ROSS BRUNSON
SEAN WALBERG

CompTIA
Linux+
POWERED BY LPI

Cert Guide

Learn, prepare, and practice for exam success

CompTIA® Linux+™/ LPIC-1

Exams LX0-103 & LX0-104/
101-400 & 102-400

DVD INCLUDED

PEARSON IT
CERTIFICATION

CompTIA® Linux+ / LPIC-1 Cert Guide

Ross Brunson
Sean Walberg

PEARSON

800 East 96th Street
Indianapolis, Indiana 46240 USA

CompTIA Linux+ / LPIC-1 Cert Guide (Exams LX0-103 & LX0-104/101-400 & 102-400)

Ross Brunson

Sean Walberg

ISBN-13: 978-0-7897-5455-4

ISBN-10: 0-7897-5455-X

Library of Congress Control Number: 2015945796

Printed in the United States of America

First Printing December 2015

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

This book is designed to provide information about Linux. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information is provided on an "as is" basis. The authors and publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the DVD or programs that may accompany it. The opinions expressed in this book belong to the authors and are not necessarily those of Pearson IT Certification.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Publisher
Paul Boger

Associate Publisher
Dave Dusthimer

Executive Editor
Mary Beth Ray

Development Editor
Eleanor C. Bru

Managing Editor
Sandra Schroeder

Project Editor
Seth Kerney

Copy Editor
Geneil Breeze

Indexer
Tim Wright

Proofreader
Brad Herriman

Technical Editors
Ted Jordan
William "Bo" Rothwell

Publishing Coordinator
Vanessa Evans

Multimedia Developer
Lisa Matthews

Interior Designer
Mark Shirar

Cover Designer
Alan Clements

Composition
Trina Wurst

Contents at a Glance

Contents

About the Authors

Ross Brunson has more than 20 years of experience as a Linux and Open Source trainer, training manager, and technologist and is author of the popular LPIC-1 Exam Cram (QUE Publishing).

Ross is currently senior training/certification engineer at SUSE and recently spent almost five years as the director of member services for the Linux Professional Institute, where he contributed to placing several LPI courses into the Cisco Networking Academy, conducted dozens of Train-the-Trainer sessions, and provided sales enablement support for the worldwide Master Affiliate network spanning more than 100 countries.

Ross holds a number of key IT certifications and is also author of several successful technical books and dozens of technical courses for major organizations (including the first LPI Certification Bootcamps). He is skilled at both contributing to and building community around IT products.

He lives in Paradise Valley, Montana, with his family and enjoys traveling far and wide, winter sports, and photography.

Sean Walberg has more than 20 years of experience as a Linux administrator, network engineer, and software developer. He has written extensively on Linux certification for IBM and NetDevGroup, and has contributed to other books both as an author and technical reviewer.

Sean currently works at Northfield IT and is responsible for infrastructure automation for a large professional sports league. Using tools like Ruby, shell scripts, and Chef, he automates the creation and maintenance of more than a thousand servers and the associated network infrastructure. Sean works closely with developers to scale applications to the demands of an internationally recognized series of web properties.

He lives in Northern Virginia with his wife and three sons.

About the Contributing Author

At the impressionable age of 14, **William "Bo" Rothwell** crossed paths with a TRS-80 Micro Computer System (affectionately known as a "Trash 80"). Soon after the adults responsible for Bo made the mistake of leaving him alone with the TRS-80. He immediately dismantled it and held his first computer class, showing his friends what made this "computer thing" work.

Since this experience, Bo's passion for understanding how computers work and sharing this knowledge with others has resulted in a rewarding career in IT training. His experience includes Linux, Unix, and programming languages such as Perl, Python, Tcl, and BASH. He is the founder and president of One Course Source, an IT training organization.

About the Technical Reviewer

Ted Jordan has more than 25 years of programming, administration, and training experience in UNIX, IRIX, Solaris, and Linux. His career spans from General Motors, Silicon Graphics, to SUN. He holds the LPIC, Linux+, and SUSE Linux certifications. He is the founder and president of two successful startups, the latest being Funutation Tech Camps where he teaches kids to code computer games.

Ted lives with his family near Worcester, Massachusetts, and enjoys tennis, golf, and karaoke.

Dedications

Ross Brunson: *To my good friends, Andres and Ken, we few, we happy few. With love and respect to my wife and daughter, for putting up with my being locked in my office writing and editing while the sun shone and breezes blew. To every student/attendee/customer I've ever taught a Linux topic to, it's really all for you.*

Sean Walberg: *To my amazingly beautiful and intelligent wife, Rebecca. The completion of this book happens to coincide with the start of our new adventure together, and I can think of no one else I'd like to share it with.*

Acknowledgments

Ross Brunson: To the ultimate nerd-herders, Ellie Bru and Mary Beth Ray; Ellie for her ability to take the peeping and muttering of technical geeks and make it something useful, workable, and often profound, and Mary Beth for believing in authors and technologists, even when we break her heart by blowing out deadlines and not doing what we say we will on time.

To my little brother, Leighton, who will make the most awesome history professor one day.

To Sean Walberg, who I have known and respected over several book projects, years of interactions in the LPI community, and all the way back to the Cramsession days with Qcumber and the gang.

To Bo Rothwell and Ted Jordan, awesome technical editors and great guys, as well as two of the best technical trainers it is my pleasure to know.

Sean Walberg: To the crew at Pearson, most notably Geneil, Ellie, and Mary Beth: This project is better because of your patience and input.

The technical editors, Ted and Bo, also deserve special mention. Not only did you fix my technical missteps, but your years of experience as trainers pointed out where I was using some words that were going to confuse new Linux users.

Finally, my knowledge of Linux wasn't earned alone. It came through long nights, hard work, and lots of arguing with people like Marc Caron, Hany Fahim, Patrick leMaistre, Daniel Little, Dave Rose, and of course, my co-author Ross Brunson who I'm happy to have known for more than 15 years.

We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@pearsonitcertification.com
Mail: Pearson IT Certification
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Register your copy of CompTIA Linux+ / LPIC-1 Cert Guide at www.pearsonitcertification.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780789754554) and click Submit. Once the process is complete, you will find any available bonus content under "Registered Products." Be sure to check the box that you would like to hear from us in order to receive exclusive discounts on future editions of this product.

Introduction

This book was written to help people learn to use Linux. Not just learning Linux by memorizing commands, but learning Linux by understanding how the parts are put together. Approaching Linux from this perspective means that you'll know where to look when you run up against something new and are better suited to handle problems as they come up. The authors of this book are experienced writers, but more importantly, are in the trenches every day.

The CompTIA Linux+ exams LX0-103 and LX0-104 and Linux Professional Institute LPIC Level 1 exams 101-400 and 102-400 (which are identical) encompass the knowledge necessary to become an entry level Linux administrator. There are certainly other books that cover this material, but this is the one that looks beyond the exam to preparing people for the Linux workforce.

You don't need to be taking either the Linux+ or LPIC exams to get use out of this book. Concepts such as filesystems, hardware, shell usage, and managing email systems are needed in the workforce, and we, as authors, have endeavored to produce a book that is just as helpful to all new Linux users.

Goals and Methods

The goal of this book is to provide a guided tour of the Linux operating system with an eye to achieving an entry-level certification at the completion of the book. Readers with no intention of writing an exam will still find this book helpful as the certification content, by design, closely maps to the skills required by a Linux administrator. The authors also hope that the examples and practical advice in this text prove valuable well after the reader is done with the book.

The Linux+ and LPIC Level 1 certification exams are broken into specific topics that build upon each other, and the book does its best to mirror those. Not only does this provide a natural progression to learning Linux, but for those who are taking the exam, allows them to focus on troublesome areas.

Linux commands and their output are interspersed with the text to provide concrete examples right next to the description. Examples, for the most part, are adaptations of real world usage rather than being contrived. And since no good Linux graybeard should take himself too seriously, the authors have done their best to inject some levity into the discussion.

Who Should Read This Book?

This book was written for people who want to learn Linux—people just getting into the information technical field, Windows administrators who want to branch out to Linux, or students looking to understand Linux. Even if you're not taking the Linux+ or LPIC Level 1 exams you'll find this book helpful.

The first half of the book focuses on concepts and basic command usage, while the second half turns the attention to applications found in a typical Linux environment. People looking to be more competent Linux users, as opposed to administrators, will find immense benefit in the first half of the book, but will still appreciate the view of what else can be done on Linux provided by the second half.

Managers looking for some Linux familiarity will also find this book helpful because of the abundant examples and real world applications that will help them to speak in the same language as their more technical reports.

This book was not meant just to be read and cast aside. Instead, it can be a reference for common command usage and some basic application administration.

How To Use This Book

The best way to learn Linux is to use Linux. There are many examples within the text, from simple one-line commands to reusable scripts. Find yourself a Linux distribution such as Fedora, Ubuntu, Debian, or openSUSE. They're free and run on most hardware.

If you don't have a spare computer on which to install Linux you can try a LiveCD, which is a bootable image that runs entirely in memory. Most distributions offer a LiveCD download. Alternatively, you can run Linux in a virtual machine with software like VirtualBox (http://www.virtualbox.org).

All the software shown in this book is available on the most basic of Linux distributions and does not need an extra download. However Chapter 13, "Basic SQL Management," offers a sample database that you can use to follow the examples. To install this, download the compressed attachment from http://www.pearsonitcertification.com/title/9780789754554. Inside the compressed file are two database files. The first, called **lpic_basic.sqlite3**, contains the data for the first part of the chapter. The second includes the additional data for the later examples. Instructions for using the databases are found in Chapter 13.

Above all, experiment with your Linux system. Try a couple of different distributions. Run the commands in this book and see whether you can come up with your own examples. Poke around in the configuration files and explore alternative uses for the commands in this book.

How This Book Is Organized

Although you could read this book cover-to-cover, it is designed to be flexible and allow you to easily move between chapters and sections of chapters to cover only the material you need. If you do intend to read them all, the order in which they are presented is an excellent sequence.

Chapters 1 through 12 cover the following topics:

- **Chapter 1, "Installing Linux":** This chapter teaches you the basics of how a Linux system is installed. Core topics like hard disk partitioning and dealing with hardware are the focus of this chapter.
- **Chapter 2, "Boot Process and Runlevels":** The Linux system has a specific order in which things happen both for starting up and shutting down. This chapter discusses the way these processes work and how to make changes so that you get the services that you need on your system.
- **Chapter 3, "Package Install and Management":** Finding, installing, and configuring software is a big part of the system administrator's job description. This chapter walks you through the usage of both the Debian and RedHat package systems.
- **Chapter 4, "Basic Command Line Usage":** This chapter takes you through the basics of working on the Linux command line, including running applications and some commands to orient yourself on a new system. The work here forms the basis of the next three chapters.
- **Chapter 5, "File Management":** This chapter delves into the commands that manipulate files. You create, delete, compress, move, and look at the files on disk and gain a solid understanding of how the Linux filesystems operate.
- **Chapter 6, "Text Processing/Advanced Command Line":** The Linux command line is a programming environment that lets you do complicated tasks with a few keystrokes. This chapter introduce you to the most powerful feature of the shell of all: chaining together individual commands into increasingly powerful command lines. Along the way you learn how to search through text using regular expressions.
- **Chapter 7, "Process Management":** Things that run on a Linux system are called processes, and this chapter teaches you how to manipulate these processes. You learn how to start and stop processes, run them in the background, and see which ones are taking the most resources from your computer.
- **Chapter 8, "Editing Text":** This chapter teaches you to be productive in the vim editor. Vim makes repetitive tasks a breeze and lets you perform powerful edits on text files without moving the mouse. As most configuration and

programming on Linux is through a text file, an administrator who can wield a text editor with efficiency is one who has her work done on time.

- **Chapter 9, "Partitions and Filesystems":** This chapter takes a deep dive into how a Linux system uses disks. You learn how filesystems work and how to add and remove capacity from a Linux workstation.
- **Chapter 10, "Permissions and Ownership":** Linux was built as a multiuser system from the very beginning, so an understanding of how access to resources is granted and checked is important to maintain the security of your data and the sanity of your users. This chapter investigates the Linux permission model along with the commands used to check and set permissions.
- **Chapter 11, "Customizing Shell Environments":** This chapter explores ways that you can customize your command line, such as by making shorter versions of longer commands or adding your own functions to the command line. Here, we also look at the roles played by internationalization and localization, which are methods that let the shell adapt to different languages and countries without needing to maintain multiple installations.
- **Chapter 12, "Shell Scripting":** The Linux shell is actually a sophisticated programming environment and this chapter shows you the basics. You don't have to be a programmer to write shell scripts—this chapter starts with the most basic script and works from there.
- **Chapter 13, "Basic SQL Management":** The Structured Query Language is a way that databases query and manipulate data. This chapter, through real world examples, teaches you the basics of SQL so that you can more effectively help your users and answer questions about your own data.
- **Chapter 14, "Configuring User Interfaces and Desktops":** Linux isn't just a command line system—there are many graphical tools from word processors to video games. This chapter shows you how to use Linux in a graphical mode.
- **Chapter 15, "Managing Users and Groups":** Users and groups are the other half of the Linux permissions model that was started in Chapter 10. This chapter teaches the administrative tasks associated with managing the users on your system.
- **Chapter 16, "Schedule and Automate Tasks":** This chapter walks you through the various ways that Linux systems can run tasks without user intervention, such as to process statistics from logs while you're sleeping.
- **Chapter 17, "Configuring Print and Email Services":** This chapter looks at two basic services that Linux is often called to solve: printing and email. With printing, you learn how the Common Unix Printing System (CUPS) is put together and how it can be used to manage printing for a single system or

a large enterprise. In the email half of the chapter you learn how email works and what software is used on Linux to perform the various roles in an Internet email system. You also see how to do basic account management in an email system.

- **Chapter 18, "Logging and Time Services":** Logs provide a detailed accounting of what happened when you weren't looking. This chapter explains the Linux logging systems and how to configure and use them. Additionally you learn how time is kept on a Linux system and how different Linux systems can talk to coordinate their time.

- **Chapter 19, "Networking Fundamentals":** A Linux system that provides network services is only as good as its network configuration. This chapter gives you the solid understanding of networking needed to determine whether Linux or the network is causing a problem. You also learn about the various services used to connect computers on a network.

- **Chapter 20, " System Security":** Security is all about assessing the risk to your machine and keeping the bad guys out. In this chapter you learn how to assess the security of your system, lock down services to only people you want, and encrypt your data from prying eyes.

- **Chapter 21, "Final Preparation":** In this final chapter you find exam questions that challenge your understanding of the material and provide a test that assesses your readiness to take either the LPIC 101 or Linux+ exams.

- **Glossary**: The glossary defines all terms that you were asked to define at the end of each chapter.

Each chapter follows the same format and incorporates the following tools to assist you by assessing your current knowledge and emphasizing specific areas of interest within the chapter:

- **"Do I Know This Already?" Quizzes:** Each chapter begins with a quiz to help you assess your current knowledge of the subject. The quiz is divided into specific areas of emphasis that enable you to best determine where to focus your efforts when working through the chapter.

- **Foundation Topics:** The foundation topics are the core sections of each chapter. They focus on the specific commands, concepts, or skills that you must master to successfully prepare for the examination.

- **Exam Preparation Tasks:** At the end of the foundation topics, the Exam Preparation Tasks highlight the key topics from the chapter and lists the pages where you can find them for quick review. This section also provides a list of key terms that you should be able to define in preparation for the exam. It is unlikely that you will be able to successfully complete the certification exam by just studying the key topics and key terms, although they are a good tool for last-minute preparation just before taking the exam. For a thorough understanding of how to prepare for the exam, see Chapter 21.

- **Review Questions:** Questions at the end of each chapter measure your understanding of the topics discussed in the chapter.
- **DVD-Based Practice Exam:** This book includes a DVD containing several interactive practice exams. It is recommended that you continue to test your knowledge and test-taking skills by using these exams. You will find that your test-taking skills improve by continued exposure to the test format. Remember that the potential range of exam questions is limitless. Therefore, your goal should not be to "know" every possible answer but to have a sufficient understanding of the subject matter so that you can figure out the correct answer with the information provided.

Pearson IT Certification Practice Test Engine and Questions on the DVD

The DVD in the back of the book includes the Pearson IT Certification Practice Test engine—software that displays and grades a set of exam-realistic multiple-choice questions. Using the Pearson IT Certification Practice Test engine, you can either study by going through the questions in Study Mode, or take a simulated exam that mimics real exam conditions. You can also serve up questions in a Flash Card Mode, which displays just the question and no answers, challenging you to state the answer in your own words before checking the actual answers to verify your work.

The installation process requires two major steps: installing the software and then activating the exam. The DVD in the back of this book has a recent copy of the Pearson IT Certification Practice Test engine. The practice exam (the database of exam questions) is not on the DVD.

Note The cardboard DVD case in the back of this book includes the DVD and a piece of paper. The paper lists the activation code for the practice exam associated with this book. Do not lose the activation code. On the opposite side of the paper from the activation code is a unique, one-time-use coupon code for the purchase of the Premium Edition eBook and Practice Test.

Install the Software from the DVD

The Pearson IT Certification Practice Test is a Windows-only desktop application. Unfortunately, you cannot easily run this .exe on a Linux machine. You can run it on a Mac using a Windows virtual machine, but it was built specifically for the PC platform. The minimum system requirements are as follows:

- Windows 10, Windows 8.1, Windows 7, or Vista (SP2)
- Microsoft .NET Framework 4.0 Client
- Pentium-class 1 GHz processor (or equivalent)
- 512 MB RAM
- 650 MB disk space plus 50 MB for each downloaded practice exam
- Access to the Internet to register and download exam databases

The software installation process is routine as compared with other software installation processes. If you have already installed the Pearson IT Certification Practice Test software from another Pearson product, there is no need for you to reinstall the software. Simply launch the software on your desktop and proceed to activate the practice exam from this book by using the activation code included in the DVD sleeve.

The following steps outline the installation process:

1. Insert the DVD into your PC.
2. The media interface that automatically runs allows you to access and use all DVD-based features, including the exam engine and sample content from other Cisco self-study products. From the main menu, click the Install the Exam Engine option.
3. Respond to windows prompts as with any typical software installation process.

The installation process gives you the option to activate your exam with the activation code supplied on the paper in the DVD sleeve. This process requires that you establish a Pearson website login. You need this login to activate the exam, so please do register when prompted. If you already have a Pearson website login, there is no need to register again. Just use your existing login.

Activate and Download the Practice Exam

Once the exam engine is installed, you should then activate the exam associated with this book (if you did not do so during the installation process) as follows:

1. Start the Pearson IT Certification Practice Test software from the Windows Start menu or from your desktop shortcut icon.
2. To activate and download the exam associated with this book, from the My Products or Tools tab, click the Activate Exam button.
3. At the next screen, enter the activation key from the paper inside the cardboard DVD holder in the back of the book. Once entered, click the Activate button.
4. The activation process downloads the practice exam. Click Next and then click Finish.

When the activation process completes, the My Products tab should list your new exam. If you do not see the exam, make sure that you have selected the My Products tab on the menu. At this point, the software and practice exam are ready to use. Simply select the exam and click the Open Exam button.

To update a particular exam you have already activated and downloaded, display the Tools tab and click the Update Products button. Updating your exams ensures that you have the latest changes and updates to the exam data.

If you want to check for updates to the Pearson Cert Practice Test exam engine software, display the Tools tab and click the Update Application button. You can then ensure that you are running the latest version of the software engine.

Activating Other Exams

The exam software installation process and the registration process, only have to happen once. Then, for each new exam, only a few steps are required. For instance, if you buy another Pearson IT Certification Cert Guide, extract the activation code from the DVD sleeve in the back of that book; you do not even need the DVD at this point. From there, all you have to do is start the exam engine (if not still up and running) and perform steps 2 through 4 from the previous list.

Certification Exam Topics and This Book

The questions for each certification exam are a closely guarded secret. However, we do know which topics you must know to successfully complete this exam. CompTIA and LPI publishes them as an exam blueprint.

Tables I.1 and I.2 list the exam topics for each exam.

Table I.1 CompTIA Linux+ (LX0-103) and LPIC-1 (101-400) Exam

Exam Topics for CompTIA Linux+ (LX0-103) and LPIC-1 (101-400) Exam
Topic 101: System Architecture
101.1 Determine and configure hardware settings
101.2 Boot the system
101.3 Change run levels/boot targets and shutdown or reboot system
Topic 102: Linux Installation and Package Management
102.1 Design hard disk layout
102.2 Install a boot manager
102.3 Manage shared libraries
102.4 Use Debian package management
102.5 Use RPM and YUM package management
Topic 103: GNU and Unix Commands
103.1 Work on the command line
103.2 Process text streams using filters
103.3 Perform basic file management
103.4 Use streams, pipes, and redirects
103.5 Create, monitor, and kill processes
103.6 Modify process execution priorities
103.7 Search text files using regular expressions
103.8 Perform basic file editing operations using vi
Topic 104: Devices, Linux Filesystems, Filesystem Hierarchy Standard
104.1 Create partitions and filesystems
104.2 Maintain the integrity of filesystems
104.3 Control mounting and unmounting of filesystems
104.4 Manage disk quotas
104.5 Manage file permissions and ownership
104.6 Create and change hard and symbolic links
104.7 Find system files and place files in the correct location

Table I-2 CompTIA Linux+ (LX0-104) and LPIC-1 (102-400)

Exam Topics for CompTIA Linux+ (LX0-104) and LPIC-1 (102-400)
Topic 105: Shells, Scripting, and Data Management
105.1 Customize and use the shell environment
105.2 Customize or write simple scripts
105.3 SQL data management
Topic 106: User Interfaces and Desktops
106.1 Install and configure X11
106.2 Set up a display manager
106.3 Accessibility
Topic 107: Administrative Tasks
107.1 Manage user and group accounts and related system files
107.2 Automate system administration tasks by scheduling jobs
107.3 Localization and internationalization
Topic 108: Essential System Services
108.1 Maintain system time
108.2 System logging
108.3 Mail Transfer Agent (MTA) basics
108.4 Manage printers and printing
Topic 109: Networking Fundamentals
109.1 Fundamentals of Internet protocols
109.2 Basic network configuration
109.3 Basic network troubleshooting
109.4 Configure client side DNS
Topic 110: Security
110.1 Perform security administration tasks
110.2 Set up host security
110.3 Securing data with encryption

Assessing Exam Readiness

Exam candidates never really know whether they are adequately prepared for the exam until they have completed about 30% of the questions. At that point, if you are not prepared, it is too late. The best way to determine your readiness is to work through the "Do I Know This Already?" quizzes at the beginning of each chapter and review the foundation and key topics presented in each chapter. It is best to work your way through the entire book unless you can complete each subject without having to do any research or look up any answers.

Exam Registration

For LPI exams, start at lpi.org to get a member ID and a link to pearsonvue.com/lpi/ to schedule an exam. For the Linux+ variants, sign up directly from https://certification.comptia.org/certifications/linux.

Where Are the Companion Content Files?

Register this print version of *CompTIA Linux+ / LPIC-1 Cert Guide* to access the content from the DVD online.

This print version of this title comes with a disc of companion content. You have online access to these files by following these steps:

1. Go to www.pearsonITcertification.com/register and log in or create a new account.
2. Enter the ISBN: 9780789754554.
3. Answer the challenge question as proof of purchase.
4. Click on the Access Bonus Content link in the Registered Products section of your account page to be taken to the page where your downloadable content is available.

Please note that many of our companion content files can be very large, especially image and video files.

If you are unable to locate the files for this title by following these steps, please visit www.pearsonITcertification.com/ contact and select the Site Problems/Comments option. Our customer service representatives will assist you.

This page intentionally left blank

This chapter covers the following topics:

- Filesystem Overview
- File Management Commands
- Where Are Those Files?
- Backup Commands

This chapter covers the following objectives:

- Perform basic file management: 103.3
- Create and change hard and symbolic links: 104.6
- Find system files and place files in the correct location: 104.7

CHAPTER 5

File Management

Most of what you do on a Linux machine involves manipulating files in some manner. You have to know where certain files go, such as binaries, configuration, and user data. You also need to be able to manipulate files from the command line rather than a GUI.

"Do I Know This Already?" Quiz

The "Do I Know This Already?" quiz enables you to assess whether you should read this entire chapter or simply jump to the "Exam Preparation Tasks" section for review. If you are in doubt, read the entire chapter. Table 5-1 outlines the major headings in this chapter and the corresponding "Do I Know This Already?" quiz questions. You can find the answers in Appendix A, "Answers to the 'Do I Know This Already?' Quizzes and Review Questions."

Table 5-1 "Do I Know This Already?" Foundation Topics Section-to-Question Mapping

Foundation Topics Section	Questions Covered in This Section
Filesystem Overview	1, 3
File Management Commands	2, 4-6
Where Are Those Files?	7-8
Backup Commands	9-11

1. Files that change often should go under:

   a. /usr

   b. /proc

   c. /root

   d. /var

2. Your shell is in /usr/local. You type **cd ../bin**. Which directory is shown when you type **pwd**?
   a. /bin
   b. /usr/bin
   c. /usr/local/bin
   d. Nothing, this command returns an error.

3. Which of the following directories should be on the same partition as the root?
   a. /boot
   b. /usr
   c. /home
   d. /sbin

4. You happen across a file in a directory called **foo**. What is a good way to find out what the file is or does?
   a. **file foo**
   b. **/foo**
   c. **cat foo**
   d. **which foo**

5. What command would be used to update the date on a file?
   a. **tar**
   b. **file**
   c. **date**
   d. **touch**

6. You are trying to create a new series of nested directories: /a/b/c/d/. What is the fastest way to create this nested directory set?
   a. **mkdir /a; mkdir /a/b; mkdir /a/b/c; mkdir /a/b/c/d**
   b. **mkdir /a/b/c/d**
   c. **mkdir -p /a/b/c/d**
   d. **md /a/b/c/d**

7. You know that you have multiple copies of the **doit** command on your system. How do you find which one you will run if you type **doit** at the command line?

    a. **whereis doit**

    b. **locate doit**

    c. **find doit**

    d. **which doit**

8. You know that you downloaded a file called backup.tar.gz this morning but can't remember where you put it. Which is the most appropriate command to find the file?

    a. **find / -name backup.tar.gz**

    b. **find backup.tar.gz**

    c. **locate backup.tar.gz**

    d. **whereis backup.tar.gz**

9. You want to package up Fred's home directory on a USB stick to send with him as he's leaving your company. Which command is the best? Hurry, because there's cake!

    a. **find /home/fred | tar -czf > /media/removable/fred.tar.gz**

    b. **tar -czf /home/fred > /media/removable/fred.tar.gz**

    c. **cd /home/; tar -cjf /media/removable/fred.tar.bz2 fred**

    d. **cd /home/fred tar -cjf /media/removable/fred.tar.bz2 ***

10. What does the command **tar -tf archive.tar etc/pine.conf** do?

    a. Makes a new archive called archive.tar containing /etc/pine.conf

    b. Adds etc/pine.conf to archive.tar

    c. Checks to see whether etc/pine.conf is inside the archive

    d. Extracts etc/pine.conf from archive.tar

11. Which compression utility offers the highest level of compression?

    a. **bzip2**

    b. **gzip**

    c. **compress**

    d. **cpio**

Foundation Topics

Filesystem Overview

The filesystem's structure starts with the root of the filesystem, which is denoted by the forward slash character (/). Every item on the filesystem is accessible by a single unique path from the root of the system, such as /usr/local/bin/foobar, no matter which device that file is stored on.

Unix evolved its own set of traditions as to where certain files would go. The fragmentation of the commercial and academic Unixes led to differences in conventions depending on which flavor of Unix you were using.

Linux borrows practices from many different Unixes and has fragmentation of its own in the form of different distributions. The community started working on a standard for filesystem layout called the *File System Hierarchy Standard (FHS)* to make it easier for both people and software to know where files can be found.

The latest FHS is always found at http://www.pathname.com/fhs/.

LPI bases the exam questions about the directory structure from the FHS 2.3. The FHS isn't really a standard but a firm set of suggestions that most, but not all, distribution vendors obey. A good number of questions on the exams reference the FHS.

What Belongs Where

The exams make somewhat of a big deal about what the proper directories and locations are for Linux files, but few things are more vexing than to be asked what should positively be in the root (/) directory, or what can be elsewhere.

The Root of the System

Starting in the root (/) directory, the Table 5-2 lists common top-level directories and includes a short explanation for each:

Table 5-2 Common Directories

Directory	Description
bin	Binaries for all users
boot	Kernel, system map, boot files
dev	Device files
etc	Configuration files for the host

Directory	Description
home	Home directories for users
lib	Necessary shared libraries/modules
lost+found	Storage directory for unlinked files (found with fsck)
media	Mount points for removable media
mnt	Temporary mount point for the sysadmin
opt	Third-party application software
proc	Kernel and process information
root	The root user's home directory
sbin	System binaries needed for boot
tmp	Temporary data
usr	Sharable, read-only data and programs, no host-specific data
var	Variable data, logs, Web, FTP, and so on

The exam makes a big deal out of what's optional and required in the **root** (**/**) directory. If you read the FHS 2.3 (highly recommended), you see that the word "optional" appears next to the **/root** and **/home** directories. It is possible that the computer is some kind of application server where users are not expected to log in. This is key because you'll be asked questions about which directories are optional in the root filesystem.

Key Topic

The FHS documentation states, "The contents of the root filesystem must be adequate to boot, restore, recover, and/or repair the system. To boot a system, enough must be present on the root partition to mount other filesystems. This includes utilities, configuration, boot loader information, and other essential start-up data. /usr, /opt, and /var are designed such that they may be located on other partitions or filesystems."

From this statement you can understand which of the preceding directories need to be on the root partition and which can be moved to other partitions.

Classifying Data

FHS makes distinctions between data that changes and data that is static, and data that can be shared versus data that's local to the computer. Data of different categories should be separated into different directories.

Because of the way the FHS is laid out, with the root filesystem being described in section 3 and **/usr** and **/var** happening later, it's easy to misunderstand what is really supposed to be on the root filesystem as opposed to another device that's mounted after boot.

The relationship between **/usr** and **/var** is that, long ago in Unix times, **/usr** used to contain all types of data. The FHS tried to extract the data that changes and is nonsharable to **/var**, leaving **/usr** with just static, sharable data.

Where Programs Live

The FHS does not allow programs to create their individual named directories in the **/usr** section. The subdirectories allowed to exist directly under the **/usr** directory are

- **bin—**Contains user commands
- **include—**Contains header files for C programs
- **lib—**Contains libraries
- **local—**Contains local/sharable programs
- **sbin—**Contains nonessential system binaries
- **share—**Contains data/programs for multiple architectures

Key Topic

The **/usr** section has a location for programs named **/usr/local**. This is for the sysadmin to install software in a place that won't conflict with the distribution files. Programs in the **/usr/local** path are also allowed for sharing among groups of hosts.

For example, say your developers have come up with a program to calculate loans and you want to install it on the workgroup server for other systems to remotely mount and use. Because this is a third-party or custom application, the logical place for it is in **/usr/local/appname**, possibly with a link to the program binary in the **/usr/local/bin** directory (because that's where local binaries are expected to be found).

If given a choice between putting the software package BIGPROG in the /usr/local/BIGPROG section and the /opt/BIGPROG section, it's hard to choose. Read any relevant exam question closely—the main difference being that the /opt section is not considered to be sharable, whereas the /usr section is often shared and mounted by client systems.

File Management Commands

A major section of the 101 exam is dedicated to how to run commands properly with the right options and arguments. As a good sysadmin, you are expected to know how to create, delete, edit, set permissions, display, move, copy, and determine the type of files and programs.

Tips for Working with Linux Files

Because most users and sysadmins come from a Windows or other OS background, a quick set of recommendations for the less-experienced can be of help here:

- **Hidden files aren't really hidden—**They just begin with a ., such as the **.bashrc** and **.bash_profile** files. They are normally not visible unless you explicitly ask for them to be displayed and aren't deleted by commands such as **rm –f *.***.
- **Filenames can contain multiple periods or no period characters—**The filenames **this.is.a.long.file** and **thisisalongfile** are perfectly reasonable and possible.
- **Spaces in filenames look nice, but are a pain to type—**Use an _ or a - instead of spaces because it's neater and easier than prefixing all spaces with a \. (To display a space in a filename, the system shows a space prefixed with a backslash.)
- **File extensions aren't mandatory—**But they are useful for sorting, selection, and copy/move/delete commands, as well as for quickly identifying a file's type.

Basic Navigation

The command to change the current working directory, **cd**, is used frequently and knowing how to move around the filesystem is a main focus of the exams.

The following command simply moves you from wherever you are to the **/etc** directory. This type of move uses absolute pathnames and can be used from within any directory:

```
cd /etc
```

The path is called *absolute* because it defines a path starting at the root of the filesystem. The easy way to tell whether the path is absolute is that it starts with a slash (/).

Moving relatively from the current directory to a subdirectory is quick and easy, such as if you are in the **/etc/** directory and want to change into the **/etc/samba** directory. Here's how:

```
cd samba
```

This is referred to as a *relative path* because the option you pass to the **cd** command is relative to the current directory. You are in **/etc** and moving to **samba** gets you in **/etc/samba**. If you were in **/home** and ran **cd samba** it would not work unless **/home/samba** also existed.

If you get confused as to where you currently are, use the **pwd** command to print the working (current) directory:

```
# pwd
/etc/samba
```

By itself, the **cd** command takes you back to your home directory, wherever you happen to be. The tilde (~) also means "home directory," so **cd ~** takes you to your home directory and **cd ~sean** takes you to Sean's home directory.

Advanced Navigation

It's good to get experience with some complex relative path situations. For example, if you were in the directory **/home1/user1** and wanted to move into the directory **/home2/user2**, which command could be used?

```
$ tree /

/
|-- home1
|    `-- user1
`-- home2
    `-- user2
```

Remember, you aren't using absolute pathnames, just relative pathnames.

The answer is

```
# cd ../../home2/user2
```

Key Topic

Each of the **..** pairs takes you up one level: The first takes you to **/home1** and the second puts you at the root. From there it's relative pathnames. Practice this method, and remember that going up one level in this exercise only got you to the **/home1** directory. This is a relative path because the path does not start with a /. The directory in which you end up depends on where you started.

Though this example of relative and absolute pathnames was used to look at changing directories, it applies to any situation where you're prompted for a filename.

Listing Files and Directories

The ls command is used for listing directories or files, or both.

If you use the **ls** command to see a multicolumn output of the current directory, only the file or directory names are shown, not other details about the file:

```
ls
file1  file2  file3  file4
```

Use the **–l** long listing option to see all the details of a particular file or directory, or set of files or directories in a single column, like so:

```
$ ls -l
total 0
-rw-r--r--    1 root     root            0 Jan 24 18:55 file1
-rw-r--r--    1 root     root            0 Jan 24 18:55 file2
-rw-r--r--    1 root     root            0 Jan 24 18:55 file3
-rw-r--r--    1 root     root            0 Jan 24 18:55 file4
```

Key Topic

The **–l** long listing style is the only way to use the **ls** command and see the permissions, ownership, and link counts for objects. The only other command that can give such information is the **stat** command, which shows a single filesystem object at a time.

Other examples of using the **ls** command include

- **ls /home/user—**Shows a plain listing of that directory.
- **ls –a—**Lists all files, including hidden **.** files.
- **ls –d foo—**Lists just the directory called foo, not the contents.
- **ls –i—**Lists the inode number for the targetfile or directory. Inodes are the way Linux represents a file on disk and are discussed later in the section "Copying Files and Directories."
- **ls –l—**Shows permissions; links; and date, group, and owner information. Permissions dictate who can access the file and are discussed in detail in Chapter 10, "Permissions."
- **ls –lh—**Shows human-readable output of file sizes, in KB, MB, and GB, along with file details.

Chaining the options together produces useful results. For example, if you needed to see all the files (including hidden ones) in the current directory, their permissions, and their inode numbers, you would use the following command:

```
# ls -lai
290305 drwxr-x---  13 root   root      4096 Jan 24 18:55  .
     2 drwxr-xr-x  20 root   root      4096 Jan 24 17:56  ..
292606 -rw-r--r--   1 root   root      1354 Jan 21 00:23  anaconda-ks.
cfg
```

```
292748 -rw-------   1 root   root      3470 Jan 24 18:16 .bash_history
290485 -rw-r--r--   1 root   root        24 Jun 10  2000 .bash_logout
290486 -rw-r--r--   1 root   root       234 Jul  5  2001 .bash_profile
290487 -rw-r--r--   1 root   root       176 Aug 23  1995 .bashrc
290488 -rw-r--r--   1 root   root       210 Jun 10  2000 .cshrc
```

Determining File Types

With no requirement for extensions on Linux files, a tool for easily determining file types is essential. The **file** command can be used to read the file's headers and match that data against a known set of types.

The **file** command uses several possible sources, including the **stat** system call, the magic number file (**/usr/share/magic**), and a table of character sets including ASCII and EBCDIC. Finally, if the file is text and contains recognizable strings from a given programming or other language, it is used to identify the file.

The output can be used, manipulated, and filtered to show you useful things.

For example, simply using the **file** command on a given file shows the type:

```
$ file file1
file1: ASCII text
```

Running the **file** command against a known binary shows various elements about the architecture and layout of the file, such as shown here:

```
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.32, stripped
```

Running the **file** command against a directory full of files is useful for viewing the possible types, but the real gold lies in filtering the output using the pipe operator (|) and the **grep** command, showing only the results that contain the word "empty":

```
$ file /etc/* | grep empty
/etc/dumpdates:                                 empty
/etc/exports:                                   empty
/etc/fstab.REVOKE:                              empty
/etc/motd:                                      empty
/etc/printconf.local:                           empty
```

This is one way of finding empty files that are littering your system. They are probably required in the **/etc** directory but only clutter temporary directories such as **/tmp**.

NOTE The asterisk (*) in the previous command is known as a glob. A *glob* is a wildcard operator that matches some number of files based on a pattern. /etc/* matches all files in the /etc directory such as /etc/foo, /etc/bar, but not /etc/foo/bar!

One thing that's distinct about Linux (and all Unixes) is that the shell is responsible for expanding the glob to the list of files it matches. If you type **ls /tmp/thing*** and there are two files that start with thing such as **thing1** and **thing2**, it's the same thing as if you typed **ls /tmp/thing1 /tmp/thing2**:

```
$ ls thing*
thing1    thing2
```

This globbing feature is why renaming a group of files is harder. In Windows you could type **ren *.foo *.bar** and any file with an extension of foo would then have an extension of bar. In Linux, typing **mv *.foo *.bar** would expand the globs to the list of files matched—***.foo** would match the files you want to rename and ***.bar** would match nothing. This is different from what you might expect! The following output shows this problem.

```
$ ls *.foo *.bar
ls: *.bar: No such file or directory
file1.foo      file2.foo
$ echo mv *.foo *.bar
mv file1.foo file2.foo *.bar
$ mv *.foo *.bar
mv: target `*.bar' is not a directory
```

In the output, the first command shows there are three files with an extension of **foo** but none of **bar**. The **echo** command displays the output that follows it, such that it shows what would be executed if you ran the **mv** command by itself. The *.bar glob shows up because there are no files that match it. The error happens because there is no such directory called ***.bar**.

There are other glob operators. Example 5-1 shows some uses of file globs.

Example 5-1 Examples Using a Glob

```
$ ls
file  file1  file10  file11  file2
$ ls file*
file  file1  file10  file11  file2
$ ls file?
file1  file2
```

```
$ ls *1
file1  file11
$ ls file[123]
file1  file2
```

Example 5-1 starts by listing all the files in the directory. The same list of files is also available with **file***, which matches the word “file” followed by anything, or nothing at all. Note how it includes the bare name “file”. Next the **file?** glob matches anything starting with the word “file” and followed by one character. Both “file” and the files with two-digit numbers in their names are excluded.

Globs don’t have to appear at the end of a filename. ***1** matches anything ending in the number “1”. Finally, **file[123]** uses the square bracket operator that means “any one character from the set”. This matches file1 and file2.

Touching Files

The **touch** command seems odd at first, but it comes in handy often. You give it the name of one or more files, and it creates the files if they don’t exist or updates their timestamps if they do.

There are various reasons to use the **touch** command, such as creating a new blank log file or updating a file’s modification time to use as a reference such as to know the last time a job was run.

To create a new file, you can use the relative pathname for creating one in the current directory:

```
touch filename
```

Or, you can use absolute pathname to create the file, such as shown here:

```
touch /home/rossb/filename
```

Key Topic

Expect to see **touch** on the exams for log file creation, along with using a reference file to mark the last backup. In other words, if a log file is created from a successful backup, that file can be used as a date and time reference file because it occurred at a desirable time.

When you use **touch** on an existing file, the default action is to update all three of the file’s times:

- **access—**The last time a file was written/read from

- **change—**The last time the contents of the file were changed, or that the file's metadata (owner, permission, inode number) was changed
- **modify—**The last time the file's contents were changed

A programmer preparing a new release of a software package would use the **touch** command to ensure that all files have the exact same date and times. Therefore, the release could be referred to by the file date, given multiple revisions.

Setting a file's date is relatively easy; the following command sets **file1**'s date to a particular date and time:

```
touch -t 201501010830 file1
```

The time format used is represented by yyyymmddhhmm, or a four-digit year, two-digit month, two-digit day, two-digit hour, and two-digit minutes.

Reference files are useful, particularly when you just want to have a file or set of files updated to a particular date/time, not the current one. You could use

```
touch -r reffile file2update
```

The date and time of **reffile** is applied to the **file2update** file date and time.

Copying Files and Directories

One aspect of copying an object is that the act creates a new file with a separate inode. This means that the operating system sees the new file as separate from the old one. Contrast this to a move operation where it's the same file with a new name.

When you create an object in a filesystem, it gets its own permissions. **cp** doesn't always copy the permissions over to the new file. This can be done, but it requires the use of the **-p** option to preserve the permissions and ownership. The root user is the only user that can change the ownership of a file; therefore, regular users using this option always own the copied files no matter who the original owner was.

A normal copy is simple to perform. You're essentially causing the file to be replicated to the new location:

```
cp file1 /dir1/file2
```

A few options that make life easier for copying files include

- **-d—**Doesn't follow symbolic links; copies the link instead. Links point one file to another and are explored later in the "Linking Files" section.
- **-f**—Force overwriting existing files.
- **-i—**Interactively asks before overwriting.

- **-l**—Creates a hard link to the source file.
- -r or **–R**—Recursively traverses directories (copying everything).
- **-s**—Creates a symlink to the source file.
- **-u**—Only updates the copy when the source is newer than the target or the target doesn't exist.
- **–x**—Doesn't traverse to filesystems mounted from other devices.

Copying an existing directory to a new one is simple:

```
# cp -r dir1 dir2
```

The **-r** option is necessary because the **cp** command doesn't process directories by default. As long as the target directory does not exist, the previous command makes an identical copy of the source and all subordinate files and directories in the target directory.

Copying a source directory to an existing target directory doesn't attempt an overwrite; it makes the source directory into a new subdirectory of the target.

For example, if you are in the **/test** directory and have the structure shown in the following, you might assume that issuing a **cp –r dir1 dir2** would overwrite **dir2**, or at least prompt you to see whether you wanted to:

```
$ tree .
|-- dir1
|   |-- file1
|   `-- subdir1
`-- dir2
```

When you issue the **cp –r dir1 dir2** command, the filesystem (along with the **cp** command) notices the existing **dir2** entry and automatically drops the source directory into **dir2** as a subdirectory, like this:

```
|-- dir1
|   |-- file1
|   `-- subdir1
`-- dir2
    `-- dir1
        |-- file1
        `-- subdir1
```

Key Topic

The correct way to copy the contents of **dir1** into **dir2**, thereby mirroring **dir1** exactly, is to focus on the word "contents." By suffixing the source (**dir1**) with a forward slash and an asterisk (**dir1/***), you tell the **cp** command to ignore the directory entry and focus on the filenames inside the directory.

With the same initial setup, if you issue the command **cp –r dir1/* dir2**, you get the correct results:

```
$ tree .
|-- dir1
|   |-- file1
|   `-- subdir1
`-- dir2
    |-- file1
    `-- subdir1
```

The inability to properly copy a directory or its contents will come back to haunt you on the exam. In addition, if you see a source directory with only a trailing forward slash (dir1/) but no asterisk, it's identical to using (dir1). In other words, to copy just the contents of a directory, you have to address them specifically with the forward slash and asterisk (dir1/*).

Two special characters used in relative directory naming are often used when copying files. The current directory is represented by a single period (**.**) and the parent directory by two periods (**..**).

For example, if you are currently in the **/home/rossb** directory and want to copy a set of files from the **/home/lukec** directory, you can avoid typing the full path of the current directory with the (**.**) character. Both of these commands perform the same action:

```
cp /home/lukec/*.mp3 .
cp /home/lukec/*.mp3 /home/rossb
```

Moving Objects

Where the **cp** command copies a file by creating a new file, inode, and data, the **mv** command simply changes which directory file contains the file or directory entry or alters the entry in the file if it stays in the same directory. By changing just the metadata that points to the file, moving a file on the same device is quick. If the file move happens across two devices, the file is copied to the new device and deleted from the old one.

Create a file named **file1**; then run the **stat** command on it to check the details, as shown in Example 5-2.

Example 5-2 Running the **stat** Command on **file1**

```
$ touch file1
$ stat file1
  File: `file1'
  Size: 0         Blocks: 0          IO Block: 4096   regular empty
file
Device: fd00h/64768d         Inode: 2261179     Links: 1
Access: (0664/-rw-rw-r--) Uid: (500/sean)    Gid: (500/sean)
Access: 2015-02-03 21:47:46.000000000 -0600
Modify: 2015-02-03 21:47:46.000000000 -0600
Change: 2015-02-03 21:47:46.000000000 -0600
 Birth: -
```

Now move the file to a new name with the **mv** command, as shown in Example 5-3.

Example 5-3 Moving Files to a New Name

```
$ mv file1 file2
$ stat file2
  File: `file2'
  Size: 0         Blocks: 0          IO Block: 4096   regular empty
file
Device: fd00h/64768d         Inode: 2261179     Links: 1
Access: (0664/-rw-rw-r--) Uid: (500/sean)    Gid: (500/sean)
Access: 2015-02-03 21:47:46.000000000 -0600
Modify: 2015-02-03 21:47:46.000000000 -0600
Change: 2015-02-03 21:48:41.000000000 -0600
 Birth: -
```

Because the device and inode stayed the same you know this is the same file as before. The change time was modified to reflect the fact that the file was renamed.

When you move a file, the **mv** command overwrites the destination if it exists. This command supports an option, **-i**, that first checks the target to see whether it exists. If it does, **mv** asks whether you want to overwrite the target. Some distributions make **-i** a default option with a shell alias. Chapter 11, “Customizing Shell Environments,” discusses shell aliases in more detail.

Another quirk of the command is the lack of an **-r**, or recursive, option. This is because when you move a directory or a file you’re just changing the directory entry for the file. The directory continues to point to the same files so there is no need to move the files themselves.

You can avoid the overwriting of newer target files or directories with the **-u** option, preserving the latest copy of an object.

Examples of moving files and directories include moving a single directory to another directory name, as shown here:

```
mv -f dir1 dir2
```

This merely changes the directory entry **dir1** to the new name **dir2**. It also removes the "are-you-sure" prompt with the **-f** option.

Just like the **cp** command, moving directory contents requires a correctly formed command; otherwise, you'll move a directory not to the new name, but to a subdirectory of the existing directory.

For example, consider the **/test** directory again, with its structure similar to the following:

```
$ tree .
|-- dir1
|   |-- file1
|   `-- subdir1
`-- dir2
```

If you were a Windows administrator, it would make sense to run the following command to move **dir1** to **dir2**:

```
mv dir1 dir2
```

If you do this on a Linux system and then run the **tree** command, you see the following output:

```
$ tree .
`-- dir2
    `-- dir1
        |-- file1
        `-- subdir1
```

This moves **dir1** under **dir2** because **dir2** already existed. To properly move the contents of the source **dir1** to the target **dir2**, you don't need to use the nonexistent **-r** option (exam trick). You can just use a forward slash and an asterisk to refer to the files underneath **dir1**, like this:

```
mv dir1/* dir2
```

NOTE The * wildcard operator won't match hidden files because they begin with a period. Handling this case is actually quite complicated and outside the scope of the exam.

If you run the **tree** command, you see the following output:

```
$ tree .
|-- dir1
`-- dir2
    |-- file1
    `-- subdir1
```

Finally, the directories you pass to the **mv** command don't always have to be underneath your current directory. You can use absolute pathnames, such as **mv /dir1 .** to move **dir1**, which is off the root directory into the current directory. You can also run **mv /dir1 /tmp** from anywhere in the system to move that same directory into the temporary directory.

Transforming Data Formats

The **dd** command is useful for a variety of tasks, not the least of which is creating backup images, called ISO files, of CD or DVDs. The two main formats **dd** interacts with are the raw device file and the full path of a file or object on the system.

For example, when creating a new boot disk, the **.img** binary file is read block by block from the CD-ROM (as a file) and written to a USB disk raw device as a set of blocks:

```
dd if=/mnt/cdrom/images/boot.img of=/dev/sdb
```

Creating an image of a CD-ROM involves reading the raw USB device block by block and creating a file on the filesystem that contains all those blocks:

```
dd if=/dev/sdb of=/root/usb.img
```

To duplicate a USB device named sdb to another USB device named sdc, the command is

```
dd if=/dev/sdc of=/dev/sdc
```

The **if** keyword means input file and the **of** keyword means output file. The exact order is unimportant, but as you can imagine, mixing up the in and out files can cause you to do terrible things such as overwriting parts of your hard drive!

dd, unlike most other Unix utilities, does not use dashes for its options. Options are specified in the format of **option=value**.

The **dd** command is also often used to duplicate a drive or partition of a drive to another like object.

For example, to copy the first partition from the /dev/sda disk to the same location on the second hard drive on the system, you would use the following command:

```
dd if=/dev/sda1 of=/dev/sdb1
```

You can also copy an entire disk device to another on the system by leaving off the partition numbers:

```
dd if=/dev/sda of=/dev/sdb
```

This works only if the second device is as large as or larger than the first; otherwise, you get truncated and worthless partitions on the second one.

Backing up the MBR is another trick that **dd** does well. Remember that the master boot record contains the indexes of all the partitions on that drive, and thus is very important. To create a disk file that contains only the first 512 bytes of the first hard drive in the system, use this command:

```
dd if=/dev/sda of=/root/MBR.img count=1 bs=512
```

The **count** keyword sets the number of reads from the input file you want to retrieve, and the **bs** keyword sets the block size.

If you don't set the count and block size on this command to back up the MBR, you'll be copying the entire device's blocks to the filesystem—a snake-eating-its-own-tail operation that is guaranteed to fill up the partition quickly and crash the system.

The restoration procedure is just the opposite:

```
dd if=/root/MBR.img of=/dev/sda count=1 bs=512
```

Creating and Removing Directories

A basic task of file management is to be able to create and remove directories, sometimes creating or removing whole trees at once. To create a directory named **dir1**, you use **mkdir dir1**. To create a directory named **subdir1** in the **dir1** directory, you use **mkdir dir1/subdir1**.

Always think of the last segment of any directory path as the object being created or removed, and think of the rest as supporting or parent objects. The **mkdir** and **rmdir** commands are similar in features and options, including the capability of **mkdir** to create a deep subdirectory tree from scratch in a single command:

```
mkdir –p /dir1/dir2/dir3/dir4
```

One of the quirks about the **rmdir** command is that it cannot remove anything but an empty directory. For example, the last directory of the chain **/dir1/dir2/dir3/dir4** is the real target for this command, and only if that directory is empty (no regular or directory files) can it be removed.

```
rmdir –p /dir1/dir2/dir3/dir4
```

One option to the **rmdir** command does allow it to remove directories that have files and so on in them. It's called **--ignore-fail-on-non-empty** and is the longest option I know of in Linux. I'd rather type **rm –rf targetdir** 20 times than this beast.

Removing Objects

It follows that you'll want to remove objects after creating or copying them, and this is done with the **rm** command for most objects. **rmdir** can also be used.

Deleting files with the **rm** command is a matter of choosing the target to be removed and the options that work best.

If you want to remove a particular file and never be prompted by confirmation messages, the command is **rm –f target**.

To remove a directory and all its contents, and never get a confirmation message, the command is **rm –rf /full/path/to/target**.

Where Are Those Files?

Having a mechanism for finding or locating files on a Linux system is essential because the sheer amount of directories and files makes searching manually nearly impossible.

There are two methods for accomplishing this task—quick and dirty or slow and methodical. Most people try the quick **locate** command before resorting to the plodding **find** command.

Locating Files with Locate

The quickest way to find a file or set of files is to use the **locate** command. It's fast, database-driven, and secure. When you run the **locate** command you are searching a database instead of the filesystem, and only files that you have access to are shown. The downside of the database is that it's updated nightly and is therefore unaware of any changes that have happened since the last update.

locate has a quirky way of showing results. You would probably expect that using **locate** for a file named **readme** would locate only files named **readme**, but that's not quite true. It finds anything that has a filename of **readme**, including regular files and any part of the path.

For example, while attempting to locate the **readme** file, you run the following command:

```
locate readme
```

This finds both of the following entries, one with the string **readme** as a part of the filename and the other a directory:

```
/readme
/usr/src/linux-2.4.20-8/drivers/net/wan/8253x/readme.txt
```

Key Topic

Use the **locate** command to find items you know are on the disk, or that you know existed before the last **locate** database update. The database that **locate** uses is updated nightly when the system runs its maintenance routines, or on demand. If you don't have permissions to the object, it isn't shown in the **locate** output.

Use **locate** with the **-i** option to ignore the case (upper or lower) and return anything that matches your search string using a case-insensitive match:

```
locate -i string
```

The **locate** database needs to be updated regularly to ensure good results. Your distribution probably puts it in the list of nightly jobs to be run. For more details on the nightly jobs, see Chapter 16, "Schedule and Automate Tasks." Updating the database can take a long time, and it is frustrating having to wait for the updates to finish when you need to search.

The **update** commands must be run as **root**, and either one will do the job:

```
updatedb
```

Sometimes you want to exclude files or directories from the **locate** database because they either are inappropriate or simply take too long to index without any apparent benefit. This is configurable in the **/etc/updatedb.conf** file. This file is read and the variables are used by the updating commands.

The two main methods of excluding objects in the configuration file are either by filesystem type or path. The following output is an example of a working **/etc/updatedb.conf** file:

```
PRUNEFS="devpts NFS nfs afs sfs proc smbfs autofs auto iso9660"
PRUNEPATHS="/tmp /usr/tmp /var/tmp /afs /net /sfs"
export PRUNEFS
export PRUNEPATHS
```

The **PRUNEFS** keyword is for filesystem types you want excluded from the **locate** database update; as you might expect, the **PRUNEPATHS** keyword is for directory trees you want excluded. Notice that most of the paths are temporary data locations or exotic file locations.

Key Topic

Remember for the exam that **locate** returns results for the search string in any portion of the path or filename it finds the string in. There will be questions that **locate** is right for, and some that really want the **whereis** command.

Finding Files

The **find** command is the most accurate but time-consuming method for searching the system for file objects because it crawls the list of files in real time versus the **locate** indexed database. The command consists of several (sometimes confusing) sections. But, if it's learned properly, it can be a powerhouse for the busy sysadmin.

The structure of a **find** command is

```
find startpath -options arguments
```

To make sense of this jumble of sections, let's take a look at a useful **find** command and match up the sections:

```
# find /home -iname *.mp3
/home/snuffy/g3 - red house.mp3
```

The previous command sets the start path to the **/home** directory and then looks for any instance of the string **mp3** as a file extension, or after the last **.** in the filename. It finds a file in the user **snuffy**'s home directory and returns the full path for that file.

Options for **find** include

- **group—**Based on files belonging to the specified group
- **newer—**Based on files more recent than the specified file
- **name—**Based on files with names matching a case-sensitive string
- **iname—**Based on files with names matching a non-case-sensitive string
- **user—**Searches for files belonging to the specified user
- **mtime—**The modify time; used for finding files x days old
- **atime—**Based on the number of days since last accessed
- **ctime—**Based on the number of days since the directory entry was last changed

A useful feature of the **find** command is its capability to execute another command or script on each and every entry normally returned to standard output.

For example, to find all MP3 files in the user's home directories and archive a copy into the root user's home directory, you could use this command:

```
find /home -iname *.mp3 -exec cp -f {} .\;
```

This command uses the **-exec** option, which accepts every line returned to standard output one by one and inserts the full path and filename between the curly brackets (**{}**). When each line of output is parsed and the command is executed, it reaches the **\;** at the end of the line and goes back to standard input for the next line. The last line of output is the last one with a command executed on it; it doesn't just keep going and error out.

Running multiple operators in a single command is possible, too. Just be sure not to get the values for one operator mixed up in the next. You could look for all MP3 files owned by a given user with the following command:

```
find /home -iname *.mp3 -user snuffy
/home/snuffy/bls - all for you.mp3
```

The **find** command is complex, and rather than bore you with more possible options, I've worked out a number of examples of how to use **find**:

Key Topic

To find a file and execute **cat** on it, use

```
find /etc -iname fstab -exec cat {} \;
```

To delete all **core** files older than seven days, use the following:

```
find /home -mtime +7 -iname core -exec rm -f {} \;
```

To find all files on the system owned by **bob** and change the ownership to **root**, use

```
find / -user bob -exec chown root {} \;
```

To find all files by user tjordan and change his group, use this command:

```
find /data -user tjordan -exec chGRP users {} \;
```

For safety you can use **-ok** instead of **-exec** to be prompted for confirmation each time the command runs.

```
find /data -user tjordan -ok chgrp users {} \;
```

To find all inodes related to a hard link, use the command find **/ -inum 123456**.

The **find** command's operators and the capability to execute commands on the search results will be covered on the exam. Practice all the examples you see here and get inventive with the possibilities. Particularly watch out for the use of **-mtime** and its cousins: **-atime** and **-ctime**.

Which Command Will Run?

With the plethora of commands and executable scripts offered on a Linux machine, you need to know which of the possible commands will run when you type the name of it on the command line. This all depends on the contents of the **PATH** variable. This variable's contents are used as a sequentially read set of locations to search for executable objects.

The **which** command is used to determine the full path of commands that are queried from the **PATH** variable. To determine which command is indeed executed just by typing the name, run the following command:

```
which ls
alias ls='ls --color=tty'
        /bin/ls
```

As you can see, two entries were found that contain the **ls** command. The first is an alias, one that sets some color functions to the **ls** command; the other is the real command binary in **/bin/ls**.

When you execute a command, it finds the first available match, which might not be the one you wanted, as is the case with the **ls** command. To make it execute a physical binary and ignore any aliases that have been set, preface the command with a backslash (\), like so:

```
\ls
```

Try it again on a command that has two executables on the system, the **gawk** command:

```
which gawk
/bin/gawk
```

This returns a single entry, but there are multiple **gawk** commands on a Linux box. The first matching command found is returned by default, and only if you use the proper switch does it find all possibilities:

```
which -a gawk
/bin/gawk
/usr/bin/gawk
```

Researching a Command

When you need more information about a command than just which one will execute, try **whereis**. This command shows up to three possible bits of information, including its binary files, the man page path, and any source files that exist for it. Here's its syntax:

```
$ whereis ls
ls: /bin/ls /usr/man/man1/ls.1.gz
```

Options for **whereis** include

- **-b—**Searches for binaries
- **-m—**Searches for manual entries
- **-s—**Searches for sources
- **-u—**Finds unusual or improperly documented entries

Key Topic

To find a file by name but not get all the entries that contain the name in the path, use the **whereis** command—not the **locate** command—because it finds the string in all elements of the path.

In Chapter 11, Customizing Shell Environments, you will learn how to extend the shell to make common tasks even easier. The **type** command will tell you if a command has been extended. To check what happens when you type **ps**:

```
$ type ps
ps is /bin/ps
```

The output of the **type** command above indicates that the **/bin/ps** application will be run if you type **ps**.

The **ls** command is often extended to show common options, such as to add color to the output:

```
$ type ls
ls is aliased to `ls --color=auto'
```

The output above shows that when you run **ls**, you actually get **ls --color=auto**. You can see all the possible variants of **ls** by using **type**'s **-a** option:

```
$ type -a ls
ls is aliased to `ls --color=auto'
ls is /bin/ls
```

The **-a** option shows that the shell knows about both an alias and a file on disk.

Linking Files

Links come in two varieties: symbolic and hard. (Symbolic links are often known as soft links.) Each has its own set of advantages and disadvantages. Sysadmins use links for a multitude of purposes; chief among them is the need to make shortcuts on the system for users to access data without having to navigate multiple directory levels.

If you have users on your Linux systems, you need to have a single mount point accessible to multiple users. The options include having users navigate to the **/mnt/somemount** directory to save data or putting a link to that mount point in their home directories. You're much better off using a link for this task.

Symbolic Links

Symbolic links are used primarily to make a shortcut from one object to another. A symbolic link creates a tiny file with its own inode and a path to the linked file. Symlinks can span across filesystems and drives, primarily because a symlink has its own inode. Figure 5-1 shows the relationship between a symlink and the target file.

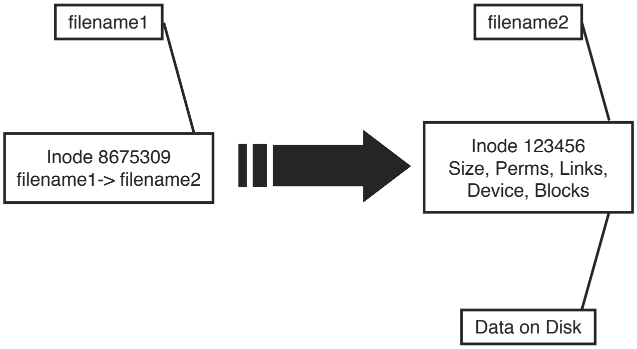

Figure 5-1 Symbolic link detail

For example, you might mount an external disk on the **/mnt/projdata** mount point and want each user to be able to access that remote share from her own home directory. You simply have to issue the following command in each user's home directory to accomplish this:

```
ln -s /mnt/projdata projdata
ls -l projdata
lrwxrwxrwx    1 root   root    13 Jan 26 12:09 projdata -> /mnt/
projdata
```

Notice that the listing for the new symlink shows exactly where the link points, and the permissions are set to the maximum so as to not interfere with the permissions on the target object.

Symbolic links always look like they have the same permissions, but don't try to change them. Changing permissions on a symlink changes the permissions on the target permissions instead.

Hard Links

A *hard link* is normally used to make a file appear in another place. A hard link is simply an additional name in a directory that points to the exact same inode and shares every aspect of the original file except the actual name (although the filename could be identical if in a different directory). Figure 5-2 shows the relationship between a hard link and the target file.

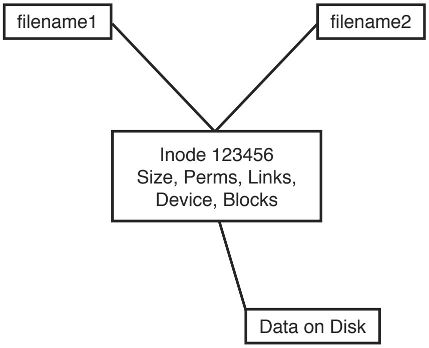

Figure 5-2 Hard link detail

For an example of using a hard link, consider the need to ensure that a frequently deleted file is easily restorable for a given user. The user, Jaime, travels a lot, but when he's in the office he seems to delete things a lot or claims the system has eaten his files. When Jaime is flying, you don't have any issues, so the problem must be the user's actions.

To anchor or back up an important file such as the company contact list in Jaime's home directory, you first must create a backup directory, something like **/backup**.

Then, you create a hard link from Jaime's **ccontactlist.txt** file to a file in the **/ backup** directory, like so:

```
cd ~jaime
ln ccontactlist.txt /backup/home_jaime_ccontactlist.txt
ls -l ccontactlist.txt
-rw-r--r--    2 jaime    users    0 Jan 26 13:08 ccontactlist.txt
```

Notice that the file appears normal, but the number **2** for the link count lets you know that another name entry for this file exists somewhere.

Also notice that the listing for the new hard link doesn't show the target file or seem to refer to it in any way. Running the **stat** command on this file won't show you the other filename or seem to be aware of it outside the higher link count.

Key Topic

The name and location of a file are the only things about the file not stored in the inode. This appears on the exam in questions for this set of objectives.

Hard links can't be created if the target is on another filesystem, disk, or remote object. The need to associate multiple names to the same inode makes this impossible.

Be careful when changing the permissions and ownership on the hard-linked files because all name entries point to exactly the same inode. Thus, any changes are instantly made to what would appear to be multiple files but what, in reality, are only filenames.

To delete a file that has multiple hard links requires the removal of every hard link or the multiple names. To find all the links for a file, run the following command:

```
ls -i ccontactlist.txt
17392 ccontactlist.txt
find / -inum 17392
/home/jaime/ccontactlist.txt
/backup/home_jaime_ccontactlist.txt
```

Key Topic

NOTE On the exam, remember that a symlink is another actual file with its own inode. A large number of symlinks can therefore cause a problem for a filesystem, such as one that contains users' home directories. Too many inodes used can restrict you from using the storage space available. Run the **df –i** command to see what the statistics are.

Backup Commands

As an administrator you often are called upon to deal with file archives, which are one or more files that have been packaged into one file and optionally compressed.

There are several uses for archives:

- You want to send a few files to someone or copy them to another server and want to package and compress them.
- You need to back up a partition to other media in case a disk fails or the file is otherwise lost.

- You want to make a temporary copy of something before you make a change so you can restore it quickly if needed.
- You need to keep files around but in compressed format, such as for archiving old logs.

A number of backup options are available for Linux systems. Some are more useful than others, and some act on files, whereas others work best on partitions or disks as a unit.

Backup commands on the exams include the following:

- **cpio**
- **tar**
- **gzip** and **gunzip**
- **bzip2** and **bunzip2**
- **xz**

Using tar

The **tar** command is the workhorse of the archival world. The name comes from the term tape archive and goes back to the time when most backup was done to a local tape drive. You can think of **tar** as a pipeline that takes in a series of files and outputs a single file that is meant to be streamed to tape, but this output could be sent to a file on disk as well.

On the way through the pipeline you can do some transformations on the files such as chop up the output onto something that fits across multiple tapes, exclude files that weren't recently changed, or rewrite the directory names stored in the archive.

tar also provides the extraction options. You take a **.tar** file, also called a *tarball*, and run it through **tar** to get back a copy of the original files. It is possible to extract only certain files and manipulate the filenames.

The **tar** command also can use various compression commands, particularly the **gzip/gunzip** and **bzip2/bunzip2** commands by the use of special option characters. This has the effect of creating a compressed archive file, typically named **.tar.gz** for **gzip**-compressed files and **.tar.bz2** for **bzip2**-compressed files.

tar commands have an unusual syntax. The command is **tar**, followed by a dash (**-**), and then all the options concatenated together such as **xvjf**. After this is a list of zero or more filenames; the meanings depend on the options you chose.

The **tar** command has three main methods that act on files or **tar** archives; each has a corresponding letter that must be the first letter in the list of options:

- **c**—Creates an archive
- **t**—Tells you the contents of an archive
- **x**—Extracts files from an archive

The rest of the command can be optional, but some typical options are

- **v**—Be verbose by giving a list of files as they are processed.
- j or **z**—Compress or decompress with **bzip2** or **gzip**, respectively.
- **f**—The next word is the name of the file to operate on.

Figure 5-3 shows your choices graphically. We look at examples of each.

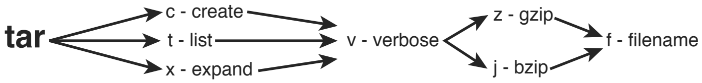

Figure 5-3 Picturing the tar options

When you're creating an archive with **tar**, you should think about what you want to archive, where you want the resulting archive to be created, and what compression if any you want to use.

To create a simple **tar** archive, the options you need are as follows:

```
tar -cf archive.tar /foo
```

In this example, the **-c** option signals **tar** to create the file specified after the **-f** option and specifies the directory you are archiving, which is the **/foo** directory. Note that you have to add the .tar suffix. By default the operation is recursive.

To create the same archive with **gzip** compression, you simply insert a **-z** option and use the letters .gz as the filename suffix:

```
tar -czf archive.tar.gz /foo
```

This creates a compressed archive file that uses the **gzip** compression algorithms. If you want slightly higher compression, use the **-j** option (instead of the **-z** option) for **bzip2** compression and create your archive with a suffix of .bz or.bz2.

You will likely see questions on the exam that test your knowledge of which compression command has the highest compression. For example, using **bzip2** generally results in a smaller archive file at the expense of more CPU cycles to compress and uncompress. The **gzip** package is almost always part of the default installation of Linux while **bzip2** may not be.

To create a **tar** archive and see the filenames as they are processed use the **-v** option:

```
tar -cvf archive.tar /foo
```

This produces the following output:

```
tar: Removing leading `/' from member names
foo/
foo/install.log
foo/install.log.syslog
foo/.bash_logout
```

If given an absolute directory name to archive, **tar** strips the leading **/** from the full path of the objects in the archive. It would not be good if you could overwrite files in your **/usr** directory by extracting a file in an unrelated directory!

You may pass more than one directory or file to **tar**. For example, **tar –cf foo.tar bin var** creates an archive called **foo.tar** containing both the **bin** and **var** directories.

Taking Pity on the Unarchiver

It's considered proper and elegant to create **tar** archives by specifying a directory that contains the files to be archived, not just a bunch of files that are in the current directory. This means that when the files are untarred they show up in a single directory instead of in the current directory.

For example, create an archive of the **/etc** directory contents with the following command:

```
tar -cf etc.tar /etc
```

When you unarchive the **tar** file, by default it creates an **etc** directory in the current directory, which contains the entirety of the **/etc** directory you archived.

Contrast this with the nightmare that happens when you navigate to the **/etc** directory and create the archive from there with this command:

```
tar -cf /root/badetc.tar *
```

This archive file contains the same files as the previous one, except they aren't contained in a top-level **etc** directory—everything is in the top level of the archive.

Imagine what will happen to your system when you unarchive this file in the root user's home directory. You will have spewed approximately 2,400 files directly into the root user's home directory!

It really does matter where you are in the filesystem and which path options you use when you create or expand an archive file. It's best practice to use absolute pathnames.

To solve the problem of 2,400 files polluting your root user's home directory, use the following command, where **badetc.tar** is the offending archive file:

tar -tf badetc.tar | xargs rm -rf

This command produces a list of the paths and filenames of files in the archive and uses the **xargs** command to feed each line of output as a filename specification to the **rm -rf** command, removing all the files and directories expanded from the **badetc.tar** file.

Useful Creation Options

A number of other options can be used for creating **tar** archives. Here is a list of the more useful and testable ones:

- **-b**—Sets the block size to fit the media to which you are archiving. This is necessary for some tape devices.
- **-M**—This specifies multiple archive targets or spreads a large archive across multiple tapes or media.
- **-g**—Creates a new format incremental backup (only those that have changed since the last full or incremental).
- **-l**—Stays on the local filesystem; it's used to keep from backing up the entire NFS network by accident.
- **-L**—This is followed by a number that reflects 1024 bytes, so **-L 500** equals 500KB. (It's used for setting the tape length so multiple tapes can be used for an archive.)
- **--remove-files**—This is dangerous because the specified files are removed from the filesystem after they have been added to the archive!

Listing Archive Files

An underrated option, listing is something that typically is used after you don't get the results you want or realize what you've just done and want to confirm how hard it is going to be to clean up.

To tell you the contents of a **tar** archive, use the following command:

```
tar -tf archive.tar
```

This produces the output shown here:

```
etc/
etc/sysconfig/
etc/sysconfig/network-scripts/
etc/sysconfig/network-scripts/ifup-aliases
etc/sysconfig/network-scripts/ifcfg-lo
```

To list an archive that uses compression, simply insert the necessary letter between the **-t** and the **-f** options, such as the **bzip2 -j** option shown here:

```
tar -tjf archive.tar.bz2
```

This produces the following output:

```
etc/
etc/sysconfig/
etc/sysconfig/network-scripts/
etc/sysconfig/network-scripts/ifup-aliases
etc/sysconfig/network-scripts/ifcfg-lo
```

To list an archive and see the file details for its contents, you add the **-v** option to the existing command to see an output of the details:

```
tar -tvjf archive.tar.bz2
```

This returns output similar to the following:

```
drwxr-xr-x root/root         0 2015-02-10 03:46 etc/
drwxr-xr-x root/root         0 2015-01-31 10:09 etc/sysconfig/
drwxr-xr-x root/root         0 2014-11-10 22:13 etc/sysconfig/network-
scripts/
```

Key Topic

When you create an archive with the **-v** option, a list of the files being archived is shown onscreen. When you unarchive an archive with the **-v** option, it shows a similar list of the files being unarchived.

It's only when you list an archive with the **-v** option that you get the type of output that approximates an **ls -l** command being run on the archive contents. This is an exam topic, so be ready for it.

Using cpio

The **cpio** command appears extensively in the Level 2 LPI objectives. This level of the exam might ask you about the **cpio** command at only the simplest levels, such as knowing that it exists, how it works in general terms, and whether it can be used to back up a Linux system.

The **cpio** command actions all treat the filesystem as the home base. If you are copying out, it's from the filesystem out to another file. The same is true with copying in—it's from a file into the filesystem.

The **cpio** command has three options for acting on files and filesystems:

- **-o** or **--create**—This copies files to an archive using a list of files typically created by the **find** command.
- **-i** or **--extract**—This copies files into the filesystem from an archive or a list of the archive contents.
- **-p** or **--pass-through**—This copies files from one directory tree to another without the use of an archive, essentially performing the same function as the **cp -r** command.

Key Topic

The **cpio** command accepts a list of files in a one-file-per-line format and uses this list to send the archived files to either the standard output or an archive file you specify.

cpio supports a variety of archive formats, including binary, ASCII, crc, and tar, to name the most relevant.

An example of creating a cpio archive from the files in the current directory is shown here:

```
find . "*" | cpio -o >  archive.cpio
```

This outputs the list of files found by this particular **find** command, with the **cpio** command taking the entirety of the files and sending them to the **archive.cpio** file by redirecting standard output to the file.

The **cpio** command doesn't accept a list of files to archive on the command line like the other utilities you've seen so far. Instead, it reads the names of the files from the standard input or console. So be aware that using either the **find** or **ls** command is necessary to feed **cpio** a list of filenames.

For example, if you needed to archive all the files that have an extension of .txt in the current directory to a **cpio** archive named txt.cpio, you would use the following command:

```
ls *.txt | cpio -o > txt.cpio
```

Notice that you're redirecting the output of cpio to a file rather than letting it write the file itself. Therefore the filename is up to you, and if you want a **cpio** file extension, you need to add it yourself.

Compression Utilities

Whereas the **tar** command is used to gather files and put them in a container, the **gzip**, and **bzip2** commands are used to compress that container. Used by themselves, they act on each file they find and replace that file with a compressed version that has an extension that indicates the file is compressed.

The **gzip** and **bzip2** compression utilities compress files and are similar in their functions and operations. The main difference is that **bzip2** offers slightly better compression than **gzip**, but **gzip** is much more widely used.

These commands replace the original file with a new file that has an additional extension, so don't delete the .gz or .bz2 files that you create. They are the original files in a compressed wrapper!

To compress all the files in the current directory with **gzip** or **bzip2**, use this command:

```
gzip *
```

This replaces all the regular files (not the directories or their contents) in the current directory with the original filenames plus a **.gz** extension. So, if you had two files named file1 and file2 in the directory, they would be replaced with

```
file1.gz
file2.gz
```

To uncompress these files, just do the exact opposite of the compression:

```
gunzip *
```

This restores the original files.

Using **bzip2** produces the same sort of results. You can issue the following command in the same directory:

```
bzip2 *
```

You would then have the following two files:

```
file1.bz2
file2.bz2
```

To uncompress these files, issue this command:

```
bunzip2 *
```

This restores the files to their original states.

xz is a third option for compressing files just like **bzip2** and **gzip**. It is newer, and in some cases has better performance than **bzip2** at a cost of more memory. Files are compressed with one of **xz**, **xz -z**, or **xz --compress**, and decompressed with one of **unxz**, **xz -d**, **xz --uncompress**, or **xz --decompress**.

The .xz file extension indicates that a file was compressed with **xz**. To uncompress **foo.xz** you would run **xz -d foo.xz**, and would be left with an uncompressed file called **foo**.

Watch for questions that ask about why you would use either **gzip** or **bzip2** for a particular compression task. **bzip2** offers slightly better compression at the expense of increased CPU cycles. **gzip2** is faster but doesn't compress as well. **gzip2** also has a recursive option (**-r**) that compresses all files in a directory.

Summary

In this chapter you learned about the Linux File System Hierarchy Standard (FHS) and what it means for laying out partitions. You also learned how to find files in real time with the **find** command, and through a database lookup with the **locate** command. This chapter also covered the **cp**, **mv**, and **touch** commands for copying, moving, and updating files, along with the proper use of file globs for matching files on the command line.

Finally you learned about the various archival and compression utilities that Linux makes available to you.

Exam Preparation Tasks

As mentioned in the section "How to Use This Book" in the Introduction, you have a couple of choices for exam preparation: the exercises here, Chapter 21, "Final Preparation," and the practice exams on the DVD.

Review All Key Topics

Review the most important topics in this chapter, noted with the Key Topics icon in the outer margin of the page. Table 5-3 lists a reference of these key topics and the page numbers on which each is found.

Key Topic

Table 5-3 Key Topics for Chapter 5

Key Topic Element	Description	Page Number
Paragraph	FHS documentation about what goes on the root volume	113
Paragraph	The use of the /usr and /usr/local/ directories	114
Paragraph	Relative pathnames and . (period character)	116
Paragraph	Long listing format (**-l**) to see permissions	117
Paragraph	Using the **touch** command	120
Paragraph	Using a glob to avoid copying into a directory incorrectly	122
Paragraph	**Locate** needs the database refreshed periodically	129
Paragraph	**Locate** searches whole names	130
Paragraph	Examples of **find** usage	131
Paragraph	When to use **whereis** versus **locate**	133
Paragraph	The filename is not stored in the inode	136
Note	Symlinks consume inodes	136
Paragraph	The order and function of **tar**'s options	138
Paragraph	Creating a **tar** archive	138
Paragraph	**bzip2** has the highest compression rate	138
Paragraph	The **-v** option to tar	141
Paragraph	**cpio** accepts its files from the standard input	142

Define Key Terms

Define the following key terms from this chapter and check your answers in the glossary:

File System Hierarchy Standard, relative path, absolute path, hard link

Review Questions

The answers to these review questions are in Appendix A.

1. You are installing a customized server and need to strip the root filesystem down to the essentials only. According to the FHS 2.3, which of the following are considered optional on the root (/) filesystem? (Choose two.)

   a. /root

   b. /usr

   c. /tmp

   d. /home

2. One of your programmers has produced an order entry system that will be shared among your users from a central file server. What is the appropriate directory to place this program and its associated files in?

   a. /usr/local/bin

   b. /usr/local

   c. /usr/share

   d. /opt

3. Which of the following is a true statement about files on a default Linux system? (Choose all that apply.)

   a. Filenames can start with a number.

   b. Filenames can contain multiple periods.

   c. Filenames can contain spaces.

   d. Filenames can contain ampersands.

   e. Filenames can contain backslashes.

4. You find a string in a shell script that contains the following command:

   ```
   cp /data/*.doc ~tarfoo
   ```

   What is the meaning of the characters ~tarfoo?

   a. A special function named tarfoo

   b. A directory named tarfoo in your home directory

   c. The tarfoo user's home directory

   d. The /data/tarfoo directory

5. You are currently in the directory /home1/user1/subdir1 and need to navigate to the directory /home12/user3. Which of the following commands will accomplish this?

   a. **cd home12/user3**

   b. **cd ~/user3**

   c. **cd ../../home12/user3**

   d. **cd ../../../home12/user3**

6. You have a directory named /dir1 that contains subdirectories and regular files. You want to replicate this directory structure exactly into an existing directory named /dir2. Which of the following commands accomplish this? (Choose all that apply.)

   a. **cp –-contents dir1/ /dir2**

   b. **cp –r /dir1/* /dir2**

   c. **xcopy /dir1 /dir2**

   d. **cp –r /dir1 /dir2**

7. You are currently in the /bbb directory and want to move the contents from the /ccc directory to this one. What is the shortest command that will accomplish this?

   a. **mv /ccc/*.* .**

   b. **mv ../ccc/*.* .**

   c. **mv /ccc/* .**

   d. **mv /ccc/ /bbb**

8. Which option to the **mkdir** and **rmdir** commands allows you to create a nested subdirectory tree?

   Example:

   ```
   /dir1/dir2/dir3/dir4
   ```

   a. **-c**

   b. **-n**

   c. **-d**

   d. **-p**

9. You are the sysadmin of a busy server and need to save space on your /home partition. You need to remove all files named **core** that are older than seven days in the users' home directories, without receiving any prompts.

    a. **find /home –mtime +7 –name core –exec rm –f {} \;**

    b. **find ~ -mtime +7 -name core -exec rm -f {} \;**

    c. **find /home -mtime -7 -name core -exec rm -f {} \;**

    d. **find /home -older 7d -name core -exec rm -f {} \;**

10. Which of the following situations would prevent you from creating a hard link?

    a. The link spans filesystems.

    b. The source of the link is a hidden file.

    c. The source of the link is a device file.

    d. The source of the link is a directory.

    e. The destination contains special characters.

11. How would you back up Rebecca's home directory using the best compression available?

    a. **cd /home; tar -czf rebecca.tgz rebecca**

    b. **find ~rebecca | tar -cjf - > rebecca.tar.bz2**

    c. **tar -cjf rebecca.tar.bz2 ~rebecca**

    d. **tar -xjf rebecca.tar.bz2 ~rebecca**

This page intentionally left blank

Index

Appendix B

Study Planner

Practice Test	Reading	Task

Element	**Task**	**Goal Date**	**First Date Completed**	**Second Date Completed (Optional)**	**Notes**
Introduction	Read Introduction				
1. Installing Linux	Read Foundation Topics				
1. Installing Linux	Review Key Topics				
1. Installing Linux	Define Key Terms				
1. Installing Linux	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 1 in practice test software				
2. Boot Process and Runlevels	Read Foundation Topics				
2. Boot Process and Runlevels	Review Key Topics				
2. Boot Process and Runlevels	Define Key Terms				
2. Boot Process and Runlevels	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 2 in practice test software				
3. Package Install and Management	Read Foundation Topics				
3. Package Install and Management	Review Key Topics				
3. Package Install and Management	Define Key Terms				
3. Package Install and Management	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 3 in practice test software				
4. Basic Command Line Usage	Read Foundation Topics				

4. Basic Command Line Usage	Review Key Topics				
4. Basic Command Line Usage	Define Key Terms				
4. Basic Command Line Usage	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 4 in practice test software				
5. File Management	Read Foundation Topics				
5. File Management	Review Key Topics				
5. File Management	Define Key Terms				
5. File Management	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 5 in practice test software				
6. Text Processing/Advanced Command Line	Read Foundation Topics				
6. Text Processing/Advanced Command Line	Review Key Topics				
6. Text Processing/Advanced Command Line	Define Key Terms				
6. Text Processing/Advanced Command Line	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 6 in practice test software				
7. Process Management	Read Foundation Topics				
7. Process Management	Review Key Topics				
7. Process Management	Define Key Terms				
7. Process Management	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 7 in practice test software				
8. Editing Text	Read Foundation Topics				
8. Editing Text	Review Key Topics				
8. Editing Text	Define Key Terms				
8. Editing Text	Answer Review Questions				

Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 8 in practice test software				
9. Partitions and Filesystems	Read Foundation Topics				
9. Partitions and Filesystems	Review Key Topics				
9. Partitions and Filesystems	Define Key Terms				
9. Partitions and Filesystems	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 9 in practice test software				
10. Permissions	Read Foundation Topics				
10. Permissions	Review Key Topics				
10. Permissions	Define Key Terms				
10. Permissions	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 10 in practice test software				
11. Customizing Shell Environments	Read Foundation Topics				
11. Customizing Shell Environments	Review Key Topics				
11. Customizing Shell Environments	Define Key Terms				
11. Customizing Shell Environments	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 11 in practice test software				
12. Shell Scripting	Read Foundation Topics				
12. Shell Scripting	Review Key Topics				
12. Shell Scripting	Define Key Terms				
12. Shell Scripting	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 12 in practice test software				
13. Basic SQL Management	Read Foundation Topics				
13. Basic SQL Management	Review Key Topics				

13. Basic SQL Management	Define Key Terms				
13. Basic SQL Management	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 13 in practice test software				
14. Configuring User Interfaces and Desktops	Read Foundation Topics				
14. Configuring User Interfaces and Desktops	Review Key Topics				
14. Configuring User Interfaces and Desktops	Define Key Terms				
14. Configuring User Interfaces and Desktops	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 14 in practice test software				
15. Managing Users and Groups	Read Foundation Topics				
15. Managing Users and Groups	Review Key Topics				
15. Managing Users and Groups	Define Key Terms				
15. Managing Users and Groups	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 15 in practice test software				
16. Schedule and Automate Tasks	Read Foundation Topics				
16. Schedule and Automate Tasks	Review Key Topics				
16. Schedule and Automate Tasks	Define Key Terms				
16. Schedule and Automate Tasks	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 16 in practice test software				
17. Configuring Print and Email Services	Read Foundation Topics				
17. Configuring Print and Email Services	Review Key Topics				
17. Configuring Print and Email Services	Define Key Terms				
17. Configuring Print and Email Services	Answer Review Questions				

Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 17 in practice test software				
18. Logging and Time Services	Read Foundation Topics				
18. Logging and Time Services	Review Key Topics				
18. Logging and Time Services	Define Key Terms				
18. Logging and Time Services	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 18 in practice test software				
19. Networking Fundamentals	Read Foundation Topics				
19. Networking Fundamentals	Review Key Topics				
19. Networking Fundamentals	Define Key Terms				
19. Networking Fundamentals	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 19 in practice test software				
20. Topic 110: Security	Read Foundation Topics				
20. Topic 110: Security	Review Key Topics				
20. Topic 110: Security	Define Key Terms				
20. Topic 110: Security	Answer Review Questions				
Practice Test	Take practice test in study mode using Exam Bank 1 questions for Chapter 20 in practice test software				
21. Final Preparation	Review Exam Essentials for each chapter on the PDF from the DVD				
21. Final Preparation	Review all Key Topics in all chapters				
21. Final Preparation	Take practice test in practice exam mode using Exam Bank #1 questions for all chapters				
21. Final Preparation	Take practice test in practice exam mode using Exam Bank #2 questions for all chapters				