linux.

SysV See *System V (SysV)*.

SysV startup script A type of startup script that follows the System V startup standards. Such a script starts one service or related set of services.

tarball A package file format based on the `tar` utility. Tarballs are easy to create and are readable on any version of Linux, or most non-Linux systems. They contain no dependency information and are not easy to remove once installed, however.

TCP/IP See *Transmission Control Protocol/Internet Protocol (TCP/IP)*.

Telnet A protocol used for performing remote text-based logins to a computer. Telnet is a poor choice for connections over the Internet because it passes all data, including passwords, in an unencrypted form, which is a security risk. See also *Secure Shell (SSH)*.

terminal program A program that's used to initiate a simple text-mode connection between two computers, especially via a modem or RS-232 serial connection.

text editor A program for editing text files on a computer.

title bar See *drag bar*.

Token Ring A type of network hardware that supports speeds of up to 16Mbps on twisted-pair cabling.

Transmission Control Protocol/Internet Protocol (TCP/IP) The network stack that's most popular in 2001, and the one upon which the Internet is built.

UID See *user ID (UID)*.

umask See *user mask (umask)*.

Universal Serial Bus (USB) A type of interface for low- to medium-speed external devices, such as keyboards, mice, cameras, modems, scanners, and removable disk drives. Linux added USB support with the 2.2.18 and 2.4.*x* kernels.

unshareable files Files that are not reasonably shared with another computer. The most common example is system configuration files.

USB See *Universal Serial Bus (USB)*.

user An individual who has an account on a computer. This term is sometimes used as a synonym for *account*.

user cron job A cron job created by an individual user to handle tasks for that user, such as running a CPU-intensive job late at night when other users won't be disturbed by the job's CPU demands. See also *system cron job*.

user ID (UID) A number associated with a particular account. Linux uses the UID internally for most operations, and it converts to the associated username only when interacting with people.

user mask (umask) A bit pattern representing the permission bits that are to be removed from files created from a process.

user private group A group strategy in which every user is associated with a unique group. Users may then add other users to their groups in order to control access to files on an individual basis.

username The name associated with an account, such as `theo` or `miranda`. Linux usernames are case-sensitive and may be from 1 to 32 characters in length, although they're usually entirely lowercase and no longer than 8 characters.

UTC See *Coordinated Universal Time (UTC)* and *Greenwich Mean Time*.

variable In computer programming or scripting, a "placeholder" for data. Variables may change from one run of a program to another, or even during a single run of a program.

variable files Files that may vary without direct intervention by the system administrator. Examples include users' data files and log files. Variable files must be stored on partitions mounted for read/write access.

virtual filesystem A filesystem that doesn't correspond to a real disk partition, removable disk, or network export. `/proc` is a virtual filesystem in Linux that provides access to information on the computer's hardware.

widget A GUI control, such as a button to be pressed.

widget set A set of programming tools that provide useful GUI elements, such as buttons, menu bars, and dialog boxes. A widget set is used by programmers to implement common GUI features.

wildcard A character or group of characters that, when used in a shell as part of a filename, match more than one character. For instance, `b??k` matches `book`, `back`, and `buck`, among many other possibilities.

window manager A program that provides decorative and functional additions to the plain windows provided by X. Linux supports dozens of window managers.

workstation A type of computer that's used primarily by one individual at a time to perform productivity tasks, such as drafting, scientific or engineering simulations, or writing. See also *desktop computer*.

X Shortened form of X Window System.

X client A program that uses X to interact with the user.

X Display Manager (XDM) A program that directly accepts either remote or local logins to a computer using X without involving a text-based login protocol like Telnet or SSH. Some Linux distributions use the original XDM program, but other distributions use variants like the GNOME Display Manager (GDM) or KDE Display Manager (KDM), both of which provide additional features.

XDM See *X Display Manager (XDM)*.

X server A program that implements X for a computer; especially the component that interacts most directly with the video hardware.

X Window System The GUI environment for Linux. The X Window System is a network-aware, cross-platform GUI that relies upon several additional components (such as a window manager and widget sets) to provide a complete GUI experience.

XFree86 A set of X servers and related utilities for Linux and other OSs.

xterm A program that allows the running of text-mode programs in X. As used in this book, xterm refers both to the original xterm program and to various programs that provide similar functionality.

zombie A process that's been killed, but for which the parent hasn't removed the process table entries. The zombie shows up in `ps` listings, but it doesn't normally consume CPU time or memory. Zombies cannot be killed with `kill`; you must kill the parent process to remove the zombie.