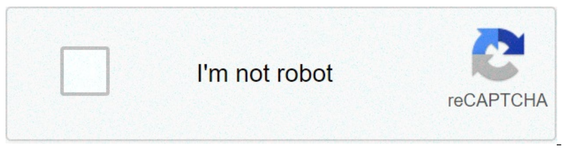

Continue

A practical guide to linux commands pdf

File loading please wait... Praise for the First Edition of A Practical Guide to Linux Commands, Editors, and Shell Programming ® "This book is a very useful tool for anyone who wants to 'look under the hood' so to speak, and really start putting the power of Linux to work. What I find particularly frustrating about man pages is that they never include examples. Sobell, on the other hand, outlines very clearly what the command does and then gives several common, easy-tounderstand examples that make it a breeze to start shell programming on one's own. As with Sobell's other works, this is simple, straight-forward, and easy to read. It's a great book and will stay on the shelf at easy arm's reach for a long time." —Ray Bartlett Travel Writer "Overall I found this book to be quite excellent, and it has earned a spot on the very front of my bookshelf. It covers the real 'guts' of Linux— the command line and its utilities—and does so very well. Its strongest points are the outstanding use of examples, and the Command Reference section. Highly recommended for Linux users of all skill levels. Well done to Mark Sobell and Prentice Hall for this outstanding book!" —Dan Clough Electronics Engineer and Slackware Linux user "Totally unlike most Linux books, this book avoids discussing everything via GUI and jumps right into making the power of the command line your friend." —Bjorn Tipling Software Engineer ask.com "This book is the best distro-agnostic, foundational Linux reference I've ever seen, out of dozens of Linux-related books I've read. Finding this book was a real stroke of luck. If you want to really understand how to get things done at the command line, where the power and flexibility of free UNIX-like OSes really live, this book is among the best tools you'll find toward that end." —Chad Perrin Writer, TechRepublic Praise for Other Books by Mark G. Sobell "I keep searching for books that collect everything you want to know about a subject in one place, and keep getting disappointed. Usually the books leave out some important topic, while others go too deep in some areas and must skim lightly over the others. A Practical Guide to Red Hat® Linux® is one of those rare books that actually pulls it off. Mark G. Sobell has created a single reference for Red Hat Linux that can't be beat! This marvelous text (with a 4-CD set of Linux Fedora Core 2 included) is well worth the price. This is as close to an 'everything you ever needed to know' book that I've seen. It's just that good and rates 5 out of 5." —Ray Lodato Slashdot contributor "Mark Sobell has written a book as approachable as it is authoritative." —Jeffrey Bianchine Advocate, Author, Journalist "Excellent reference book, well suited for the sysadmin of a Linux cluster, or the owner of a PC contemplating installing a recent stable Linux. Don't be put off by the daunting heft of the book. Sobell has strived to be as inclusive as possible, in trying to anticipate your system administration needs." —Wes Boudville Inventor "A Practical Guide to Red Hat® Linux® is a brilliant book. Thank you Mark Sobell." —C. Pozrikidis University of California at San Diego "This book presents the best overview of the Linux operating system that I have found. . . . [It] should be very helpful and understandable no matter what the reader's background: traditional UNIX user, new Linux devotee, or even Windows user. Each topic is presented in a clear, complete fashion, and very few assumptions are made about what the reader knows. . . . The book is extremely useful as a reference, as it contains a 70-page glossary of terms and is very well indexed. It is organized in such a way that the reader can focus on simple tasks without having to wade through more advanced topics until they are ready." —Cam Marshall Marshall Information Service LLC Member of Front Range UNIX Users Group [FRUUG] Boulder, Colorado "Conclusively, this is THE book to get if you are a new Linux user and you just got into the RH/Fedora world. There's no other book that discusses so many different topics and in such depth." —Eugenia Loli-Queru Editor in Chief OSNews.com This page intentionally left blank ® A Practical Guide to Linux Commands, Editors, and Shell Programming SECOND EDITION This page intentionally left blank ® A Practical Guide to Linux Commands, Editors, and Shell Programming SECOND EDITION Mark G. Sobell Upper Saddle River, NJ • Boston • Indianapolis • San Francisco New York • Toronto • Montreal • London • Munich • Paris • Madrid Capetown • Sydney • Tokyo • Singapore • Mexico City Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals. The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein. The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact: U.S. Corporate and Government Sales (800) 382-3419 For sales outside the United States, please contact: International Sales Visit us on the Web: informit.com/ph Library of Congress Cataloging-in-Publication Data Sobell, Mark G. A practical guide to Linux commands, editors, and shell programming / Mark G. Sobell.—2nd ed. p. cm. Includes bibliographical references and index. ISBN 978-0-13-136736-4 (pbk.) 1. Linux. 2. Operating systems (Computers) I. Title. QA76.76.O63S59483 2009 005.4'32—dc22 2009038191 ISBN-13: 978-0-13-136736-4 ISBN-10: 0-13-136736-6 Text printed in the United States at Edwards Brothers in Ann Arbor, Michigan. First printing, October 2009 With love for my guys, Zach, Max, and Sam This page intentionally left blank Brief Contents This page intentionally left blank Preface Linux A Practical Guide to Linux® Commands, Editors, and Shell Programming, Second Edition, explains how to work with the Linux operating system from the command line. The first few chapters of this book quickly bring readers with little computer experience up to speed. The rest of the book is appropriate for more experienced computer users. This book does not describe a particular release or distribution of Linux but rather pertains to all recent versions of Linux. Mac OS X This book also explains how to work with the UNIX/Linux foundation of Mac OS X. It looks "under the hood," past the traditional graphical user interface (GUI) that most people think of as a Macintosh, and explains how to use the powerful command-line interface (CLI) that connects you directly to OS X. As with the Linux releases, this book does not describe a particular release of OS X but rather pertains to all recent releases. Where this book refers to Linux, it implicitly refers to Mac OS X as well and makes note of differences between the two operating systems. Command-line interface (CLI) In the beginning there was the command-line (textual) interface (CLI), which enabled you to give Linux commands from the command line. There was no mouse to point with or icons to drag and drop. Some programs, such as emacs, implemented rudimentary windows using the very minimal graphics available in the ASCII character set. Reverse video helped separate areas of the screen. Linux was born and raised in this environment, so naturally all the original Linux tools were invoked from the command line. The real power of Linux still lies in this environment, which explains why many Linux professionals work exclusively from the command line. Using clear descriptions and lots of examples, this book shows you how to get the most out of your Linux system using the command-line interface. xxxi xxxii Preface Linux distributions A Linux distribution comprises the Linux kernel, utilities, and application programs. Many distributions are available, including Ubuntu, Fedora, Red Hat, Mint, OpenSUSE, Mandriva, CentOS, and Debian. Although the distributions differ from one another in various ways, all of them rely on the Linux kernel, utilities, and applications. This book is based on the code that is common to most distributions. As a consequence you can use it regardless of which distribution you are running. New in this edition This edition includes a wealth of new and updated material: • Coverage of the Mac OS X command-line interface (throughout the book). Part V covers utilities and highlights the differences between utility options used under Linux and those used under Mac OS X. • An all-new chapter on the Perl scripting language (Chapter 11; page 485). • New coverage of the rsync secure copy utility (Chapter 14; page 583). • Coverage of more than 15 new utilities in Part V, including some utilities available under Mac OS X only. • Three indexes to make it easier to find what you are looking for quickly. These indexes indicate where you can locate tables (page numbers followed by the letter t) and definitions (italic page numbers). They also differentiate between light and comprehensive coverage (page numbers in light and standard fonts, respectively). ◆ The File Tree index (page 989) lists, in hierarchical fashion, most files mentioned in this book. These files are also listed in the Main index. ◆ The Utility index (page 991) locates all utilities mentioned in this book. A page number in a light font indicates a brief mention of the utility; use of the regular font indicates more substantial coverage. ◆ The completely revised Main index (page 995) is designed for ease of use. Overlap If you read A Practical Guide to Red Hat ® Linux®: Fedora™ and Red Hat Enterprise Linux, Fourth Edition, or A Practical Guide to Ubuntu Linux®, Second Edition, or a subsequent edition of either book, you will notice some overlap between those books and the one you are reading now. The introduction, the appendix on regular expressions, and the chapters on the utilities (Chapter 3 of this book—not Part V), the filesystem, the Bourne Again Shell (bash), and Perl are very similar in the books. Chapters that appear in this book but not in the other two books include those covering the vim and emacs editors, the TC Shell (tcsh), the AWK and sed languages, the rsync utility, and Part V, which describes 97 of the most useful Linux and Mac OS X utility programs in detail. Audience This book is designed for a wide range of readers. It does not require programming experience, although some experience using a computer is helpful. It is appropriate for the following readers: • Students taking a class in which they use Linux or Mac OS X • Power users who want to explore the power of Linux or Mac OS X from the command line Preface xxxiii • Professionals who use Linux or Mac OS X at work • Beginning Macintosh users who want to know what UNIX/Linux is, why everyone keeps saying it is important, and how to take advantage of it • Experienced Macintosh users who want to know how to take advantage of the power of UNIX/Linux that underlies Mac OS X • UNIX users who want to adapt their UNIX skills to the Linux or Mac OS X environment • System administrators who need a deeper understanding of Linux or Mac OS X and the tools that are available to them, including the bash and Perl scripting languages • Computer science students who are studying the Linux or Mac OS X operating system • Programmers who need to understand the Linux or Mac OS X programming environment • Technical executives who want to get a grounding in Linux or Mac OS X Benefits A Practical Guide to Linux® Commands, Editors, and Shell Programming, Second Edition, gives you an in-depth understanding of how to use Linux and Mac OS X from the command line. Regardless of your background, it offers the knowledge you need to get on with your work: You will come away from this book with an understanding of how to use Linux/OS X, and this text will remain a valuable reference for years to come. A large amount of free software has always been available for Macintosh systems. In addition, the Macintosh shareware community is very active. By introducing the UNIX/Linux aspects of Mac OS X, this book throws open to Macintosh users the vast store of free and low-cost software available for Linux and other UNIX-like systems. In this book, Linux refers to Linux and Mac OS X tip The UNIX operating system is the common ancestor of Linux and Mac OS X. Although the graphical user interfaces (GUIs) of these two operating systems differ significantly, the command-line interfaces (CLIs) are very similar and in many cases identical. This book describes the CLIs of both Linux and Mac OS X.To make it more readable, this book uses the term Linux to refer to both Linux and Mac OS X. It makes explicit note of where the two operating systems differ. Features of This Book This book is organized for ease of use in different situations. For example, you can read it from cover to cover to learn command-line Linux from the ground up. Alternatively, once you are comfortable using Linux, you can use this book as a reference: Look up a topic of interest in the table of contents or index and read about it. Or, refer to one of the utilities covered in Part V, "Command Reference." xxxiv Preface You can also think of this book as a catalog of Linux topics: Flip through the pages until a topic catches your eye. The book also includes many pointers to Web sites where you can obtain additional information: Consider the Internet to be an extension of this book. A Practical Guide to Linux® Commands, Editors, and Shell Programming, Second Edition, offers the following features: • Optional sections allow you to read the book at different levels, returning to more difficult material when you are ready to tackle it. • Caution boxes highlight procedures that can easily go wrong, giving you guidance before you run into trouble. • Tip boxes highlight places in the text where you can save time by doing something differently or when it may be useful or just interesting to have additional information. • Security boxes point out ways you can make a system more secure. • The Supporting Web site at www.sobell.com includes corrections to the book, downloadable examples from the book, pointers to useful Web sites, and answers to even-numbered exercises. • Concepts are illustrated by practical examples found throughout the book. • The many useful URLs (Internet addresses) identify sites where you can obtain software and information. • Chapter summaries review the important points covered in each chapter. • Review exercises are included at the end of each chapter for readers who want to hone their skills. Answers to even-numbered exercises are available at www.sobell.com. • Important GNU tools, including gcc, GNU Configure and Build System, make, gzip, and many others, are described in detail. • Pointers throughout the book provide help in obtaining online documentation from many sources, including the local system and the Internet. • Important command-line utilities that were developed by Apple specifically for Mac OS X are covered in detail, including diskutil, ditto, dscl, GetFileInfo, launchctl, otool, plutil, and SetFile. • Descriptions of Mac OS X extended attributes include file forks, file attributes, attribute flags, and Access Control Lists (ACLs). • Appendix D, "Mac OS X Notes," lists some differences between Mac OS X and Linux. Preface xxxv Contents This section describes the information that each chapter covers and explains how that information can help you take advantage of the power of Linux. You may want to review the table of contents for more detail. • Chapter 1—Welcome to Linux and Mac OS X Presents background information on Linux and OS X. This chapter covers the history of Linux, profiles the OS X Mach kernel, explains how the GNU Project helped Linux get started, and discusses some of Linux's important features that distinguish it from other operating systems. Part I: The Linux and Mac OS X Operating Systems Experienced users may want to skim Part I tip If you have used a UNIX/Linux system before, you may want to skim or skip some or all of the chapters in Part I. All readers should take a look at "Conventions Used in This Book" (page 24), which explains the typographic conventions that this book uses, and "Where to Find Documentation" (page 33), which points you toward both local and remote sources of Linux documentation. Part I introduces Linux and gets you started using it. • Chapter 2—Getting Started Explains the typographic conventions this book uses to make explanations clearer and easier to read. This chapter provides basic information and explains how to log in, change your password, give Linux commands using the shell, and find system documentation. • Chapter 3—The Utilities Explains the command-line interface (CLI) and briefly introduces more than 30 command-line utilities. Working through this chapter gives you a feel for Linux and introduces some of the tools you will use day in and day out. The utilities covered in this chapter include ◆ grep, which searches through files for strings of characters; ◆ unix2dos, which converts Linux text files to Windows format; ◆ tar, which creates archive files that can hold many other files; ◆ bzip2 and gzip, which compress files so that they take up less space on disk and allow you to transfer them over a network more quickly; and ◆ diff, which displays the differences between two text files. xxxvi Preface • Chapter 4—The Filesystem Discusses the Linux hierarchical filesystem, covering files, filenames, pathnames, working with directories, access permissions, and hard and symbolic links. Understanding the filesystem allows you to organize your data so that you can find information quickly. It also enables you to share some of your files with other users while keeping other files private. • Chapter 5—The Shell Explains how to use shell features to make your work faster and easier. All of the features covered in this chapter work with both bash and tcsh. This chapter discusses ◆ Using command-line options to modify the way a command works; ◆ Making minor changes in a command line to redirect input to a command so that it comes from a file instead of the keyboard; ◆ Redirecting output from a command to go to a file instead of the screen; ◆ Using pipes to send the output of one utility directly to another utility so you can solve problems right on the command line; ◆ Running programs in the background so you can work on one task while Linux is working on a different one; and ◆ Using the shell to generate filenames to save time spent on typing and help you when you do not remember the exact name of a file. Part II: The Editors Part II covers two classic, powerful Linux command-line text editors. Most Linux distributions include the vim text editor, an "improved" version of the widely used vi editor, as well as the popular GNU emacs editor. Text editors enable you to create and modify text files that can hold programs, shell scripts, memos, and input to text formatting programs. Because Linux system administration involves editing text-based configuration files, skilled Linux administrators are adept at using text editors. • Chapter 6—The vim Editor Starts with a tutorial on vim and then explains how to use many of the advanced features of vim, including special characters in search strings, the General-Purpose and Named buffers, parameters, markers, and execution of commands from within vim. The chapter concludes with a summary of vim commands. • Chapter 7—The emacs Editor Opens with a tutorial and then explains many of the features of the emacs editor as well as how to use the META, ALT, and ESCAPE keys. In addition, this Preface xxxvii chapter covers key bindings, buffers, and incremental and complete searching for both character strings and regular expressions. It details the relationship between Point, the cursor, Mark, and Region. It also explains how to take advantage of the extensive online help facilities available from emacs. Other topics covered include cutting and pasting, using multiple windows and frames, and working with emacs modes—specifically C mode, which aids programmers in writing and debugging C code. Chapter 7 concludes with a summary of emacs commands. Part III: The Shells Part III goes into more detail about bash and introduces the TC Shell (tcsh). • Chapter 8—The Bourne Again Shell Picks up where Chapter 5 left off, covering more advanced aspects of working with a shell. For examples it uses the Bourne Again Shell—bash, the shell used almost exclusively for system shell scripts. Chapter 8 describes how to ◆ Use shell startup files, shell options, and shell features to customize the shell; ◆ Use job control to stop jobs and move jobs from the foreground to the background, and vice versa; ◆ Modify and reexecute commands using the shell history list; ◆ Create aliases to customize commands; ◆ Work with user-created and keyword variables in shell scripts; ◆ Set up functions, which are similar to shell scripts but are executed more quickly; ◆ Write and execute simple shell scripts; and ◆ Redirect error messages so they go to a file instead of the screen. • Chapter 9—The TC Shell Describes tcsh and covers features common to and different between bash and tcsh. This chapter explains how to ◆ Run tcsh and change your default shell to tcsh; ◆ Redirect error messages so they go to files instead of the screen; ◆ Use control structures to alter the flow of control within shell scripts; ◆ Work with tcsh array and numeric variables; and ◆ Use shell builtin commands. xxxviii Preface Part IV: Programming Tools Part IV covers important programming tools that are used extensively in Linux and Mac OS X system administration and general-purpose programming. Chapter 10—Programming the Bourne Again Shell Continues where Chapter 8 left off, going into greater depth about advanced shell programming using bash, with the discussion enhanced by extensive examples. This chapter discusses ◆ Control structures such as if...then...else and case; ◆ Variables, including locality of variables; ◆ Arithmetic and logical (Boolean) expressions; and ◆ Some of the most useful shell builtin commands, including exec, trap, and getopts. Once you have mastered the basics of Linux, you can use your knowledge to build more complex and specialized programs, using the shell as a programming language. Chapter 10 poses two complete shell programming problems and then shows you how to solve them step by step. The first problem uses recursion to create a hierarchy of directories. The second problem develops a quiz program, shows you how to set up a shell script that interacts with a user, and explains how the script processes data. (The examples in Part V also demonstrate many features of the utilities you can use in shell scripts.) • Chapter 11—The Perl Scripting Language Introduces the popular, feature-rich Perl programming language. This chapter covers ◆ Perl help tools including perldoc; ◆ Perl variables and control structures; ◆ File handling; ◆ Regular expressions; and ◆ Installation and use of CPAN modules. Many Linux administration scripts are written in Perl. After reading Chapter 11 you will be able to better understand these scripts and start writing your own. This chapter includes many examples of Perl scripts. • Chapter 12—The AWK Pattern Processing Language Explains how to write programs using the powerful AWK language that filter data, write reports, and retrieve data from the Internet. The advanced programming section describes how to set up two-way communication with another program using a coprocess and how to obtain input over a network instead of from a local file. Preface xxxix • Chapter 13—The sed Editor Describes sed, the noninteractive stream editor that finds many applications as a filter within shell scripts. This chapter discusses how to use sed's buffers to write simple yet powerful programs and includes many examples. • Chapter 14—The rsync Secure Copy Utility Covers rsync, a secure utility that copies an ordinary file or directory hierarchy locally or between the local system and another system on a network. As you write programs, you can use this utility to back them up to another system. Part V: Command Reference Linux includes hundreds of utilities. Chapters 12, 13, and 14 as well as Part V provide extensive examples of the use of 100 of the most important utilities with which you can solve problems without resorting to programming in C. If you are already familiar with UNIX/Linux, this part of the book will be a valuable, easy-to-use reference. If you are not an experienced user, it will serve as a useful supplement while you are mastering the earlier sections of the book. Although the descriptions of the utilities in Chapters 12, 13, and 14 and Part V are presented in a format similar to that used by the Linux manual (man) pages, they are much easier to read and understand. These utilities are included because you will work with them day in and day out (for example, ls and cp), because they are powerful tools that are especially useful in shell scripts (sort, paste, and test), because they help you work with a Linux system (ps, kill, and fsck), or because they enable you to communicate with other systems (ssh, scp, and ftp). Each utility description includes complete explanations of its most useful options, differentiating between options supported under Mac OS X and those supported under Linux. The "Discussion" and "Notes" sections present tips and tricks for taking full advantage of the utility's power. The "Examples" sections demonstrate how to use these utilities in real life, alone and together with other utilities to generate reports, summarize data, and extract information. Take a look at the "Examples" sections for AWK (more than 20 pages, starting on page 541), ftp (page 707), and sort (page 819) to see how extensive these sections are. Part VI: Appendixes Part VI includes the appendixes, the glossary, and three indexes. • Appendix A—Regular Expressions Explains how to use regular expressions to take advantage of the hidden power of Linux. Many utilities, including grep, sed, vim, AWK, and Perl, accept regular expressions in place of simple strings of characters. A single regular expression can match many simple strings. xl Preface • Appendix B—Help Details the steps typically used to solve the problems you may encounter with a Linux system. This appendix also includes many links to Web sites that offer documentation, useful Linux and Mac OS X information, mailing lists, and software. • Appendix C—Keeping the System Up-to-Date Describes how to use tools to download software and keep a system current. This appendix includes information on ◆ yum—Downloads software from the Internet, keeping a system up-to-date and resolving dependencies as it goes. ◆ apt-get—An alternative to yum for keeping a system current. ◆ BitTorrent—Good for distributing large amounts of data such as Linux installation CDs. • Appendix D—Mac OS X Notes This appendix is a brief guide to Mac OS X features and quirks that may be unfamiliar to users who have been using Linux or other UNIX-like systems. • Glossary Defines more than 500 terms that pertain to the use of Linux and Mac OS X. • Indexes ◆ File Tree Index—Lists, in hierarchical fashion, most files mentioned in this book. These files are also listed in the Main index. ◆ Utility Index—Locates all utilities mentioned in this book. ◆ Main Index—Helps you find the information you want quickly. Supplements The author's home page (www.sobell.com) contains downloadable listings of the longer programs from this book as well as pointers to many interesting and useful Linux- and OS X-related sites on the World Wide Web, a list of corrections to the book, answers to even-numbered exercises, and a solicitation for corrections, comments, and suggestions. Thanks First and foremost I want to thank my editor at Prentice Hall, Mark L. Taub, who encouraged me and kept me on track. Mark is unique in my experience: He is an editor who works with the tools I am writing about. Because Mark runs Linux on Preface xli his home computer, we share experiences as I write. His comments and direction were invaluable. Thank you, Mark T. Molly Sharp of ContentWorks worked with me during the day-by-day production of this book, providing help, listening to my rants, and keeping everything on track. Thanks to Jill Hobbs, Copyeditor, who made the book readable, understandable, and consistent; and Andrea Fox, Proofreader, who made each page sparkle and found the mistakes that the author left behind. Thanks also to the folks at Prentice Hall who helped bring this book to life, especially Julie Nahil, Full-Service Production Manager, who oversaw production of the book; John Fuller, Managing Editor, who kept the large view in check; Brandon Prebynski, Marketing Manager; Curt Johnson, Executive Marketing Manager; Kim Boedigheimer, Editorial Assistant, who attended to the many details involved in publishing this book; Heather Fox, Publicist; Cheryl Lenser, Senior Indexer; Sandra Schroeder, Design Manager; Chuti Prasertsith, Cover Designer; and everyone else who worked behind the scenes to make this book come into being. A big "Thank You" to the folks who read through the drafts of the book and made comments that caused me to refocus parts of the book where things were not clear or were left out altogether: David L. Adam; Joe Barker, Ubuntu Forums LoCo Admin; Mike Basinger, Ubuntu Forum Administrator; Michael Davis, Systems Analyst, CSC; Andy Lester, author of Land the Tech Job You Love: Why Skill and Luck Are Not Enough, who helped extensively with the Perl chapter; John Peters, J&J Associates; Rich Rosen, Interactive Data; Max Sobell, New York University; Leon Towns-von Stauber; and Jarod Wilson, Senior Software Engineer, Red Hat, Inc. I am also indebted to Denis Howe, the editor of The Free On-Line Dictionary of Computing (FOLDOC). Denis has graciously permitted me to use entries from his compilation. Be sure to visit the dictionary (www.foldoc.org). Dr. Brian Kernighan and Rob Pike graciously allowed me to reprint the bundle script from their book, The UNIX Programming Environment (Prentice Hall, 1984). Thanks also to the people who helped with the first edition of this book: Lars Kellogg-Stedman, Harvard University; Jim A. Lola, Principal Systems Consultant, Privateer Systems, LLC; Eric S. Raymond, cofounder, Open Source Initiative; Scott Mann; Randall Lechlitner, Independent Computer Consultant; Jason Wertz, Computer Science Instructor, Montgomery County Community College; Justin Howell, Solano Community College; Ed Sawicki, The Accelerated Learning Center; David Mercer, Contechst; Jeffrey Bianchine, Advocate, Author, Journalist; John Kennedy; Chris Karr; and Jim Dennis, Starshine Technical Services. Parts of A Practical Guide to Linux® Commands, Editors, and Shell Programming, Second Edition, have grown from my previous Linux books and I want to thank the people who helped with those books. Thank you to John Dong, Ubuntu Developer, Forums Council Member; Matthew Miller, Senior Systems Analyst/Administrator, BU Linux Project, Boston University Office of Information Technology; George Vish II, Senior Education Consultant, Hewlett-Packard; James Stockford, Systemateka, Inc.; xlii Preface Stephanie Troeth, Book Oven; Doug Sheppard; Bryan Helvey, IT Director, OpenGeoSolutions; Vann Scott, Baker College of Flint; David Chisnall, computer scientist extraordinaire; Thomas Achtemichuk, Mansueto Ventures; Scott James Remnant, Ubuntu Development Manager and Desktop Team Leader; Daniel R. Arfsten, Pro/Engineer Drafter/Designer; Chris Cooper, Senior Education Consultant, HewlettPackard Education Services; Sameer Verma, Associate Professor of Information Systems, San Francisco State University; Valerie Chau, Palomar College and Programmers Guild; James Kratzer; Sean McAllister; Nathan Eckenrode, New York Ubuntu Local Community Team; Christer Edwards; Nicolas Merline; Michael Price; Sean Fagan, Apple Computer; Paul M. Lambert, Apple Computer; Nicolas Roard; Stephen Worotynec, Engineering Services, Alias Systems; Gretchen Phillips, Independent Consultant, GP Enterprises; Peggy Fenner; Carsten Pfeiffer, Software Engineer and KDE Developer; Aaron Weber, Ximian; Cristof Falk, Software Developer at CritterDesign; Steve Elgersma, Computer Science Department, Princeton University; Scott Dier, University of Minnesota; and Robert Haskins, Computer Net Works. Thanks also to Dustin Puryear, Puryear Information Technology; Gabor Liptak, Independent Consultant; Bart Schaefer, Chief Technical Officer, iPost; Michael J. Jordan, Web Developer, Linux Online Inc.; Steven Gibson, owner of SuperAnt.com; John Viega, founder and Chief Scientist, Secure Software, Inc.; K. Rachael Treu, Internet Security Analyst, Global Crossing; Kara Pritchard, K & S Pritchard Enterprises, Inc.; Glen Wiley, Capital One Finances; Karel Baloun, Senior Software Engineer, Looksmart, Ltd.; Matthew Whitworth; Dameon D. Welch-Abernathy, Nokia Systems; Josh Simon, Consultant; Stan Isaacs; Dr. Eric H. Herrin II, Vice President, Herrin Software Development, Inc. And thanks to Doug Hughes, long-time system designer and administrator, who gave me a big hand with the sections on system administration, networks, the Internet, and programming. More thanks go to consultants Lorraine Callahan and Steve Wampler; Ronald Hiller, Graburn Technology, Inc.; Charles A. Plater, Wayne State University; Bob Palowoda; Tom Bialaski, Sun Microsystems; Roger Hartmuller, TIS Labs at Network Associates; Kaowen Liu; Andy Spitzer; Rik Schneider; Jesse St. Laurent; Steve Bellenot; Ray W. Hiltbrand; Jennifer Witham; Gert-Jan Hagenaars; and Casper Dik. A Practical Guide to Linux® Commands, Editors, and Shell Programming, Second Edition, is based in part on two of my previous UNIX books: UNIX System V: A Practical Guide and A Practical Guide to the UNIX System. Many people helped me with those books, and thanks here go to Pat Parseghian, Dr. Kathleen Hemenway, Brian LaRose; Byron A. Jeff, Clark Atlanta University; Charles Stross; Jeff Gitlin, Lucent Technologies; Kurt Hockenbury; Maury Bach, Intel Israel Ltd.; Peter H. Salus; Rahul Dave, University of Pennsylvania; Sean Walton, Intelligent Algorithmic Solutions; Tim Segall, Computer Sciences Corporation; Behrouz Forouzan, DeAnza College; Mike Keenan, Virginia Polytechnic Institute and State University; Mike Johnson, Oregon State University; Jandelyn Plane, University of Maryland; Arnold Robbins and Sathis Menon, Georgia Institute of Technology; Cliff Shaffer, Virginia Polytechnic Institute and State University; and Steven Stepanek, California State University, Northridge. Preface xliii I continue to be grateful to the many people who helped with the early editions of my UNIX books. Special thanks are due to Roger Sippl, Laura King, and Roy Harrington for introducing me to the UNIX system. My mother, Dr. Helen Sobell, provided invaluable comments on the original manuscript at several junctures. Also, thanks go to Isaac Rabinovitch, Professor Raphael Finkel, Professor Randolph Bentson, Bob Greenberg, Professor Udo Pooch, Judy Ross, Dr. Robert Veroff, Dr. Mike Denny, Joe DiMartino, Dr. John Mashey, Diane Schulz, Robert Jung, Charles Whitaker, Don Cragun, Brian Dougherty, Dr. Robert Fish, Guy Harris, Ping Liao, Gary Lindgren, Dr. Jarrett Rosenberg, Dr. Peter Smith, Bill Weber, Mike Bianchi, Scooter Morris, Clarke Echols, Oliver Grillmeyer, Dr. David Korn, Dr. Scott Weikart, and Dr. Richard Curtis. I take responsibility for any errors and omissions in this book. If you find one or just have a comment, let me know () and I will fix it in the next printing. My home page (www.sobell.com) contains a list of errors and credits those who found them. It also offers copies of the longer scripts from the book and pointers to many interesting Linux pages. Mark G. Sobell San Francisco, California This page intentionally left blank 1 Welcome to Linux and Mac OS X In This Chapter An operating system is the low-level software that schedules tasks, allocates storage, and handles the interfaces to peripheral hardware, such as printers, disk drives, the screen, keyboard, and mouse. An operating system has two main parts: the kernel and the system programs. The kernel allocates machine resources—including memory, disk space, and CPU (page 949) cycles—to all other programs that run on the computer. The system programs include device drivers, libraries, utility programs, shells (command interpreters), configuration scripts and files, application programs, servers, and documentation. They perform higher-level housekeeping tasks, often acting as servers in a client/server relationship. Many of the libraries, servers, and utility programs were written by the GNU Project, which is discussed shortly. 1Chapter1 1 2 Chapter 1 Welcome to Linux and Mac OS X Linux kernel The Linux kernel was developed by Finnish undergraduate student Linus Torvalds, who used the Internet to make the source code immediately available to others for free. Torvalds released Linux version 0.01 in September 1991. The new operating system came together through a lot of hard work. Programmers around the world were quick to extend the kernel and develop other tools, adding functionality to match that already found in both BSD UNIX and System V UNIX (SVR4) as well as new functionality. The name Linux is a combination of Linus and UNIX. The Linux operating system, which was developed through the cooperation of many, many people around the world, is a product of the Internet and is a free (open source; page 969) operating system. In other words, all the source code is free. You are free to study it, redistribute it, and modify it. As a result, the code is available free of cost—no charge for the software, source, documentation, or support (via newsgroups, mailing lists, and other Internet resources). As the GNU Free Software Definition (www.gnu.org/philosophy/free-sw.html) puts it: Free beer Mach kernel "Free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer." OS X runs the Mach kernel, which was developed at Carnegie Mellon University (CMU) and is free software. CMU concluded its work on the project in 1994, although other groups have continued this line of research. Much of the Mac OS X software is open source: the Mac OS X kernel is based on Mach and FreeBSD code, utilities come from BSD and the GNU project, and system programs come mostly from BSD code, although Apple has developed a number of new programs. Linux, OS X, and UNIX tip Linux and OS X are closely related to the UNIX operating system. This book describes Linux and OS X. To make reading easier, this book talks about Linux when it means OS X and Linux, and points out where OS X behaves differently from Linux. For the same reason, this chapter frequently uses the term Linux to describe both Linux and OS X features. The History of UNIX and GNU–Linux This section presents some background on the relationships between UNIX and Linux and between GNU and Linux. Go to www.levenez.com/unix for an impressive diagram of the history of UNIX. The Heritage of Linux: UNIX The UNIX system was developed by researchers who needed a set of modern computing tools to help them with their projects. The system allowed a group of people working together on a project to share selected data and programs while keeping other information private. Universities and colleges played a major role in furthering the popularity of the UNIX operating system through the "four-year effect." When the UNIX operating The History of UNIX and GNU–Linux 3 system became widely available in 1975, Bell Labs offered it to educational institutions at nominal cost. The schools, in turn, used it in their computer science programs, ensuring that computer science students became familiar with it. Because UNIX was such an advanced development system, the students became acclimated to a sophisticated programming environment. As these students graduated and went into industry, they expected to work in a similarly advanced environment. As more of them worked their way up the ladder in the commercial world, the UNIX operating system found its way into industry. BSD (Berkeley) UNIX In addition to introducing students to the UNIX operating system, the Computer Systems Research Group (CSRG) at the University of California at Berkeley made significant additions and changes to it. In fact, it made so many popular changes that one version of the system is called the Berkeley Software Distribution (BSD) of the UNIX system (or just Berkeley UNIX). The other major version is UNIX System V (SVR4), which descended from versions developed and maintained by AT&T and UNIX System Laboratories. Mac OS X inherits much more strongly from the BSD branch of the tree. Fade to 1983 Richard Stallman (www.stallman.org) announced1 the GNU Project for creating an operating system, both kernel and system programs, and presented the GNU Manifesto,2 which begins as follows: GNU, which stands for Gnu's Not UNIX, is the name for the complete UNIX-compatible software system which I am writing so that I can give it away free to everyone who can use it. Some years later, Stallman added a footnote to the preceding sentence when he realized that it was creating confusion: The wording here was careless. The intention was that nobody would have to pay for *permission* to use the GNU system. But the words don't make this clear, and people often interpret them as saying that copies of GNU should always be distributed at little or no charge. That was never the intent; later on, the manifesto mentions the possibility of companies providing the service of distribution for a profit. Subsequently I have learned to distinguish carefully between "free" in the sense of freedom and "free" in the sense of price. Free software is software that users have the freedom to distribute and change. Some users may obtain copies at no charge, while others pay to obtain copies—and if the funds help support improving the software, so much the better. The important thing is that everyone who has a copy has the freedom to cooperate with others in using it. 1. www.gnu.org/gnu/initial-announcement.html 2. www.gnu.org/gnu/manifesto.html 4 Chapter 1 Welcome to Linux and Mac OS X In the manifesto, after explaining a little about the project and what has been accomplished so far, Stallman continues: Why I Must Write GNU I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will. So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. I have resigned from the AI Lab to deny MIT any legal excuse to prevent me from giving GNU away. Next Scene, 1991 The GNU Project has moved well along toward its goal. Much of the GNU operating system, except for the kernel, is complete. Richard Stallman later writes: By the early '90s we had put together the whole system aside from the kernel (and we were also working on a kernel, the GNU Hurd,3 which runs on top of Mach4). Developing this kernel has been a lot harder than we expected, and we are still working on finishing it.5 ...[M]any believe that once Linus Torvalds finished writing the kernel, his friends looked around for other free software, and for no particular reason most everything necessary to make a UNIX-like system was already available. What they found was no accident—it was the GNU system. The available free software6 added up to a complete system because the GNU Project had been working since 1984 to make one. The GNU Manifesto had set forth the goal of developing a free UNIX-like system, called GNU. The Initial Announcement of the GNU Project also outlines some of the original plans for the GNU system. By the time Linux was written, the [GNU] system was almost finished.7 3. www.gnu.org/software/hurd/hurd.html 4. www.gnu.org/software/hurd/gnumach.html 5. www.gnu.org/software/hurd/hurd-and-linux.html 6. www.gnu.org/philosophy/free-sw.html 7. www.gnu.org/gnu/linux-and-gnu.html The History of UNIX and GNU–Linux 5 Today the GNU "operating system" runs on top of the FreeBSD (www.freebsd.org) and NetBSD (www.netbsd.org) kernels with complete Linux binary compatibility and on top of Hurd pre-releases and Darwin (developer.apple.com/opensource) without this compatibility. The Code Is Free The tradition of free software dates back to the days when UNIX was released to universities at nominal cost, which contributed to its portability and success. This tradition eventually died as UNIX was commercialized and manufacturers came to regard the source code as proprietary, making it effectively unavailable. Another problem with the commercial versions of UNIX related to their complexity. As each manufacturer tuned UNIX for a specific architecture, the operating system became less portable and too unwieldy for teaching and experimentation. MINIX Two professors created their own stripped-down UNIX look-alikes for educational purposes: Doug Comer created XINU and Andrew Tanenbaum created MINIX. Linus Torvalds created Linux to counteract the shortcomings in MINIX. Every time there was a choice between code simplicity and efficiency/features, Tanenbaum chose simplicity (to make it easy to teach with MINIX), which meant this system lacked many features people wanted. Linux went in the opposite direction. You can obtain Linux at no cost over the Internet. You can also obtain the GNU code via the U.S. mail at a modest cost for materials and shipping. You can support the Free Software Foundation (www.fsf.org) by buying the same (GNU) code in higher-priced packages, and you can buy commercial packaged releases of Linux (called distributions; e.g., Ubuntu, Red Hat, openSUSE), that include installation instructions, software, and support. GPL Linux and GNU software are distributed under the terms of the GNU General Public License (GPL, www.gnu.org/licenses/licenses.html). The GPL says you have the right to copy, modify, and redistribute the code covered by the agreement. When you redistribute the code, however, you must also distribute the same license with the code, thereby making the code and the license inseparable. If you get source code off the Internet for an accounting program that is under the GPL and then modify that code and redistribute an executable version of the program, you must also distribute the modified source code and the GPL agreement with it. Because this arrangement is the reverse of the way a normal copyright works (it gives rights instead of limiting them), it has been termed a copyleft. (This paragraph is not a legal interpretation of the GPL; it is intended merely to give you an idea of how it works. Refer to the GPL itself when you want to make use of it.) Have Fun! Two key words for Linux are "Have Fun!" These words pop up in prompts and documentation. The UNIX—now Linux—culture is steeped in humor that can be seen throughout the system. For example, less is more—GNU has replaced the UNIX paging utility named more with an improved utility named less. The utility to view PostScript documents is named ghostscript, and one of several replacements for 6 Chapter 1 Welcome to Linux and Mac OS X the vi editor is named elvis. While machines with Intel processors have "Intel Inside" logos on their outside, some Linux machines sport "Linux Inside" logos. And Torvalds himself has been seen wearing a T-shirt bearing a "Linus Inside" logo. What Is So Good About Linux? In recent years Linux has emerged as a powerful and innovative UNIX work-alike. Its popularity has surpassed that of its UNIX predecessors. Although it mimics UNIX in many ways, the Linux operating system departs from UNIX in several significant ways: The Linux kernel is implemented independently of both BSD and System V, the continuing development of Linux is taking place through the combined efforts of many capable individuals throughout the world, and Linux puts the power of UNIX within easy reach of both business and personal computer users. Using the Internet, today's skilled programmers submit additions and improvements to the operating system to Linus Torvalds, GNU, or one of the other authors of Linux. Standards In 1985, individuals from companies throughout the computer industry joined together to develop the POSIX (Portable Operating System Interface for Computer Environments) standard, which is based largely on the UNIX System V Interface Definition (SVID) and other earlier standardization efforts. These efforts were spurred by the U.S. government, which needed a standard computing environment to minimize its training and procurement costs. Released in 1988, POSIX is a group of IEEE standards that define the API (application program interface; page 941), shell, and utility interfaces for an operating system. Although aimed at UNIX-like systems, the standards can apply to any compatible operating system. Now that these standards have gained acceptance, software developers are able to develop applications that run on all conforming versions of UNIX, Linux, and other operating systems. Applications A rich selection of applications is available for Linux—both free and commercial— as well as a wide variety of tools: graphical, word processing, networking, security, administration, Web server, and many others. Large software companies have recently seen the benefit in supporting Linux and now have on-staff programmers whose job it is to design and code the Linux kernel, GNU, KDE, or other software that runs on Linux. For example, IBM (www.ibm.com/linux) is a major Linux supporter. Linux conforms increasingly more closely to POSIX standards, and some distributions and parts of others meet this standard. These developments indicate that Linux is becoming mainstream and is respected as an attractive alternative to other popular operating systems. Peripherals Another aspect of Linux that appeals to users is the amazing range of peripherals that is supported and the speed with which support for new peripherals emerges. Linux often supports a peripheral or interface card before any

company does. Unfortunately some types of peripherals—particularly proprietary graphics cards—lag in their support because the manufacturers do not release specifications or source code for drivers in a timely manner, if at all. What Is So Good About Linux? 7 Software Also important to users is the amount of software that is available—not just source code (which needs to be compiled) but also prebuilt binaries that are easy to install and ready to run. These programs include more than free software. Netscape, for example, has been available for Linux from the start and included Java support before it was available from many commercial vendors. Its sibling Mozilla/Thunderbird/Firefox is also a viable browser, mail client, and newsreader, performing many other functions as well. Platforms Linux is not just for Intel-based platforms (which now include Apple computers): It has been ported to and runs on the Power PC—including older Apple computers (ppclinux), Compaq's (née Digital Equipment Corporation) Alpha-based machines, MIPS-based machines, Motorola's 68K-based machines, various 64-bit systems, and IBM's S/390. Nor is Linux just for single-processor machines: As of version 2.0, it runs on multiple-processor machines (SMPs; page 978). It also includes an O(1) scheduler, which dramatically increases scalability on SMP systems. Emulators Linux supports programs, called emulators, that run code intended for other operating systems. By using emulators you can run some DOS, Windows, and Macintosh programs under Linux. For example, Wine (www.winehq.com) is an open-source implementation of the Windows API that runs on top of the X Window System and UNIX/Linux. Virtual machines A virtual machine (VM or guest) appears to the user and to the software running on it as a complete physical machine. It is, however, one of potentially many such VMs running on a single physical machine (the host). The software that provides the virtualization is called a virtual machine monitor (VMM) or hypervisor. Each VM can run a different operating system from the other VMs. For example, on a single host you could have VMs running Windows, Ubuntu 7.10, Ubuntu 9.04, and Fedora 10. A multitasking operating system allows you to run many programs on a single physical system. Similarly, a hypervisor allows you to run many operating systems (VMs) on a single physical system. VMs provide many advantages over single, dedicated machines: • Isolation—Each VM is isolated from the other VMs running on the same host: Thus, if one VM crashes or is compromised, the others are not affected. • Security—When a single server system running several servers is compromised, all servers are compromised. If each server is running on its own VM, only the compromised server is affected; other servers remain secure. • Power consumption—Using VMs, a single powerful machine can replace many less powerful machines, thereby cutting power consumption. • Development and support—Multiple VMs, each running a different version of an operating system and/or different operating systems, can facilitate development and support of software designed to run in many environments. With this organization you can easily test a product in different environments before releasing it. Similarly, when a user submits a bug, you can reproduce the bug in the same environment it occurred in. • Servers—In some cases, different servers require different versions of system libraries. In this case, you can run each server on its own VM, all on a single piece of hardware. • Testing—Using VMs, you can experiment with cutting-edge releases of operating systems and applications without concern for the base (stable) system, all on a single machine. • Networks—You can set up and test networks of systems on a single machine. • Sandboxes—A VM presents a sandbox—an area (system) that you can work in without regard for the results of your work or for the need to clean up. • Snapshots—You can take snapshots of a VM and return the VM to the state it was in when you took the snapshot simply by reloading the VM from the snapshot. Xen Xen, which was created at the University of Cambridge and is now being developed in the open-source community, is an open-source VMM. Xen introduces minimal performance overhead when compared with running each of the operating systems natively. This book does not cover the installation or use of Xen. For more information on Xen, refer to the Xen home page at www.cl.cam.ac.uk/research/srg/netos/xen and to wiki.xensource.com/xenwiki. VMware VMware, Inc. (www.vmware.com) offers VMware Server, a free, downloadable, proprietary product you can install and run as an application under Linux. VMware Server enables you to install several VMs, each running a different operating system, including Windows and Linux. VMware also offers a free VMware player that enables you to run VMs you create with the VMware Server. KVM The Kernel-based Virtual Machine (KVM; kvm.qumranet.com and libvirt.org) is an open-source VM and runs as part of the Linux kernel. It works only on systems based on the Intel VT (VMX) CPU or the AMD SVM CPU. Qemu Qemu (bellard.org/qemu), written by Fabrice Bellard, is an open-source VMM that runs as a user application with no CPU requirements. It can run code written for a different CPU than that of the host machine. VirtualBox VirtualBox (www.virtualbox.org) is an open-source VM developed by Sun Microsystems. What Is So Good About Linux? 9 Why Linux Is Popular with Hardware Companies and Developers Two trends in the computer industry set the stage for the growing popularity of UNIX and Linux. First, advances in hardware technology created the need for an operating system that could take advantage of available hardware power. In the mid-1970s, minicomputers began challenging the large mainframe computers because, in many applications, minicomputers could perform the same functions less expensively. More recently, powerful 64-bit processor chips, plentiful and inexpensive memory, and lower-priced hard disk storage have allowed hardware companies to install multiuser operating systems on desktop computers. Proprietary operating systems Second, with the cost of hardware continually dropping, hardware manufacturers could no longer afford to develop and support proprietary operating systems. A proprietary operating system is one that is written and owned by the manufacturer of the hardware (for example, DEC/Compaq owns VMS). Today's manufacturers need a generic operating system that they can easily adapt to their machines. Generic operating systems A generic operating system is written outside of the company manufacturing the hardware and is sold (UNIX, OS X, Windows) or given (Linux) to the manufacturer. Linux is a generic operating system because it runs on different types of hardware produced by different manufacturers. Of course, if manufacturers can pay only for development and avoid per-unit costs (which they have to pay to Microsoft for each copy of Windows they sell), they are much better off. In turn, software developers need to keep the prices of their products down; they cannot afford to create new versions of their products to run under many different proprietary operating systems. Like hardware manufacturers, software developers need a generic operating system. Although the UNIX system once met the needs of hardware companies and researchers for a generic operating system, over time it has become more proprietary as manufacturers added support for their own specialized features and introduced new software libraries and utilities. Linux emerged to serve both needs: It is a generic operating system that takes advantage of available hardware. Linux Is Portable A portable operating system is one that can run on many different machines. More than 95 percent of the Linux operating system is written in the C programming language, and C is portable because it is written in a higher-level, machine-independent language. (The C compiler is written in C.) Because Linux is portable, it can be adapted (ported) to different machines and can meet special requirements. For example, Linux is used in embedded computers, such as the ones found in cellphones, PDAs, and the cable boxes on top of many 10 Chapter 1 Welcome to Linux and Mac OS X TVs. The file structure takes full advantage of large, fast hard disks. Equally important, Linux was originally designed as a multiuser operating system—it was not modified to serve several users as an afterthought. Sharing the computer's power among many users and giving them the ability to share data and programs are central features of the system. Because it is adaptable and takes advantage of available hardware, Linux runs on many different microprocessor-based systems as well as mainframes. The popularity of the microprocessor-based hardware drives Linux; these microcomputers are getting faster all the time, at about the same price point. Linux on a fast microcomputer has become good enough to displace workstations on many desktops. This widespread acceptance benefits both users, who do not like having to learn a new operating system for each vendor's hardware, and system administrators, who like having a consistent software environment. The advent of a standard operating system has given a boost to the development of the software industry. Now software manufacturers can afford to make one version of a product available on machines from different manufacturers. The C Programming Language Ken Thompson wrote the UNIX operating system in 1969 in PDP-7 assembly language. Assembly language is machine dependent: Programs written in assembly language work on only one machine or, at best, on one family of machines. For this reason, the original UNIX operating system could not easily be transported to run on other machines (it was not portable). To make UNIX portable, Thompson developed the B programming language, a machine-independent language, from the BCPL language. Dennis Ritchie developed the C programming language by modifying B and, with Thompson, rewrote UNIX in C in 1973. Originally, C was touted as a "portable assembler." The revised operating system could be transported more easily to run on other machines. That development marked the start of C. Its roots reveal some of the reasons why it is such a powerful tool. C can be used to write machine-independent programs. A programmer who designs a program to be portable can easily move it to any computer that has a C compiler. C is also designed to compile into very efficient code. With the advent of C, a programmer no longer had to resort to assembly language to get code that would run well (that is, quickly—although an assembler will always generate more efficient code than a high-level language). C is a good systems language. You can write a compiler or an operating system in C. It is a highly structured but is not necessarily a high-level language. C allows a programmer to manipulate bits and bytes, as is necessary when writing an operating system. At the same time, it has high-level constructs that allow for efficient, modular programming. In the late 1980s the American National Standards Institute (ANSI) defined a standard version of the C language, commonly referred to as ANSI C or C89 (for the year the standard was published). Ten years later the C99 standard was published; it is mostly supported by the GNU Project's C compiler (named gcc). The original version of the language is often referred to as Kernighan & Ritchie (or K&R) C, named for the authors of the book that first described the C language. Another researcher at Bell Labs, Bjarne Stroustrup, created an object-oriented programming language named C++, which is built on the foundation of C. Because object-oriented programming is desired by many employers today, C++ is preferred over C in many environments. Another language of choice is Objective-C, which was used to write the first Web browser. The GNU Project's C compiler supports C, C++, and Objective-C. Overview of Linux 11 Compilers Database Management Systems Word Processors Mail and Message Facilities Shells Linux Kernel Hardware Figure 1-1 A layered view of the Linux operating system Overview of Linux The Linux operating system has many unique and powerful features. Like other operating systems, it is a control program for computers. But like UNIX, it is also a well-thought-out family of utility programs (Figure 1-1) and a set of tools that allow users to connect and use these utilities to build systems and applications. Linux Has a Kernel Programming Interface The Linux kernel—the heart of the Linux operating system—is responsible for allocating the computer's resources and scheduling user jobs so each one gets its fair share of system resources, including access to the CPU; peripheral devices, such as hard disk, DVD, and CD-ROM storage; printers; and tape drives. Programs interact with the kernel through system calls, special functions with well-known names. A programmer can use a single system call to interact with many kinds of devices. For example, there is one write() system call, rather than many device-specific ones. When a program issues a write() request, the kernel interprets the context and passes the request to the appropriate device. This flexibility allows old utilities to work with devices that did not exist when the utilities were written. It also makes it 12 Chapter 1 Welcome to Linux and Mac OS X possible to move programs to new versions of the operating system without rewriting them (provided the new version recognizes the same system calls). Linux Can Support Many Users Depending on the hardware and the types of tasks the computer performs, a Linux system can support from 1 to more than 1,000 users, each concurrently running a different set of programs. The per-user cost of a computer that can be used by many people at the same time is less than that of a computer that can be used by only a single person at a time. It is less because one person cannot generally take advantage of all the resources a computer has to offer. That is, no one can keep all the printers going constantly, keep all the system memory in use, keep all the disks busy reading and writing, keep the Internet connection in use, and keep all the terminals busy at the same time. By contrast, a multiuser operating system allows many people to use all of the system resources almost simultaneously. The use of costly resources can be maximized and the cost per user can be minimized—the primary objectives of a multiuser operating system. Linux Can Run Many Tasks Linux is a fully protected multitasking operating system, allowing each user to run more than one job at a time. Processes can communicate with one another but remain fully protected from one another, just as the kernel remains protected from all processes. You can run several jobs in the background while giving all your attention to the job being displayed on the screen, and you can switch back and forth between jobs. If you are running the X Window System (page 16), you can run different programs in different windows on the same screen and watch all of them. This capability helps users be more productive. Linux Provides a Secure Hierarchical Filesystem A file is a collection of information, such as text for a memo or report, an accumulation of sales figures, an image, a song, or an executable program. Each file is stored under a unique identifier on a storage device, such as a hard disk. The Linux filesystem provides a structure whereby files are arranged under directories, which are like folders or boxes. Each directory has a name and can hold other files and directories. Directories, in turn, are arranged under other directories, and so forth, in a treelike organization. This structure helps users keep track of large numbers of files by grouping related files in directories. Each user has one primary directory and as many subdirectories as required (Figure 1-2). Standards With the idea of making life easier for system administrators and software developers, a group got together over the Internet and developed the Linux Filesystem Standard (FSSTND), which has since evolved into the Linux Filesystem Hierarchy Standard (FHS). Before this standard was adopted, key programs were located in different places in different Linux distributions. Today you can sit down at a Linux system and expect to find any given standard program at a consistent location (page 91). Overview of Linux 13 / home tmp etc max sam hls bin report Figure 1-2 notes log The Linux filesystem structure Links A link allows a given file to be accessed by means of two or more names. The alternative names can be located in the same directory as the original file or in another directory. Links can make the same file appear in several users' directories, enabling those users to share the file easily. Windows uses the term shortcut in place of link to describe this capability. Macintosh users will be more familiar with the term alias. Under Linux, an alias is different from a link; it is a command macro feature provided by the shell (page 324). Security Like most multiuser operating systems, Linux allows users to protect their data from access by other users. It also allows users to share selected data and programs with certain other users by means of a simple but effective protection scheme. This level of security is provided by file access permissions, which limit the users who can read from, write to, or execute a file. More recently, Linux has implemented Access Control Lists (ACLs), which give users and administrators finer-grained control over file access permissions. The Shell: Command Interpreter and Programming Language In a textual environment, the shell—the command interpreter—acts as an interface between you and the operating system. When you enter a command on the screen, the shell interprets the command and calls the program you want. A number of shells are available for Linux. The four most popular shells are • The Bourne Again Shell (bash), an enhanced version of the original Bourne Shell (the original UNIX shell). • The Debian Almquist Shell (dash), a smaller version of bash, with fewer features. Most startup shell scripts call dash in place of bash to speed the boot process. 14 Chapter 1 Welcome to Linux and Mac OS X • The TC Shell (tcsh), an enhanced version of the C Shell, developed as part of BSD UNIX. • The Z Shell (zsh), which incorporates features from a number of shells, including the Korn Shell. Because different users may prefer different shells, multiuser systems can have several different shells in use at any given time. The choice of shells demonstrates one of the advantages of the Linux operating system: the ability to provide a customized interface for each user. Shell scripts Besides performing its function of interpreting commands from a keyboard and sending those commands to the operating system, the shell is a high-level programming language. Shell commands can be arranged in a file for later execution (Linux calls these files shell scripts; Windows calls them batch files). This flexibility allows users to perform complex operations with relative ease, often by using rather short commands, or to build with surprisingly little effort elaborate programs that perform highly complex operations. Filename Generation Wildcards and ambiguous file references When you type commands to be processed by the shell, you can construct patterns using characters that have special meanings to the shell. These characters are called wildcard characters. The patterns, which are called ambiguous file references, are a kind of shorthand: Rather than typing in complete filenames, you can type patterns; the shell expands these patterns into matching filenames. An ambiguous file reference can save you the effort of typing in a long filename or a long series of similar filenames. For example, the shell might expand the pattern mak* to make-3.80.tar.gz. Patterns can also be useful when you know only part of a filename or cannot remember the exact spelling of a filename. Completion In conjunction with the Readline library, the shell performs command, filename, pathname, and variable completion: You type a prefix and press ESCAPE, and the shell lists the items that begin with that prefix or completes the item if the prefix specifies a unique item. Device-Independent Input and Output Redirection Devices (such as a printer or a terminal) and disk files appear as files to Linux programs. When you give a command to the Linux operating system, you can instruct it to send the output to any one of several devices or files. This diversion is called output redirection. Device independence In a similar manner, a program's input, which normally comes from a keyboard, can be redirected so that it comes from a disk file instead. Input and output are device independent; that is, they can be redirected to or from any appropriate device. Overview of Linux 15 As an example, the cat utility normally displays the contents of a file on the screen. When you run a cat command, you can easily cause its output to go to a disk file instead of the screen. Shell Functions One of the most important features of the shell is that users can use it as a programming language. Because the shell is an interpreter, it does not compile programs written for it but rather interprets programs each time they are loaded from the disk. Loading and interpreting programs can be time-consuming. Many shells, including the Bourne Again Shell, support shell functions that the shell holds in memory so it does not have to read them from the disk each time you execute them. The shell also keeps functions in an internal format so it does not have to spend as much time interpreting them. Job Control Job control is a shell feature that allows users to work on several jobs at once, switching back and forth between them as desired. When you start a job, it is frequently run in the foreground so it is connected to the terminal. Using job control, you can move the job you are working with to the background and continue running it there while working on or observing another job in the foreground. If a background job then needs your attention, you can move it to the foreground so it is once again attached to the terminal. (The concept of job control originated with BSD UNIX, where it appeared in the C Shell.) A Large Collection of Useful Utilities Linux includes a family of several hundred utility programs, often referred to as commands. These utilities perform functions that are universally required by users. The sort utility, for example, puts lists (or groups of lists) in alphabetical or numerical order and can be used to sort lists by part number, last name, city, ZIP code, telephone number, age, size, cost, and so forth. The sort utility is an important programming tool that is part of the standard Linux system. Other utilities allow users to create, display, print, copy, search, and delete files as well as to edit, format, and typeset text. The man (for manual) and info utilities provide online documentation for Linux. Interprocess Communication Pipes and filters Linux enables users to establish both pipes and filters on the command line. A pipe sends the output of one program to another program as input. A filter is a special kind of pipe that processes a stream of input data to yield a stream of output data. A filter processes another program's output, altering it as a result. The filter's output then becomes input to another program. 16 Chapter 1 Welcome to Linux and Mac OS X Pipes and filters frequently join utilities to perform a specific task. For example, you can use a pipe to send the output of the sort utility to head (a filter that lists the first ten lines of its input); you can then use another pipe to send the output of head to a third utility, lpr, that sends the data to a printer. Thus, in one command line, you can use three utilities together to sort and print part of a file. System Administration On a Linux system the system administrator is frequently the owner and only user of the system. This person has many responsibilities. The first responsibility may be to set up the system, install the software, and possibly edit configuration files. Once the system is up and running, the system administrator is responsible for downloading and installing software (including upgrading the operating system), backing up and restoring files, and managing such system facilities as printers, terminals, servers, and a local network. The system administrator is also responsible for setting up accounts for new users on a multiuser system, bringing the system up and down as needed, monitoring the system, and taking care of any problems that arise. Additional Features of Linux The developers of Linux included features from BSD, System V, and Sun Microsystems' Solaris, as well as new features, in their operating system. Although most of the tools found on UNIX exist for Linux, in some cases these tools have been replaced by more modern counterparts. This section describes some of the popular tools and features available under Linux. GUIs: Graphical User Interfaces X11 The X Window System (also called X or X11) was developed in part by researchers at the Massachusetts Institute of Technology (MIT) and provides the foundation for the GUIs available with Linux. Given a terminal or workstation screen that supports X, a user can interact with the computer through multiple windows on the screen, display graphical information, or use special-purpose applications to draw pictures, monitor processes, or preview formatted output. X is an across-the-network protocol that allows a user to open a window on a workstation or computer system that is remote from the CPU generating the window. Aqua Mac OS X comes with two graphical interfaces that can be used simultaneously. Most Macintosh users are familiar with Aqua, the standard Mac OS X graphical interface. Aqua is based on a rendering technology named Quartz and has a standard look and feel for applications. Mac OS X also supports X11, which also uses Quartz. Desktop manager Usually two layers run on top of X: a desktop manager and a window manager. A desktop manager is a picture-oriented user interface that enables you to interact with system programs by manipulating icons instead of typing the corresponding Additional Features of Linux 17 commands to a shell. Most Linux distributions run the GNOME desktop manager (www.gnome.org) by default, but they can also run KDE (www.kde.org) and a number of other desktop managers. Mac OS X handles the desktop in Aqua, not in X11, so there is no desktop manager under X11. Window manager A window manager is a program that runs under the desktop manager and allows you to open and close windows, run programs, and set up a mouse so it has different effects depending on how and where you click. The window manager also gives the screen its personality. Whereas Microsoft Windows allows you to change the color of key elements in a window, a window manager under X allows you to customize the overall look and feel of the screen: You can change the way a window looks and works (by giving it different borders, buttons, and scrollbars), set up virtual desktops, create menus, and more. Several popular window managers run under X and Linux. Most Linux distributions provide both Metacity (the default under GNOME) and kwin (the default under KDE). Other window managers, such as Sawfish and WindowMaker, are also available. Under Mac OS X, most windows are managed by a Quartz layer, which applies the Apple Aqua look and feel. For X11 applications only, this task is performed by quartz-wm, which mimics the Apple Aqua look and feel so X11 applications on the Mac desktop have the same appearance as native Mac OS X applications. (Inter)Networking Utilities Linux network support includes many utilities that enable you to access remote systems over a variety of networks. In addition to sending email to users on other systems, you can access files on disks mounted on other computers as if they were located on the local system, make your files available to other systems in a similar manner, copy files back and forth, run programs on remote systems while displaying the results on the local system, and perform many other operations across local area networks (LANs) and wide area networks (WANs), including the Internet. Layered on top of this network access is a wide range of application programs that extend the computer's resources around the globe. You can carry on conversations with people throughout the world, gather information on a wide variety of subjects, and download new software over the Internet quickly and reliably. Software Development One of Linux's most impressive strengths is its rich software development environment. Linux supports compilers and interpreters for many computer languages. Besides C and C++, languages available for Linux include Ada, Fortran, Java, Lisp, Pascal, Perl, and Python. The bison utility generates parsing code that makes it easier to write programs to build compilers (tools that parse files containing structured information). The flex utility generates scanners (code that recognizes lexical patterns in text). The make utility and the GNU Configure and Build System make it 18 Chapter 1 Welcome to Linux and Mac OS X easier to manage complex development projects. Source code management systems, such as CVS, simplify version control. Several debuggers, including ups and gdb, can help you track down and repair software defects. The GNU C compiler (gcc) works with the gprof profiling utility to help programmers identify potential bottlenecks in a program's performance. The C compiler includes options to perform extensive checking of C code, thereby making the code more portable and reducing debugging time. Table B-4 on page 904 lists some sites you can download software from. Under OS X, Apple's Xcode development environment provides a unified graphical front end to most of these tools as well as other options and features. Chapter Summary The Linux operating system grew out of the UNIX heritage to become a popular alternative to traditional systems (that is, Windows) available for microcomputer (PC) hardware. UNIX users will find a familiar environment in Linux. Distributions of Linux contain the expected complement of UNIX utilities, contributed by programmers around the world, including the set of tools developed as part of the GNU Project. The Linux community is committed to the continued development of this system. Support for new microcomputer devices and features is added soon after the hardware becomes available, and the tools available on Linux continue to be refined. Given the many commercial software packages available to run on Linux platforms and the many hardware manufacturers offering Linux on their systems, it is clear that the system has evolved well beyond its origin as an undergraduate project to become an operating system of choice for academic, commercial, professional, and personal use. Exercises 1. What is free software? List three characteristics of free software. 2. Why is Linux popular? Why is it popular in academia? 3. What are multiuser systems? Why are they successful? 4. What is the Free Software Foundation/GNU? What is Linux? Which parts of the Linux operating system did each provide? Who else has helped build and refine this operating system? 5. In which language is Linux written? What does the language have to do with the success of Linux? 6. What is a utility program? 7. What is a shell? How does it work with the kernel? With the user? Exercises 19 8. How can you use utility programs and a shell to create your own applications? 9. Why is the Linux filesystem referred to as hierarchical? 10. What is the difference between a multiprocessor and a multiprocessing system? 11. Give an example of when you would want to use a multiprocessing system. 12. Approximately how many people wrote Linux? Why is this project unique? 13. What are the key terms of the GNU General Public License? This page intentionally left blank I PART I The Linux and Mac OS X Operating Systems CHAPTER 2 Getting Started CHAPTER 3 The Utilities 45 CHAPTER 4 The Filesystem CHAPTER 5 The Shell 23 77 117 21 This page intentionally left blank 2 Getting Started In This Chapter Conventions Used in This Book . . . . 24 Logging In from a Terminal or Terminal Emulator . . . . 28 su/sudo: Curbing Your Power (root Privileges) . . . . 31 Where to Find Documentation . . . . . 33 The --help Option . . . . 33 man: Displays the System Manual . . . . 35 info: Displays Information About Utilities. . . . . 36 HOWTOs: Finding Out How Things Work . . . . 40 What to Do If You Cannot Log In. . . . 41 Changing Your Password . . . . 41 One way or another you are sitting in front of a screen that is connected to a computer that is running Linux. You may be working with a graphical user interface (GUI) or a textual interface. This book is about the textual interface, also called the command-line interface (CLI). If you are working with a GUI, you will need to use a terminal emulator such as xterm, Konsole, GNOME Terminal, Terminal (under Mac OS X), or a virtual console (page 40) to follow the examples in this book. 2Chapter2 This chapter starts with a discussion of the typographical conventions used in this book, followed by a section about logging in on the system. The next section introduces the shell and explains how to fix mistakes on the command line. Next come a brief reminder about the powers of working with root privileges and suggestions about how to avoid making mistakes that will make your system inoperable or hard to work with. The chapter continues with a discussion about where to find more information about Linux. It concludes with additional information on logging in, including how to change your password. 23 24 Chapter 2 Getting Started Be sure to read the warning about the dangers of misusing the powers of working with root privileges on page 31. While heeding that warning, feel free to experiment with the system: Give commands, create files, follow the examples in this book, and have fun. Conventions Used in This Book This book uses conventions to make its explanations shorter and clearer. The following paragraphs describe these conventions. Mac OS X The term Mac OS X refers to both Mac OS X and Mac OS X Server. This book points out important differences between the two. Mac OS X versions References to Mac OS X refer to version 10.5 (Leopard). Because the book focuses on the underlying operating system, which changes little from one release of OS X to the next, the text will remain relevant through several future releases. The author's Web site (www.sobell.com) provides corrections and updates as appropriate. Text and examples The text is set in this type, whereas examples are shown in a called a fixed-width font): monospaced font (also $ cat practice This is a small file I created with a text editor. Items you enter Everything you enter at the keyboard is shown in a bold typeface. Within the text, this bold typeface is used; within examples and screens, this one is used. In the previous example, the dollar sign ($) on the first line is a prompt that Linux displays, so it is not bold; the remainder of the first line is entered by a user, so it is bold. Utility names Names of utilities are printed in this sans serif typeface. This book references the emacs text editor and the ls utility or ls command (or just ls) but instructs you to enter ls –a on the command line. In this way the text distinguishes between utilities, which are programs, and the instructions you give on the command line to invoke the utilities. Filenames Filenames appear in a bold typeface. Examples are memo5, letter.1283, and reports. Filenames may include uppercase and lowercase letters; however, the popular Linux ext2, ext3, and ext4 filesystems are case sensitive (page 945), so memo5, MEMO5, and Memo5 refer to three different files. The default Mac OS X filesystem, HFS+, is not case sensitive; under OS X, memo5, MEMO5, and Memo5 refer to the same file. For more information refer to "Case Sensitivity" on page 927. Character strings Within the text, characters and character strings are marked by putting them in a bold typeface. This convention avoids the need for quotation marks or other delimiters before and after a string. An example is the following string, which is displayed by the passwd utility: Sorry, passwords do not match. Conventions Used in This Book 25 Keys and characters This book uses SMALL CAPS for three kinds of items: • Keyboard keys, such as the SPACE bar and the RETURN,1 ESCAPE, and TAB keys. • The characters that keys generate, such as the SPACEs generated by the SPACE bar. • Keyboard keys that you press with the CONTROL key, such as CONTROL-D. (Even though D is shown as an uppercase letter, you do not have to press the SHIFT key; enter CONTROL-D by holding the CONTROL key down and pressing d.) Prompts and RETURNs Most examples include the shell prompt—the signal that Linux is waiting for a command—as a dollar sign ($), a hashmark or pound sign (#), or sometimes a percent sign (%). The prompt does not appear in a bold typeface in this book because you do not enter it. Do not type the prompt on the keyboard when you are experimenting with examples from this book. If you do, the examples will not work. Examples omit the RETURN keystroke that you must use to execute them. An example of a command line is $ vim memo.1204 To use this example as a model for running the vim text editor, give the command vim memo.1204 and press the RETURN key. (Press ESCAPE ZZ to exit from vim; see page 151 for a vim tutorial.) This method of entering commands makes the examples in the book correspond to what appears on the screen. Definitions optional All glossary entries marked with FOLDOC are courtesy of Denis Howe, editor of the Free On-Line Dictionary of Computing (foldoc.org), and are used with permission. This site is an ongoing work containing definitions, anecdotes, and trivia. Optional Information Passages marked as optional appear in a gray box. This material is not central to the ideas presented in the chapter but often involves more challenging concepts. A good strategy when reading a chapter is to skip the optional sections and then return to them when you are comfortable with the main ideas presented in the chapter. This is an optional paragraph. URLs (Web addresses) Web addresses, or URLs, have an implicit http:// prefix, unless ftp:// or https:// is shown. You do not normally need to specify a prefix in a browser when the prefix is http://, but you must use a prefix when you specify an FTP or secure HTTP site. Thus you can specify a URL in a browser exactly as shown in this book. 1. Different keyboards use different keys to move the cursor (page 949) to the beginning of the next line. This book always refers to the key that ends a line as the RETURN key. Your keyboard may have a RET, NEWLINE, ENTER, RETURN , or other key. Use the corresponding key on your keyboard each time this book asks you to press RETURN. 26 Chapter 2 Getting Started Tip, caution, and security boxes The following boxes highlight information that may be helpful while you are using or administering a Linux system. This is a tip box tip A tip box may help you avoid repeating a common mistake or may point toward additional information. This box warns you about something caution A caution box warns you about a potential pitfall. This box marks a security note security A security box highlights a potential security issue. These notes are usually for system administrators, but some apply to all users. Logging In from a Terminal or Terminal Emulator Above the login prompt on a terminal, terminal emulator, or other textual device, many systems display a message called issue (stored in the /etc/issue file). This message usually identifies the version of Linux running on the system, the name of the system, and the device you are logging in on. A sample issue message follows: Ubuntu 9.10 tiny tty1 The issue message is followed by a prompt to log in. Enter your username and password in response to the system prompts. The following example shows Max logging in on the system named tiny: tiny login: max Password: Last login: Wed Mar 10 19:50:38 from dog max]$ If you are using a terminal (page 983) and the screen does not display the login: prompt, check whether the terminal is plugged in and turned on, and then press the RETURN key a few times. If login: still does not appear, try pressing CONTROL-Q (Xon). Did you log in last? security As you are logging in to a textual environment, after you enter your username and password, the system displays information about the last login on this account, showing when it took place and where it originated. You can use this information to determine whether anyone has accessed the account since you last used it. If someone has, perhaps an unauthorized user has learned your password and logged in as you. In the interest of maintaining security, advise the system administrator of any circumstances that make you suspicious and change your password. If you are using an Apple, PC, another Linux system, or a workstation (page 988), open the program that runs ssh (secure; page 828), telnet (not secure; page 852), or Logging In from a Terminal or Terminal Emulator 27 whichever communications/emulation software you use to log in on the system, and give it the name or IP address (page 960) of the system you want to log in on. Log in, making sure you enter your username and password as they were specified when your account was set up; the routine that verifies the username and password is case sensitive. Like most systems, Linux does not display your password when you enter it. By default Mac OS X does not allow remote logins (page 936). telnet is not secure security One of the reasons telnet is not secure is that it sends your username and password over the network in cleartext (page 947) when you log in, allowing someone to capture your login information and log in on your account. The ssh utility encrypts all information it sends over the network and, if available, is a better choice than telnet. The ssh program has been implemented on many operating systems, not just Linux. Many user interfaces to ssh include a terminal emulator. Following is an example of logging in using ssh from a Linux system: $ ssh 's password: Permission denied, please try again. 's password: Last login: Wed Mar 10 21:21:49 2005 from dog [ max]$ In the example Max mistyped his password, received an error message and another prompt, and retyped the password correctly. If your username is the same on the system you are logging in from and the system you are logging in on, you can omit your username and the following at sign (@). In the example, Max would have given the command ssh tiny. After you log in, the shell prompt (or just prompt) appears, indicating you have successfully logged in; it shows the system is ready for you to give a command. The first shell prompt may be preceded by a short message called the message of the day, or motd, which is stored in the /etc/motd file. The usual prompt is a dollar sign ($). Do not be concerned if you have a different prompt; the examples in this book will work regardless of what the prompt looks like. In the previous example, the $ prompt (last line) is preceded by the username (max), an at sign (@), the system name (tiny), and the name of the directory Max is working in (max). For information on how to change the prompt, refer to page 299 (bash) or page 373 (tcsh). Make sure TERM is set correctly tip The TERM shell variable establishes the pseudographical characteristics of a character-based terminal or terminal emulator. Typically TERM is set for you—you do not have to set it manually. If things on the screen do not look right, refer to "Specifying a Terminal" on page 906. 28 Chapter 2 Getting Started Working with the Shell Before the introduction of the graphical user interface (GUI), UNIX and then Linux provided only a textual interface (also called a command-line interface or CLI). Today, a textual interface is available when you log in from a terminal, a terminal emulator, or a textual virtual console, or when you use ssh or telnet to log in on a system. When you log in and are working in a textual (nongraphical) environment, and when you are using a terminal emulator window in a graphical environment, you are using the shell as a command interpreter. That is, the shell displays a prompt that indicates it is ready for you to give it a command. In response to the prompt, you type a command. The shell executes this command and then displays another prompt. Advantages of the textual interface Although the concept may seem antiquated, a textual interface has a place in modern computing. In some cases an administrator may use a command-line tool either because a graphical equivalent does not exist or because the graphical tool is not as powerful or flexible as the textual one. When logging in using a dial-up line or network connection, using a GUI may not be practical because of the slow connection speed. Frequently, on a server system, a graphical interface may not even be installed. The first reason for this omission is that a GUI consumes a lot of system resources; on a server, those resources are better dedicated to the main task of the server. Additionally, security mandates that a server system run as few tasks as possible because each additional task can make the system more vulnerable to attack. Also, some users prefer a textual interface because it is cleaner and more efficient than a GUI. Some simple tasks, such as changing file access permissions, can be done more easily from the command line (see chmod on page 626). Pseudographical interface Before the introduction of GUIs, resourceful programmers created textual interfaces that included graphical elements such as boxes, borders outlining rudimentary windows, highlights, and, more recently, color. These textual interfaces, called pseudographical interfaces, bridge the gap between textual and graphical interfaces. An example of a modern utility that uses a pseudographical interface is the dpkg-reconfigure utility, which reconfigures an installed software package. This section explains how to identify the shell you are using and describes the keystrokes you can use to correct mistakes on the command line. It covers how to abort a running command and briefly discusses how to edit a command line. Several chapters of this book are dedicated to shells: Chapter 5 introduces shells, Chapter 8 goes into more detail about the Bourne Again Shell with some coverage of the TC Shell, Chapter 9 covers the TC Shell exclusively, and Chapter 10 discusses writing programs (shell scripts) using the Bourne Again Shell. Which Shell Are You Running? This book discusses both the Bourne Again Shell (bash) and the TC Shell (tcsh). You are probably running bash, but you may be running tcsh or another shell such as the Z Shell (zsh). You can identify the shell you are running by using the ps utility (page 796). Type ps in response to the shell prompt and press RETURN. $ ps PID TTY 2402 pts/5 7174 pts/5 TIME CMD 00:00:00 bash 00:00:00 ps The command shows that you are running two utilities or commands: bash and ps. If you are running tcsh, ps will display tcsh instead of bash. If you are running a different shell, ps will display its name. Correcting Mistakes This section explains how to correct typographical and other errors you may make while you are logged in on a textual display. Because the shell and most other utilities do not interpret the command line or other text until after you press RETURN, you can readily correct typing mistakes before you press RETURN. You can correct typing mistakes in several ways: erase one character at a time, back up a word at a time, or back up to the beginning of the command line in one step. After you press RETURN, it is too late to correct a mistake: You must either wait for the command to run to completion or abort execution of the program (page 30). Erasing a Character While entering characters from the keyboard, you can back up and erase a mistake by pressing the erase key once for each character you want to delete. The erase key backs over as many characters as you wish. It does not, in general, back up past the beginning of the line. The default erase key is BACKSPACE. If this key does not work, try pressing DELETE or CONTROL-H. If these keys do not work, give the following stty command to set the erase and line kill (see "Deleting a Line" on the next page) keys to their default values: $ stty ek See page 838 for more information on stty. Deleting a Word You can delete a word you entered by pressing CONTROL-W. A word is any sequence of characters that does not contain a SPACE or TAB. When you press CONTROL-W, the cursor moves left to the beginning of the current word (as you are entering a word) or the previous word (when you have just entered a SPACE or TAB), removing the word. 2. The command stty is an abbreviation for set teletypewriter, the first terminal that UNIX was run on. Today stty is commonly thought of as meaning set terminal. 30 Chapter 2 Getting Started CONTROL-Z suspends a program tip Although it is not a way of correcting a mistake, you may press the suspend key (typically CONTROLZ) by mistake and wonder what happened. If the system displays a message containing the word Stopped, you have just stopped your job using job control. Give the command fg to continue your job in the foreground, and you should return to where you were before you pressed the suspend key. For more information refer to "Moving a Job from the Foreground to the Background" on page 135. Deleting a Line Any time before you press RETURN, you can delete the line you are entering by pressing the (line) kill key. When you press this key, the cursor moves to the left, erasing characters as it goes, back to the beginning of the line. The default line kill key is CONTROL-U. If this key does not work, try CONTROL-X. If these keys do not work, give the stty command described under "Erasing a Character." Aborting Execution Sometimes you may want to terminate a running program. For example, you may want to stop a program that is performing a lengthy task such as displaying the contents of a file that is several hundred pages long or copying a file that is not the one you meant to copy. To terminate a program from a textual display, press the interrupt key (CONTROL-C or sometimes DELETE or DEL). If you would like to use another key as your interrupt key, see "Special Keys and Characteristics" on page 838. If this method does not terminate the program, try stopping the program with the suspend key (typically CONTROL-Z), giving a jobs command to verify the number of the job running the program, and using kill to abort the job. The job number is the number within the brackets at the left end of the line that jobs displays ([1] in the next example). In the following example, the user uses the –TERM option to kill to send a termination signal to the job specified by the job number, which is preceded by a percent sign (%1). You can omit -TERM from the command as kill sends a termination signal by default. $ bigjob CONTROL-Z [1]+ Stopped $ jobs [1]+ Stopped $ kill -TERM %1 $ RETURN [1]+ Killed bigjob bigjob bigjob The kill command returns a prompt; press RETURN again to see the confirmation message. If the terminal interrupt signal does not appear to work, wait ten seconds and press RETURN. If there is no message indicating the job was killed, use the kill (-KILL) signal. A running program cannot usually ignore a kill signal; it is almost sure to abort the program. For more information refer to "Running a Command in the Background" on page 134 and kill on page 729. su/sudo: Curbing Your Power (root Privileges) 31 When you press the interrupt key, the Linux operating system sends a terminal interrupt signal to the program you are running and to the shell. Exactly what effect this signal has depends on the program. A program may stop execution and exit immediately (most common), it may ignore the signal, or it may receive the signal and take another action based on a custom signal handler procedure. The Bourne Again Shell has a custom signal handler procedure for the terminal interrupt signal. The behavior of this signal handler depends on whether the shell is waiting for the user to finish typing a command or waiting for a program to finish executing. If the shell is waiting for the user, the signal handler discards the partial line and displays a prompt. If the shell is waiting for a program to finish, the signal handler does nothing; the shell keeps waiting. When the program finishes executing, the shell displays a prompt. Repeating/Editing Command Lines To repeat a previous command under bash or tcsh, press the UP ARROW key. Each time you press this key, the shell displays an earlier command line. To reexecute the displayed command line, press RETURN. Press the DOWN ARROW key to browse through the command lines in the other direction. The RIGHT and LEFT ARROW keys move the cursor back and forth along the displayed command line. At any point along the command line, you can add characters by typing them. Use the erase key to remove characters from the command line. For information about more complex command-line editing, see page 310 (bash) and page 363 (tcsh). su/sudo: Curbing Your Power (root Privileges) UNIX and Linux systems have always had a privileged user named root. When you are working as the root user ("working with root privileges"), you have extraordinary systemwide powers. A user working with root privileges is sometimes referred to as Superuser or administrator. When working with root privileges, you can read from or write to any file on the system, execute programs that ordinary users cannot, and more. On a multiuser system you may not be permitted to gain root privileges and so may not be able to run certain programs. Nevertheless, someone—the system administrator—can, and that person maintains the system. Do not experiment while you are working with root privileges caution Feel free to experiment when you are not working with root privileges. When you are working with root privileges, do only what you have to do and make sure you know exactly what you are doing. After you have completed the task at hand, revert to working as yourself. When working with root privileges, you can damage the system to such an extent that you will need to reinstall Linux to get it working again. 32 Chapter 2 Getting Started Under a conventional setup, you can gain root privileges in one of two ways. First you can log in as the user named root; when you do so you are working with root privileges until you log off. Alternatively, while you are working as yourself, you can use the su (substitute user) utility to execute a single command with root privileges or to gain root privileges temporarily so you can execute several commands. Logging in as root and running su to gain root privileges require you to enter the root password. The following example shows how to use su to execute a single command. $ ls -l /lost+found ls: cannot open directory /lost+found: Permission denied $ su -c 'ls -l /lost+found' Password: Enter the root password total 0 $ The first command in the preceding example shows that a user who does not have root privileges is not permitted to list files in the /lost+found directory: ls displays an error message. The second command uses su with the –c (command) option to execute the same command with root privileges. Single quotation marks enclose the command to ensure the shell interprets the command properly. When the command finishes executing (ls shows there are no files in the directory), the user no longer has root privileges. Without any arguments, su spawns a new shell running with root privileges. Typically the shell displays a hash or pound sign (#) prompt when you are working with root privileges. Give an exit command to return to the normal prompt and nonroot privileges. $ su Password: # ls -l /lost+found total 0 # exit exit $ Enter the root password Some distributions (e.g., Ubuntu) ship with the root account locked—there is no root password—and rely on the sudo (www.sudo.ws) utility to allow users to gain root privileges. The sudo utility requires you to enter your password (not the root password) to gain root privileges. The following example allows the user to gain root privileges to view the contents of the /lost+found directory. $ sudo ls -l /lost+found [sudo] password for sam: total 0 $ Enter your password With an argument of –i, sudo spawns a new shell running with root privileges. Typically the shell displays a hash or pound sign (#) prompt when you are working with root privileges. Give an exit command to return to the normal prompt and nonroot privileges. $ sudo -i [sudo] password for sam: # ls -l /lost+found total 0 # exit logout $ Enter your password Where to Find Documentation 33 Where to Find Documentation Distributions of Linux do not typically come with hardcopy reference manuals. However, its online documentation has always been one of Linux's strengths. The man (or manual) and info pages have been available via the man and info utilities since early releases of the operating system. Not surprisingly, with the ongoing growth of Linux and the Internet, the sources of documentation have expanded as well. This section discusses some of the places you can look for information on Linux. See Appendix B as well. The --help Option Most GNU utilities provide a --help option that displays information about the utility. Non-GNU utilities may use a –h or –help option to display help information. $ cat --help Usage: cat [OPTION] [FILE]... Concatenate FILE(s), or standard input, to standard output. -A, --show-all -b, --number-nonblank -e -E, --show-ends ... equivalent to -vET number nonblank output lines equivalent to -vE display $ at end of each line If the information that --help displays runs off the screen, send the output through the less pager (page 34) using a pipe (page 56): $ ls --help | less man: Displays the System Manual The man utility (page 759) displays (man) pages from the system documentation in a textual environment. This documentation is helpful when you know which utility you want to use but have forgotten exactly how to use it. You can also refer to the man pages to get more information about specific topics or to determine which features are available with Linux. You can search for topics covered by man pages using the apropos utility (page 35). Because the descriptions in the system documentation are often terse, they are most helpful if you already understand the basic functions of a utility. 34 Chapter 2 Getting Started Figure 2-1 The man utility displaying information about itself Online man pages tip The tldp.org/manpages/man.html site holds links to copies of most man pages. In addition to presenting man pages in an easy-to-read HTML format, this site does not require you to install a utility to read the man page. less (pager) Manual sections To find out more about a utility, give the command man, followed by the name of the utility. Figure 2-1 shows man displaying information about itself; the user entered a man man command. The man utility automatically sends its output through a pager—usually less (page 48), which displays one screen at a time. When you access a manual page in this manner, less displays a prompt [e.g., Manual page man(1) line 1] at the bottom of the screen after it displays each screen of text and waits for you to request another screen of text by pressing the SPACE bar. Pressing h (help) displays a list of less commands. Pressing q (quit) stops less and causes the shell to display a prompt. For more information refer to page 48. Based on the FHS (Filesystem Hierarchy Standard; page 91), the Linux system manual and the man pages are divided into ten sections, where each section describes related tools: 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. User Commands System Calls Subroutines Devices File Formats Games Miscellaneous System Administration Kernel New Where to Find Documentation 35 This layout closely mimics the way the set of UNIX manuals has always been divided. Unless you specify a manual section, man displays the earliest occurrence in the manual of the word you specify on the command line. Most users find the information they need in sections 1, 6, and 7; programmers and system administrators frequently need to consult the other sections. In some cases the manual contains entries for different tools with the same name. For example, the following command displays the man page for the passwd utility from section 1 of the system manual: $ man passwd To see the man page for the passwd file from section 5, enter this command: $ man 5 passwd The preceding command instructs man to look only in section 5 for the man page. In documentation you may see this man page referred to as passwd(5). Use the –a option (see the adjacent tip) to view all man pages for a given subject (press qRETURN to display the next man page). For example, give the command man –a passwd to view all man pages for passwd. Options tip An option modifies the way a utility or command works. Options are usually specified as one or more letters preceded by one or two hyphens. An option typically appears following the name of the utility you are calling and a SPACE. Other arguments (page 941) to the command follow the option and a SPACE. For more information refer to "Options" on page 119. apropos: Searches for a Keyword When you do not know the name of the command you need to carry out a particular task, you can use apropos with a keyword to search for it. This utility searches for the keyword in the short description line (the top line) of all man pages and displays those that contain a match. The man utility, when called with the –k (keyword) option, provides the same output as apropos. The database apropos uses, named whatis, is not available on many Linux systems when they are first installed, but is built automatically by cron (see crontab on page 649 for a discussion of cron). The following example shows the output of apropos when you call it with the who keyword. The output includes the name of each command, the section of the manual that contains it, and the brief description from the top of the man page. This list includes the utility that you need (who) and identifies other, related tools that you might find useful: $ apropos who at.allow (5) at.deny (5) from (1) w (1) who (1) whoami (1) - determine who can submit jobs via at or batch - determine who can submit jobs via at or batch - print names of those who have sent mail - Show who is logged on and what they are doing. - show who is logged on - print effective userid batch doing. 36 Chapter 2 Getting Started Figure 2-2 whatis The initial screen info coreutils displays The whatis utility is similar to apropos but finds only complete word matches for the name of the utility: $ whatis who who (1) - show who is logged on info: Displays Information About Utilities The textual info utility is a menu-based hypertext system developed by the GNU project (page 3) and distributed with Linux. The info utility can display documentation on many Linux shells, utilities, and programs developed by the GNU project. See www.gnu.org/software/texinfo/manual/info for the info manual. Figure 2-2 shows the screen that info displays when you give the command info coreutils (the coreutils software package holds the Linux core utilities). man and info display different information tip The info utility displays more complete and up-to-date information on GNU utilities than does man. When a man page displays abbreviated information on a utility that is covered by info, the man page refers to info. The man utility frequently displays the only information available on non-GNU utilities. When info displays information on non-GNU utilities, it is frequently a copy of the man page. Because the information on this screen is drawn from an editable file, your display may differ from the screens shown in this section. You can press any of the following keys or key combinations while the initial info screen is displayed: • ? lists info commands • SPACE scrolls through the menu of items for which information is available • m followed by the name of a menu item displays that item • m followed by a SPACE displays a list of menus • q or CONTROL-C quits info Where to Find Documentation 37 Figure 2-3 The screen info coreutils displays after you type /sleepRETURN twice The notation info uses to describe keyboard keys may not be familiar to you. The notation C-h is the same as CONTROL-H. Similarly M-x means hold down the META or ALT key and press x. On some systems you need to press ESCAPE and then x to duplicate the function of META-X. For more information refer to "Keys: Notation and Use" on page 216. For Mac keyboards see "Activating the META key" on page 935. You may find pinfo easier to use than info tip The pinfo utility is similar to info but is more intuitive if you are not familiar with the emacs editor. This utility runs in a textual environment, as does info. When it is available, pinfo uses color to make its interface easier to use. You may have to install pinfo before you can use it; see Appendix C. After giving the command info coreutils, type /sleepRETURN to search for the string sleep. When you type /, the cursor moves to the bottom line of the screen and displays Search for string [string]: where string is the last string you searched for. Press RETURN to search for string or enter the string you want to search for. Typing sleep displays sleep on that line, and pressing RETURN displays the next occurrence of sleep. Next, type /RETURN (or /sleepRETURN) to search for the second occurrence of sleep as shown in Figure 2-3. The asterisk at the left end of the line indicates that this entry is a menu item. Following the asterisk is the name of the menu item and a description of the item. Each menu item is a link to the info page that describes the item. To jump to that page, use the ARROW keys to move the cursor to the line containing the menu item and press RETURN. Alternatively, you can type the name of the menu item in a menu command to view the information. To display information on sleep, for example, you can give the command m sleep, followed by RETURN. When you type m (for menu), the cursor moves to the bottom line of the window (as it did when you typed /) and displays Menu item:. Typing sleep displays sleep on that line, and pressing RETURN displays information about the menu item you have chosen. 38 Chapter 2 Getting Started Figure 2-4 The info page on the sleep utility Figure 2-4 shows the top node of information on sleep. A node groups a set of information you can scroll through with the SPACE bar. To display the next node, press n. Press p to display the previous node. (The sleep item has only one node.) As you read through this book and learn about new utilities, you can use man or info to find out more about those utilities. If you can print PostScript documents, you can print a manual page by using the man utility with the –t option. For example, man –t cat | lpr prints information about the cat utility. You can also use a Web browser to display the documentation at www.tldp.org and print the desired information from the browser. HOWTOs: Finding Out How Things Work A HOWTO document explains in detail how to do something related to Linux—from setting up a specialized piece of hardware to performing a system administration task to setting up specific networking software. Mini-HOWTOs offer shorter explanations. As with Linux software, one person or a few people generally are responsible for writing and maintaining a HOWTO document, but many people may contribute to it. The Linux Documentation Project (LDP; page 40) site houses most HOWTO and mini-HOWTO documents. Use a Web browser to visit www.tldp.org, click HOWTOs, and pick the index you want to use to find a HOWTO or mini-HOWTO. You can also use the LDP search feature on its home page to find HOWTOs and other documents. Getting Help with the System This section describes several places you can look for help in using a Linux system. Where to Find Documentation 39 Figure 2-5 Google reporting on an error message Finding Help Locally /usr/share/doc The /usr/share/doc and /usr/src/linux/Documentation (present only if you installed the kernel source code) directories often contain more detailed and different information about a utility than man or info provides. Frequently this information is meant for people who will be compiling and modifying the utility, not just using it. These directories hold thousands of files, each containing information on a separate topic. For example, the following commands display information on grep and info. The second command uses zcat (page 618) to allow you to view a compressed file by decompressing it and sending it through the less pager (page 34). $ less /usr/share/doc/grep/README $ zcat /usr/share/doc/info/README.gz | less Using the Internet to Get Help The Internet provides many helpful sites related to Linux. Aside from sites that offer various forms of documentation, you can enter an error message from a program you are having a problem with in a search engine such as Google (www.google.com, or its Linux-specific version at www.google.com/linux). Enclose the error message within double quotation marks to improve the quality of the results. The search will likely yield a post concerning your problem and suggestions about how to solve it. See Figure 2-5. GNU GNU manuals are available at www.gnu.org/manual. In addition, you can visit the GNU home page (www.gnu.org) to obtain other documentation and GNU resources. Many of the GNU pages and resources are available in a variety of languages. 40 Chapter 2 Getting Started Figure 2-6 The Linux Documentation Project home page The Linux Documentation Project The Linux Documentation Project (www.tldp.org; Figure 2-6), which has been around for almost as long as Linux, houses a complete collection of guides, HOWTOs, FAQs, man pages, and Linux magazines. The home page is available in English, Portuguese, Spanish, Italian, Korean, and French. It is easy to use and supports local text searches. It also provides a complete set of links you can use to find almost anything you want related to Linux (click Links in the Search box or go to www.tldp.org/links). The links page includes sections on general information, events, getting started, user groups, mailing lists, and newsgroups, with each section containing many subsections. More About Logging In This section covers how to use virtual consoles, what to do if you have a problem logging in, and how to change your password. Using Virtual Consoles When running Linux on a personal computer, you will frequently work with the display and keyboard attached to the computer. Using this physical console, you can access as many as 63 virtual consoles (also called virtual terminals). Some are set up to allow logins; others act as graphical displays. To switch between virtual consoles, hold the CONTROL and ALT keys down and press the function key that corresponds to the console you want to view. For example, CONTROL-ALT-F5 displays the fifth virtual console. This book refers to the console you see when you press CONTROL-ALT-F1 as the system console, or just console. More About Logging In 41 Typically, six virtual consoles are active and have textual login prompts. When you want to use both textual and graphical interfaces, you can set up a textual session on one virtual console and a graphical session on another. No matter which virtual console you start a graphical session from, the graphical session runs on the first unused virtual console (typically number seven). What to Do If You Cannot Log In If you enter either your username or your password incorrectly, the system displays an error message after you enter both your username and your password. This message indicates you have entered either the username or the password incorrectly or they are not valid. It does not differentiate between an unacceptable username and an unacceptable password—a strategy meant to discourage unauthorized people from guessing names and passwords to gain access to the system. Following are some common reasons why logins fail: • The username and password are case sensitive. Make sure the CAPS LOCK key is off and enter your username and password exactly as specified or as you set them up. • You are not logging in on the right machine. The login/password combination may not be valid if you are trying to log in on the wrong machine. On a larger, networked system, you may have to specify the machine you want to connect to before you can log in. • Your username is not valid. The login/password combination may not be valid if you have not been set up as a user. Check with the system administrator. Logging Out To log out from a character-based interface, press CONTROL-D in response to the shell prompt. This action sends the shell an EOF (end of file) signal. Alternatively, you can give the command exit. Exiting from a shell does not end a graphical session; it just exits from the shell you are working with.

For example, exiting from the shell that GNOME terminal provides closes the GNOME terminal window. Changing Your Password If someone else assigned you a password, it is a good idea to give yourself a new one. For security reasons none of the passwords you enter is displayed by any utility. Protect your password security Do not allow someone to find out your password: Do not put your password in a file that is not encrypted, allow someone to watch you type your password, or give your password to someone you do not know (a system administrator never needs to know your password). You can always write your password down and keep it in a safe, private place. 42 Chapter 2 Getting Started Choose a password that is difficult to guess security Do not use phone numbers, names of pets or kids, birthdays, words from a dictionary (not even a foreign language), and so forth. Do not use permutations of these items or a l33t-speak variation of a word: Modern dictionary crackers may also try these permutations. Differentiate between important and less important passwords security It is a good idea to differentiate between important and less important passwords. For example, Web site passwords for blogs or download access are not very important; it is acceptable to use the same password for these types of sites. However, your login, mail server, and bank account Web site passwords are critical: Use different passwords for each and never use these passwords for an unimportant Web site. To change your password, give the command passwd. The first item passwd asks for is your current (old) password. This password is verified to ensure that an unauthorized user is not trying to alter your password. Then the system requests a new password. pwgen can help you pick a password security The pwgen utility (which you may need to install; see Appendix C) generates a list of almost random passwords. With a little imagination, you can pronounce, and therefore remember, some of these passwords. After you enter your new password, the system asks you to retype it to make sure you did not make a mistake when you entered it the first time. If the new password is the same both times you enter it, your password is changed. If the passwords differ, it means that you made an error in one of them, and the system displays this error message: Sorry, passwords do not match If your password is not long enough, the system displays the following message: You must choose a longer password When it is too simple, the system displays this message: Bad: new password is too simple If the system displays an error message, you need to start over. Press times until the shell displays a prompt and run passwd again. RETURN a few When you successfully change your password, you change the way you log in. If you forget your password, someone working with root privileges can run passwd to change it and tell you your new password. Under OS X, the passwd utility changes your login password, but does not change your Keychain password. The Keychain password is used by various graphical applications. You can change the Keychain password using the Keychain Access application. Chapter Summary Secure passwords 43 To be relatively secure, a password should contain a combination of numbers, uppercase and lowercase letters, and punctuation characters and meet the following criteria: • Must be at least four to six or more characters long (depending on how the administrator sets up the system). Seven or eight characters is a good compromise between length and security. • Should not be a word in a dictionary of any language, no matter how seemingly obscure. • Should not be the name of a person, place, pet, or other thing that might be discovered easily. • Should contain at least two letters and one digit or punctuation character. • Should not be your username, the reverse of your username, or your username shifted by one or more characters. Only the first item is mandatory. Avoid using control characters (such as CONTROL-H) because they may have a special meaning to the system, making it impossible for you to log in. If you are changing your password, the new password should differ from the old one by at least three characters. Changing the case of a character does not make it count as a different character. Chapter Summary As with many operating systems, your access to a Linux system is authorized when you log in. You enter your username in response to the login: prompt, followed by a password. You can use passwd to change your password any time while you are logged in. Choose a password that is difficult to guess and that conforms to the criteria imposed by the system administrator. The system administrator is responsible for maintaining the system. On a single-user system, you are the system administrator. On a small, multiuser system, you or another user will act as the system administrator, or this job may be shared. On a large, multiuser system or network of systems, there is frequently a full-time system administrator. When extra privileges are required to perform certain system tasks, the system administrator gains root privileges by logging in as root, or running su or sudo. On a multiuser system, several trusted users may be allowed to gain root privileges. Do not work with root privileges as a matter of course. When you have to do something that requires root privileges, work with root privileges for only as long as you need to; revert to working as yourself as soon as possible. The man utility provides online documentation on system utilities. This utility is helpful both to new Linux users and to experienced users who must often delve into the system documentation for information on the fine points of a utility's behavior. The info utility helps the beginner and the expert alike. It includes documentation on many Linux utilities. 44 Chapter 2 Getting Started Exercises 1. The following message is displayed when you attempt to log in with an incorrect username or an incorrect password: Login incorrect This message does not indicate whether your username, your password, or both are invalid. Why does it not tell you this information? 2. Give three examples of poor password choices. What is wrong with each? Include one that is too short. Give the error message the system displays. 3. Is fido an acceptable password? Give several reasons why or why not. 4. What would you do if you could not log in? 5. Try to change your password to dog. What happens? Now change it to a more secure password. What makes that password relatively secure? Advanced Exercises 6. Change your login shell to tcsh without using root privileges. 7. How many man pages are in the Devices subsection of the system manual? (Hint: Devices is a subsection of Special Files.) 8. The example on page 35 shows that man pages for passwd appear in sections 1 and 5 of the system manual. Explain how you can use man to determine which sections of the system manual contain a manual page with a given name. 9. How would you find out which Linux utilities create and work with archive files? 3 The Utilities In This Chapter When Linus Torvalds introduced Linux and for a long time thereafter, Linux did not have a graphical user interface (GUI): It ran using a textual interface, also referred to as a command-line interface (CLI). All the tools ran from a command line. Today the Linux GUI is important but many people—especially system administrators—run many command-line utilities. Commandline utilities are often faster, more powerful, or more complete than their GUI counterparts. Sometimes there is no GUI counterpart to a textual utility; some people just prefer the hands-on feeling of the command line. 3Chapter3 When you work with a command-line interface, you are working with a shell (Chapters 5, 8, and 10). Before you start working with a shell, it is important that you understand something about the characters that are special to the shell, so this chapter starts with a discussion of special characters. The chapter then describes five basic utilities: ls, cat, rm, less, and hostname. It continues by describing several other file manipulation utilities as well as utilities that display who is logged in; that communicate with other users; that print, compress, and decompress files; and that pack and unpack archive files. 45 46 Chapter 3 The Utilities Special Characters Special characters, which have a special meaning to the shell, are discussed in "Filename Generation/Pathname Expansion" on page 136. These characters are mentioned here so you can avoid accidentally using them as regular characters until you understand how the shell interprets them. For example, it is best to avoid using any of the following characters in a filename (even though emacs and some other programs do) because they make the file harder to reference on the command line: & ; | * ? ' " ' [ ] ( ) $ < > { } # / \ ! ~ Whitespace Although not considered special characters, RETURN, SPACE, and TAB have special meanings to the shell. RETURN usually ends a command line and initiates execution of a command. The SPACE and TAB characters separate elements on the command line and are collectively known as whitespace or blanks. Quoting special characters If you need to use a character that has a special meaning to the shell as a regular character, you can quote (or escape) it. When you quote a special character, you keep the shell from giving it special meaning. The shell treats a quoted special character as a regular character. However, a slash (/) is always a separator in a pathname, even when you quote it. Backslash To quote a character, precede it with a backslash (\). When two or more special characters appear together, you must precede each with a backslash (for example, you would enter ** as **). You can quote a backslash just as you would quote any other special character—by preceding it with a backslash (\\). Single quotation marks Another way of quoting special characters is to enclose them between single quotation marks: '**'. You can quote many special and regular characters between a pair of single quotation marks: 'This is a special character: >'. The regular characters are interpreted as usual, and the shell also interprets the special characters as regular characters. The only way to quote the erase character (CONTROL-H), the line kill character (CONTROL-U), and other control characters (try CONTROL-M) is by preceding each with a CONTROL-V. Single quotation marks and backslashes do not work. Try the following: $ echo 'xxxxxxCONTROL-U' $ echo xxxxxxCONTROL-V CONTROL-U optional Although you cannot see the CONTROL-U displayed by the second of the preceding pair of commands, it is there. The following command sends the output of echo (page 57) through a pipe (page 56) to od (octal display; page 776) to display CONTROL-U as octal 25 (025): $ echo xxxxxxCONTROL-V CONTROL-U | od -c 0000000 x x x x x x 025 0000010 The is the NEWLINE character that echo sends at the end of its output. Basic Utilities 47 Basic Utilities One of the important advantages of Linux is that it comes with thousands of utilities that perform myriad functions. You will use utilities whenever you work with Linux, whether you use them directly by name from the command line or indirectly from a menu or icon. The following sections discuss some of the most basic and important utilities; these utilities are available from a textual interface. Some of the more important utilities are also available from a GUI; others are available only from a GUI. Run these utilities from a command line tip This chapter describes command-line, or textual, utilities. You can experiment with these utilities from a terminal, a terminal emulator within a GUI, or a virtual console (page 40). Folder/directory The term directory is used extensively in the next sections. A directory is a resource that can hold files. On other operating systems, including Windows and Mac OS X, and frequently when speaking about a Linux GUI, a directory is referred to as a folder. That is a good analogy: A traditional manila folder holds files just as a directory does. In this chapter you work in your home directory tip When you log in on the system, you are working in your home directory. In this chapter that is the only directory you use: All the files you create in this chapter are in your home directory. Chapter 4 goes into detail about directories. ls: Lists the Names of Files Using the editor of your choice, create a small file named practice. (A tutorial on the vim editor appears on page 151 and a tutorial on emacs appears on page 208.) After exiting from the editor, you can use the ls (list) utility to display a list of the names of the files in your home directory. In the first command in Figure 3-1, ls lists the name of the practice file. (You may also see files that the system or a program created automatically.) Subsequent commands in Figure 3-1 display the contents of the file and remove the file. These commands are described next. Refer to page 745 or give the command info coreutils 'ls invocation' for more information. $ ls practice $ cat practice This is a small file that I created with a text editor. $ rm practice $ ls $ cat practice cat: practice: No such file or directory $ Figure 3-1 Using ls, cat, and rm on the file named practice 48 Chapter 3 The Utilities cat: Displays a Text File The cat utility displays the contents of a text file. The name of the command is derived from catenate, which means to join together, one after the other. (Figure 5-8 on page 127 shows how to use cat to string together the contents of three files.) A convenient way to display the contents of a file to the screen is by giving the command cat, followed by a SPACE and the name of the file. Figure 3-1 shows cat displaying the contents of practice. This figure shows the difference between the ls and cat utilities: The ls utility displays the name of a file, whereas cat displays the contents of a file. Refer to page 618 or give the command info coreutils 'cat invocation' for more information. rm: Deletes a File The rm (remove) utility deletes a file. Figure 3-1 shows rm deleting the file named practice. After rm deletes the file, ls and cat show that practice is no longer in the directory. The ls utility does not list its filename, and cat says that no such file exists. Use rm carefully. Refer to page 804 or give the command info coreutils 'rm invocation' for more information. If you are running Mac OS X, see "Many Utilities Do Not Respect Apple Human Interface Guidelines" on page 936. A safer way of removing files tip You can use the interactive form of rm to make sure that you delete only the file(s) you intend to delete. When you follow rm with the –i option (see page 35 for a tip on options) and the name of the file you want to delete, rm displays the name of the file and then waits for you to respond with y (yes) before it deletes the file. It does not delete the file if you respond with a string that begins with a character other than y. $ rm -i toollist rm: remove regular file 'toollist'? y Optional: You can create an alias (page 324) for rm –i and put it in your startup file (page 82) so rm always runs in interactive mode. less Is more: Display a Text File One Screen at a Time Pagers When you want to view a file that is longer than one screen, you can use either the less utility or the more utility. Each of these utilities pauses after displaying a screen of text; press the SPACE bar to display the next screen of text. Because these utilities show one page at a time, they are called pagers. Although less and more are very similar, they have subtle differences. At the end of the file, for example, less displays an END message and waits for you to press q before returning you to the shell. In contrast, more returns you directly to the shell. While using both utilities you can press h to display a Help screen that lists commands you can use while paging through a file. Give the commands less practice and more practice in place of the cat command in Figure 3-1 to see how these commands work. Use the command less /usr/share/dict/words instead if you want to experiment with a longer file. Refer to page 735 or to the less and more man pages for more information on less and more. Working with Files 49 hostname: Displays the System Name The hostname utility displays the name of the system you are working on. Use this utility if you are not sure you are logged in on the right machine. Refer to the hostname man page for more information. $ hostname dog Working with Files This section describes utilities that copy, move, print, search through, display, sort, and compare files. If you are running Mac OS X, see "Resource forks" on page 929. Filename completion tip After you enter one or more letters of a filename (following a command) on a command line, press TAB and the Bourne Again Shell will complete as much of the filename as it can. When only one filename starts with the characters you entered, the shell completes the filename and places a SPACE after it. You can keep typing or you can press RETURN to execute the command at this point. When the characters you entered do not uniquely identify a filename, the shell completes what it can and waits for more input. When pressing TAB does not change the display, press TAB again (Bourne Again Shell, page 320) or CONTROL-D (TC Shell, page 360) to display a list of possible completions. cp: Copies a File The cp (copy) utility (Figure 3-2) makes a copy of a file. This utility can copy any file, including text and executable program (binary) files. You can use cp to make a backup copy of a file or a copy to experiment with. The cp command line uses the following syntax to specify source and destination files: cp source-file destination-file The source-file is the name of the file that cp will copy. The destination-file is the name that cp assigns to the resulting (new) copy of the file. $ ls memo $ cp memo memo.copy $ ls memo memo.copy Figure 3-2 cp copies a file 50 Chapter 3 The Utilities The cp command line in Figure 3-2 copies the file named memo to memo.copy. The period is part of the filename—just another character. The initial ls command shows that memo is the only file in the directory. After the cp command, a second ls shows two files in the directory, memo and memo.copy. Sometimes it is useful to incorporate the date in the name of a copy of a file. The following example includes the date January 30 (0130) in the copied file: $ cp memo memo.0130 Although it has no significance to Linux, the date can help you find a version of a file you created on a certain date. Including the date can also help you avoid overwriting existing files by providing a unique filename each day. For more information refer to "Filenames" on page 79. Refer to page 640 or give the command info coreutils 'cp invocation' for more information. Use scp (page 810), ftp (page 704), or rsync (page 583) when you need to copy a file from one system to another on a common network. cp can destroy a file caution If the destination-file exists before you give a cp command, cp overwrites it. Because cp overwrites (and destroys the contents of) an existing destination-file without warning, you must take care not to cause cp to overwrite a file that you need. The cp –i (interactive) option prompts you before it overwrites a file. See page 35 for a tip on options. The following example assumes that the file named orange.2 exists before you give the cp command. The user answers y to overwrite the file: $ cp –i orange orange.2 cp: overwrite 'orange.2'? y mv: Changes the Name of a File The mv (move) utility can rename a file without making a copy of it. The mv command line specifies an existing file and a new filename using the same syntax as cp: mv existing-filename new-filename The command line in Figure 3-3 changes the name of the file memo to memo.0130. The initial ls command shows that memo is the only file in the directory. After you give the mv command, memo.0130 is the only file in the directory. Compare this result to that of the cp example in Figure 3-2. The mv utility can be used for more than changing the name of a file. Refer to "mv, cp: Move or Copy Files" on page 90. Also refer to page 771 or give the command info coreutils 'mv invocation' for more information. mv can destroy a file caution Just as cp can destroy a file, so can mv. Also like cp, mv has a –i (interactive) option. See the caution box labeled "cp can destroy a file." Working with Files 51 $ ls memo $ mv memo memo.0130 $ ls memo.0130 Figure 3-3 mv renames a file lpr: Prints a File The lpr (line printer) utility places one or more files in a print queue for printing. Linux provides print queues so that only one job is printed on a given printer at a time. A queue allows several people or jobs to send output simultaneously to a single printer with the expected results. On systems that have access to more than one printer, you can use lpstat –p to display a list of available printers. Use the –P option to instruct lpr to place the file in the queue for a specific printer—even one that is connected to another system on the network. The following command prints the file named report: $ lpr report Because this command does not specify a printer, the output goes to the default printer, which is the printer when you have only one printer. The next command line prints the same file on the printer named mailroom: $ lpr -P mailroom report You can see which jobs are in the print queue by giving an lpstat –o command or by using the lpq utility: $ lpq lp is ready and printing Rank Owner Job Files active max 86 (standard input) Total Size 954061 bytes In this example, Max has one job that is being printed; no other jobs are in the queue. You can use the job number (86 in this case) with the lprm utility to remove the job from the print queue and stop it from printing: $ lprm 86 You can send more than one file to the printer with a single command. The following command line prints three files on the printer named laser1: $ lpr -P laser1 05.txt 108.txt 12.txt Refer to pages 132 and 742 or to the lpr man page for more information. 52 Chapter 3 The Utilities $ cat memo Helen: In our meeting on June 6 we discussed the issue of credit. Have you had any further thoughts about it? Max $ grep 'credit' memo discussed the issue of credit. Figure 3-4 grep searches for a string grep: Searches for a String The grep1 utility searches through one or more files to see whether any contain a specified string of characters. This utility does not change the file it searches but simply displays each line that contains the string. The grep command in Figure 3-4 searches through the file memo for lines that contain the string credit and displays the single line that meets this criterion. If memo contained such words as discredit, creditor, or accreditation, grep would have displayed those lines as well because they contain the string it was searching for. The –w (words) option causes grep to match only whole words. Although you do not need to enclose the string you are searching for in single quotation marks, doing so allows you to put SPACEs and special characters in the search string. The grep utility can do much more than search for a simple string in a single file. Refer to page 719 or to the grep man page for more information. See also Appendix A, "Regular Expressions." head: Displays the Beginning of a File By default the head utility displays the first ten lines of a file. You can use head to help you remember what a particular file contains. For example, if you have a file named months that lists the 12 months of the year in calendar order, one to a line, then head displays Jan through Oct (Figure 3-5). This utility can display any number of lines, so you can use it to look at only the first line of a file, at a full screen, or even more. To specify the number of lines 1. Originally the name grep was a play on an ed—an original UNIX editor, available on Linux— command: g/re/p. In this command g stands for global, re is a regular expression delimited by slashes, and p means print. Working with Files 53 $ head months Jan Feb Mar Apr May Jun Jul Aug Sep Oct $ tail -5 months Aug Sep Oct Nov Dec Figure 3-5 head displays the first lines of a file; tail displays the last lines of a file displayed, include a hyphen followed by the number of lines you want head to display. For example, the following command displays only the first line of months: $ head -1 months Jan The head utility can also display parts of a file based on a count of blocks or characters rather than lines. Refer to page 727 or give the command info coreutils 'head invocation' for more information. tail: Displays the End of a File The tail utility is similar to head but by default displays the last ten lines of a file. Depending on how you invoke it, this utility can display fewer or more than ten lines, use a count of blocks or characters rather than lines to display parts of a file, and display lines being added to a file that is changing. The tail command in Figure 3-5 displays the last five lines (Aug through Dec) of the months file. You can monitor lines as they are added to the end of the growing file named logfile with the following command: $ tail -f logfile Press the interrupt key (usually CONTROL-C) to stop tail and display the shell prompt. Refer to page 843 or give the command info coreutils 'tail invocation' for more information. 54 Chapter 3 The Utilities $ cat days Monday Tuesday Wednesday Thursday Friday Saturday Sunday $ sort days Friday Monday Saturday Sunday Thursday Tuesday Wednesday Figure 3-6 sort displays the lines of a file in order sort: Displays a File in Order The sort utility displays the contents of a file in order by lines; it does not change the original file. Figure 3-6 shows cat displaying the file named days, which contains the name of each day of the week on a separate line in calendar order. The sort utility then displays the file in alphabetical order. The sort utility is useful for putting lists in order. The –u option generates a sorted list in which each line is unique (no duplicates). The –n option puts a list of numbers in numerical order. Refer to page 817 or give the command info coreutils 'sort invocation' for more information. uniq: Removes Duplicate Lines from a File The uniq (unique) utility displays a file, skipping adjacent duplicate lines, but does not change the original file. If a file contains a list of names and has two successive entries for the same person, uniq skips the duplicate line (Figure 3-7). If a file is sorted before it is processed by uniq, this utility ensures that no two lines in the file are the same. (Of course, sort can do that all by itself with the –u option.) Refer to page 872 or give the command info coreutils 'uniq invocation' for more information. diff: Compares Two Files The diff (difference) utility compares two files and displays a list of the differences between them. This utility does not change either file; it is useful when you want to compare two versions of a letter or a report or two versions of the source code for a program. The diff utility with the –u (unified output format) option first displays two lines indicating which of the files you are comparing will be denoted by a plus sign (+) Working with Files 55 $ cat dups Cathy Fred Joe John Mary Mary Paula $ uniq dups Cathy Fred Joe John Mary Paula Figure 3-7 uniq removes duplicate lines and which by a minus sign (–). In Figure 3-8, a minus sign indicates the colors.1 file; a plus sign indicates the colors.2 file. The diff –u command breaks long, multiline text into hunks. Each hunk is preceded by a line starting and ending with two at signs (@@). This hunk identifier indicates the starting line number and the number of lines from each file for this hunk. In Figure 3-8, the hunk covers the section of the colors.1 file (indicated by a minus sign) from the first line through the sixth line. The +1,5 then indicates that the hunk covers colors.2 from the first line through the fifth line. Following these header lines, diff –u displays each line of text with a leading minus sign, a leading plus sign, or a SPACE. A leading minus sign indicates that the line occurs only in the file denoted by the minus sign. A leading plus sign indicates that the line occurs only in the file denoted by the plus sign. A line that begins with a SPACE (neither a plus sign nor a minus sign) occurs in both files in the same location. Refer to page 663 or to the diff info page for more information. $ diff -u colors.1 colors.2 --- colors.1 2009-07-29 16:41:11.000000000 -0700 +++ colors.2 2009-07-29 16:41:17.000000000 -0700 @@ -1,6 +1,5 @@ red +blue green yellow -pink -purple orange Figure 3-8 diff displaying the unified output format 56 Chapter 3 The Utilities file: Identifies the Contents of a File You can use the file utility to learn about the contents of a file without having to open and examine the file yourself. In the following example, file reports that letter_e.bz2 contains data that was compressed by the bzip2 utility (page 60): $ file letter_e.bz2 letter_e.bz2: bzip2 compressed data, block size = 900k Next file reports on two more files: $ file memo zach.jpg memo: ASCII text zach.jpg: JPEG image data, ... resolution (DPI), 72 x 72 Refer to page 686 or to the file man page for more information. | (Pipe): Communicates Between Processes Because pipes are integral to the functioning of a Linux system, this chapter introduces them for use in examples. Pipes are covered in detail beginning on page 131. If you are running Mac OS X, see the tip "Pipes do not work with resource forks" on page 929. A process is the execution of a command by Linux (page 306). Communication between processes is one of the hallmarks of both UNIX and Linux. A pipe (written as a vertical bar [|] on the command line and appearing as a solid or broken vertical line on a keyboard) provides the simplest form of this kind of communication. Simply put, a pipe takes the output of one utility and sends that output as input to another utility. Using UNIX/Linux terminology, a pipe takes standard output of one process and redirects it to become standard input of another process. (For more information refer to "Standard Input and Standard Output" on page 123.) Most of what a process displays on the screen is sent to standard output. If you do not redirect it, this output appears on the screen. Using a pipe, you can redirect standard output so it becomes standard input of another utility. For example, a utility such as head can take its input from a file whose name you specify on the command line following the word head, or it can take its input from standard input. The following command line sorts the lines of the months file (Figure 3-5, page 53) and uses head to display the first four months of the sorted list: $ sort months | head -4 Apr Aug Dec Feb The next command line displays the number of files in a directory (assuming no filenames include SPACEs). The wc (word count) utility with the –w (words) option displays the number of words in its standard input or in a file you specify on the command line: $ ls | wc -w 14 Four More Utilities 57 $ ls memo memo.0714 practice $ echo Hi Hi $ echo This is a sentence. This is a sentence. $ echo star: * star: memo memo.0714 practice $ Figure 3-9 echo copies the command line (but not the word echo) to the screen You can use a pipe to send output of a program to the printer: $ tail months | lpr Four More Utilities The echo and date utilities are two of the most frequently used members of the large collection of Linux utilities. The script utility records part of a session in a file, and todos or unix2dos makes a copy of a text file that can be read on a machine running Windows or Mac OS X. echo: Displays Text The echo utility copies the characters you type on the command line after echo to the screen. Figure 3-9 shows some examples. The last example shows how the shell treats an unquoted asterisk (*) on the command line: It expands the asterisk into a list of filenames in the directory. The echo utility is a good tool for learning about the shell and other Linux utilities. Some examples on page 138 use echo to illustrate how special characters, such as the asterisk, work. Throughout Chapters 5, 8, and 10, echo helps explain how shell variables work and how you can send messages from shell scripts to the screen. Refer to page 680 or give the command info coreutils 'echo invocation' for more information. See the bash and tcsh man pages for information about the versions of echo that are built into those shells. optional You can use echo to create a file by redirecting its output to a file: $ echo 'My new file.' > myfile $ cat myfile My new file. The greater than (>) sign tells the shell to send the output of echo to the file named myfile instead of to the screen. For more information refer to "Redirecting Standard Output" on page 126. 58 Chapter 3 The Utilities date: Displays the Time and Date The date utility displays the current date and time: $ date Wed Mar 18 17:12:20 PDT 2009 The following example shows how you can choose the format and select the contents of the output of date: $ date +"%A %B %d" Wednesday March 18 Refer to page 655 or give the command info coreutils 'date invocation' for more information. script: Records a Shell Session The script utility records all or part of a login session, including your input and the system's responses. This utility is useful only from character-based devices, such as a terminal or a terminal emulator. It does capture a session with vim; however, because vim uses control characters to position the cursor and display different typefaces, such as bold, the output will be difficult to read and may not be useful. When you cat a file that has captured a vim session, the session quickly passes before your eyes. By default script captures the session in a file named typescript. To specify a different filename, follow the script command with a SPACE and the filename. To append to a file, use the –a option after script but before the filename; otherwise script overwrites an existing file. Following is a session being recorded by script: $ script Script started, file is typescript $ whoami sam $ ls -l /bin | head -5 total 5024 -rwxr-xr-x 1 root root 2928 Sep -rwxr-xr-x 1 root root 1054 Apr -rwxr-xr-x 1 root root 7168 Sep -rwxr-xr-x 1 root root 701008 Aug $ exit exit Script done, file is typescript 21 26 21 27 21:42 15:37 19:18 02:41 archdetect autopartition autopartition-loop bash Use the exit command to terminate a script session. You can then view the file you created using cat, less, more, or an editor. Following is the file that was created by the preceding script command: $ cat typescript Script started on Thu Sep 24 20:54:59 2009 $ whoami sam Four More Utilities 59 $ ls -l /bin total 5024 -rwxr-xr-x 1 -rwxr-xr-x 1 -rwxr-xr-x 1 -rwxr-xr-x 1 $ exit exit | head -5 root root root root root root root root 2928 1054 7168 701008 Sep Apr Sep Aug 21 26 21 27 21:42 15:37 19:18 02:41 archdetect autopartition autopartition-loop bash Script done on Thu Sep 24 20:55:29 2009 If you will be editing the file with vim, emacs, or another editor, you can use fromdos or dos2unix (both below) to eliminate from the typescript file the ^M characters that appear at the ends of the lines. Refer to the script man page for more information. todos/unix2dos: Converts Linux and Mac OS X Files to Windows Format If you want to share a text file you created on a Linux system with someone on a system running Windows or Mac OS X, you need to convert the file before the person on the other system can read it easily. The todos (to DOS; part of the tofrodos package) or unix2dos (UNIX to DOS; part of the unix2dos package) utility converts a Linux text file so it can be read on a Windows or OS X system. Give the following command to convert a file named memo.txt (created with a text editor) to a DOSformat file: $ todos memo.txt or $ unix2dos memo.txt You can now email the file as an attachment to someone on a Windows or OS X system. Without any options, todos overwrites the original file. Use the –b (backup) option to cause todos to make a copy of the file with a .bak filename extension before modifying it. Use the –n (new) option to cause unix2dos to write the modified file to a new file as specified by a second argument (unix2dos old new). fromdos/dos2unix You can use the fromdos (from DOS; part of the tofrodos package) or dos2unix (DOS to UNIX; part of the dos2unix package) utility to convert Windows or OS X files so they can be read on a Linux system: $ fromdos memo.txt or $ dos2unix memo.txt See the todos and fromdos or unix2dos and dos2unix man pages for more information. 60 Chapter 3 The Utilities optional tr You can also use tr (translate; page 864) to change a Windows or OS X text file into a Linux text file. In the following example, the –d (delete) option causes tr to remove RETURNs (represented by \r) as it makes a copy of the file: $ cat memo | tr -d '\r' > memo.txt The greater than (>) symbol redirects the standard output of tr to the file named memo.txt. For more information refer to "Redirecting Standard Output" on page 126. Converting a file the other way without using todos or unix2dos is not as easy. Compressing and Archiving Files Large files use a lot of disk space and take longer than smaller files to transfer from one system to another over a network. If you do not need to look at the contents of a large file often, you may want to save it on a CD, DVD, or another medium and remove it from the hard disk. If you have a continuing need for the file, retrieving a copy from another medium may be inconvenient. To reduce the amount of disk space a file occupies without removing the file, you can compress the file without losing any of the information it holds. Similarly a single archive of several files packed into a larger file is easier to manipulate, upload, download, and email than multiple files. You may download compressed, archived files from the Internet. The utilities described in this section compress and decompress files and pack and unpack archives. bzip2: Compresses a File The bzip2 utility compresses a file by analyzing it and recoding it more efficiently. The new version of the file looks completely different. In fact, because the new file contains many nonprinting characters, you cannot view it directly. The bzip2 utility works particularly well on files that contain a lot of repeated information, such as text and image data, although most image data is already in a compressed format. The following example shows a boring file. Each of the 8,000 lines of the letter_e file contains 72 e's and a NEWLINE character that marks the end of the line. The file occupies more than half a megabyte of disk storage. $ ls -l -rw-rw-r-- 1 sam sam 584000 Mar 1 22:31 letter_e The –l (long) option causes ls to display more information about a file. Here it shows that letter_e is 584,000 bytes long. The –v (verbose) option causes bzip2 to report how much it was able to reduce the size of the file. In this case, it shrank the file by 99.99 percent: $ bzip2 -v letter_e letter_e: 11680.00:1, 0.001 bits/byte, 99.99% saved, 584000 in, 50 out. Compressing and Archiving Files $ ls -l -rw-rw-r-.bz2 filename extension 1 sam sam 50 Mar 61 1 22:31 letter_e.bz2 Now the file is only 50 bytes long. The bzip2 utility also renamed the file, appending .bz2 to its name. This naming convention reminds you that the file is compressed; you would not want to display or print it, for example, without first decompressing it. The bzip2 utility does not change the modification date associated with the file, even though it completely changes the file's contents. Keep the original file by using the –k option tip The bzip2 utility (and its counterpart, bunzip2) remove the original file when they compress or decompress a file. Use the –k (keep) option to keep the original file. In the following, more realistic example, the file zach.jpg contains a computer graphics image: $ ls -l -rw-r--r-- 1 sam sam 33287 Mar 1 22:40 zach.jpg The bzip2 utility can reduce the size of the file by only 28 percent because the image is already in a compressed format: $ bzip2 -v zach.jpg zach.jpg: 1.391:1, 5.749 bits/byte, 28.13% saved, 33287 in, 23922 out. $ ls -l -rw-r--r-- 1 sam sam 23922 Mar 1 22:40 zach.jpg.bz2 Refer to page 615 or to the bzip2 man page for more information. See also www.bzip.org, and the Bzip2 mini-HOWTO (see page 38 for instructions on obtaining this document). bunzip2 and bzcat: Decompress a File You can use the bunzip2 utility to restore a file that has been compressed with bzip2: $ bunzip2 letter_e.bz2 $ ls -l -rw-rw-r-- 1 sam sam 584000 Mar $ bunzip2 zach.jpg.bz2 $ ls -l -rw-r--r-- 1 sam sam 33287 Mar 1 22:31 letter_e 1 22:40 zach.jpg The bzcat utility displays a file that has been compressed with bzip2. The equivalent of cat for .bz2 files, bzcat decompresses the compressed data and displays the decompressed data. Like cat, bzcat does not change the source file. The pipe in the following example redirects the output of bzcat so instead of being displayed on the screen it becomes the input to head, which displays the first two lines of the file: $ bzcat letter_e.bz2 | head -2 eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee 62 Chapter 3 The Utilities After running bzcat, the contents of letter_e.bz2 is unchanged; the file is still stored on the disk in compressed form. bzip2recover The bzip2recover utility supports limited data recovery from media errors. Give the command bzip2recover followed by the name of the compressed, corrupted file from which you want to recover data. gzip: Compresses a File gunzip and zcat The gzip (GNU zip) utility is older and less efficient than bzip2. Its flags and operation are very similar to those of bzip2. A file compressed by gzip is marked by a .gz filename extension. Linux stores manual pages in gzip format to save disk space; likewise, files you download from the Internet are frequently in gzip format. Use gzip, gunzip, and zcat just as you would use bzip2, bunzip2, and bzcat, respectively. Refer to page 724 or to the gzip man page for more information. compress The compress utility can also compress files, albeit not as well as gzip. This utility marks a file it has compressed by adding .Z to its name. gzip versus zip tip Do not confuse gzip and gunzip with the zip and unzip utilities. These last two are used to pack and unpack zip archives containing several files compressed into a single file that has been imported from or is being exported to a system running Windows. The zip utility constructs a zip archive, whereas unzip unpacks zip archives. The zip and unzip utilities are compatible with PKZIP, a Windows program that compresses and archives files. tar: Packs and Unpacks Archives The tar utility performs many functions. Its name is short for tape archive, as its original function was to create and read archive and backup tapes. Today it is used to create a single file (called a tar file, archive, or tarball) from multiple files or directory hierarchies and to extract files from a tar file. The cpio utility (page 644) performs a similar function. In the following example, the first ls shows the sizes of the files g, b, and d. Next tar uses the –c (create), –v (verbose), and –f (write to or read from a file) options to create an archive named all.tar from these files. Each line of output displays the name of the file tar is appending to the archive it is creating. The tar utility adds overhead when it creates an archive. The next command shows that the archive file all.tar occupies about 9,700 bytes, whereas the sum of the sizes of the three files is about 6,000 bytes. This overhead is more appreciable on smaller files, such as the ones in this example. $ ls -l g b d -rw-r--r-1 zach -rw-r--r-1 zach -rw-r--r-1 zach other 1178 Aug 20 14:16 b zach 3783 Aug 20 14:17 d zach 1302 Aug 20 14:16 g $ tar -cvf all.tar g b d g b d Compressing and Archiving Files $ ls -l all.tar -rw-r--r-1 zach $ tar -tvf -rw-r--r--rw-r--r--rw-r--r-- all.tar zach /zach zach /other zach /zach zach 63 9728 Aug 20 14:17 all.tar 1302 2009-08-20 14:16 g 1178 2009-08-20 14:16 b 3783 2009-08-20 14:17 d The final command in the preceding example uses the –t option to display a table of contents for the archive. Use –x instead of –t to extract files from a tar archive. Omit the –v option if you want tar to do its work silently.2 Compressed tar files You can use bzip2, compress, or gzip to compress tar files, making them easier to store and handle. Many files you download from the Internet will already be in one of these formats. Files that have been processed by tar and compressed by bzip2 frequently have a filename extension of .tar.bz2 or .tbz. Those processed by tar and gzip have an extension of .tar.gz, .tgz, or .gz, whereas files processed by tar and compress use .tar.Z as the extension. You can unpack a tarred and gzipped file in two steps. (Follow the same procedure if the file was compressed by bzip2, but use bunzip2 instead of gunzip.) The next example shows how to unpack the GNU make utility after it has been downloaded (ftp.gnu.org/pub/gnu/make/make-3.81.tar.gz): $ ls -l mak* -rw-r--r-- 1 sam sam 1564560 Mar 26 18:25 make-3.81.tar.gz $ gunzip mak* $ ls -l mak* -rw-r--r-- 1 sam sam 6072320 Mar 26 18:25 make-3.81.tar $ tar -xvf mak* make-3.81/ make-3.81/config/ make-3.81/config/dospaths.m4 make-3.81/config/gettext.m4 make-3.81/config/iconv.m4 ... make-3.81/tests/run_make_tests make-3.81/tests/run_make_tests.pl make-3.81/tests/test_driver.pl The first command lists the downloaded tarred and gzipped file: make-3.81.tar.gz (about 1.2 megabytes). The asterisk (*) in the filename matches any characters in any filenames (page 138), so ls displays a list of files whose names begin with mak; in this case there is only one. Using an asterisk saves typing and can improve accuracy with long filenames. The gunzip command decompresses the file and yields 2. Although the original UNIX tar did not use a leading hyphen to indicate an option on the command line, the GNU/Linux version accepts hyphens, but works as well without them. This book precedes tar options with a hyphen for consistency with most other utilities. 64 Chapter 3 The Utilities make-3.81.tar (no .gz extension), which is about 4.8 megabytes. The tar command creates the make-3.81 directory in the working directory and unpacks the files into it. $ ls -ld mak* drwxr-xr-x 8 sam sam 4096 Mar 31 2006 make-3.81 -rw-r--r-- 1 sam sam 6072320 Mar 26 18:25 make-3.81.tar $ ls -l make-3.81 total 2100 -rw-r--r-- 1 sam sam 53838 Mar 31 -rw-r--r-- 1 sam sam 2810 Mar 19 -rw-r--r-- 1 sam sam 18043 Dec 10 -rw-r--r-- 1 sam sam 106390 Mar 31 ... -rw-r--r-- 1 sam sam 16907 Feb 11 -rw-r--r-- 1 sam sam 17397 Feb 11 drwxr-xr-x 6 sam sam 4096 Mar 31 2006 2006 1996 2006 ABOUT-NLS AUTHORS COPYING ChangeLog 2006 vmsjobs.c 2006 vpath.c 2006 w32 After tar extracts the files from the archive, the working directory contains two files whose names start with mak: make-3.81.tar and make-3.81. The –d (directory) option causes ls to display only file and directory names, not the contents of directories as it normally does. The final ls command shows the files and directories in the make-3.81 directory. Refer to page 846 or to the tar man page for more information. tar: the –x option may extract a lot of files caution Some tar archives contain many files. To list the files in the archive without unpacking them, run tar with the –t option and the name of the tar file. In some cases you may want to create a new directory (mkdir [page 86]), move the tar file into that directory, and expand it there. That way the unpacked files will not mingle with existing files, and no confusion will occur. This strategy also makes it easier to delete the extracted files. Depending on how they were created, some tar files automatically create a new directory and put the files into it; the –t option indicates where tar will place the files you extract. tar: the –x option can overwrite files caution The –x option to tar overwrites a file that has the same filename as a file you are extracting. Follow the suggestion in the preceding caution box to avoid overwriting files. optional You can combine the gunzip and tar commands on one command line with a pipe (|), which redirects the output of gunzip so that it becomes the input to tar: $ gunzip -c make-3.81.tar.gz | tar -xvf - The –c option causes gunzip to send its output through the pipe instead of creating a file. The final hyphen (–) causes tar to read from standard input. Refer to "Pipes" (page 131), gzip (pages 62 and 724), and tar (page 846) for more information about how this command line works. A simpler solution is to use the –z option to tar. This option causes tar to call gunzip (or gzip when you are creating an archive) directly and simplifies the preceding command line to Locating Commands 65 $ tar -xvzf make-3.81.tar.gz In a similar manner, the –j option calls bzip2 or bunzip2. Locating Commands The whereis and slocate utilities can help you find a command whose name you have forgotten or whose location you do not know. When multiple copies of a utility or program are present, which tells you which copy you will run. The slocate utility searches for files on the local system. which and whereis: Locate a Utility Search path When you give Linux a command, the shell searches a list of directories for a program with that name and runs the first one it finds. This list of directories is called a search path. For information on how to change the search path see page 298. If you do not change the search path, the shell searches only a standard set of directories and then stops searching. However, other directories on the system may also contain useful utilities. which The which utility locates utilities by displaying the full pathname of the file for the utility. (Chapter 4 contains more information on pathnames and the structure of the Linux filesystem.) The local system may include several utilities that have the same name. When you type the name of a utility, the shell searches for the utility in your search path and runs the first one it finds. You can find out which copy of the utility the shell will run by using which. In the following example, which reports the location of the tar utility: $ which tar /bin/tar The which utility can be helpful when a utility seems to be working in unexpected ways. By running which, you may discover that you are running a nonstandard version of a tool or a different one from the one you expected. ("Important Standard Directories and Files" on page 91 provides a list of standard locations for executable files.) For example, if tar is not working properly and you find that you are running /usr/local/bin/tar instead of /bin/tar, you might suspect that the local version is broken. whereis The whereis utility searches for files related to a utility by looking in standard locations instead of using your search path. For example, you can find the locations for files related to tar: $ whereis tar tar: /bin/tar /usr/include/tar.h /usr/share/man/man1/tar.1.gz In this example whereis finds three references to tar: the tar utility file, a tar header file, and the tar man page. 66 Chapter 3 The Utilities which versus whereis tip Given the name of a utility, which looks through the directories in your search path (page 297), in order, and locates the utility. If your search path includes more than one utility with the specified name, which displays the name of only the first one (the one you would run). The whereis utility looks through a list of standard directories and works independently of your search path. Use whereis to locate a binary (executable) file, any manual pages, and source code for a program you specify; whereis displays all the files it finds. which, whereis, and builtin commands caution Both the which and whereis utilities report only the names for utilities as they are found on the disk; they do not report shell builtins (utilities that are built into a shell; see page 141). When you use whereis to try to find where the echo command (which exists as both a utility program and a shell builtin) is kept, you get the following result: $ whereis echo echo: /bin/echo /usr/share/man/man1/echo.1.gz The whereis utility does not display the echo builtin. Even the which utility reports the wrong information: $ which echo /bin/echo Under bash you can use the type builtin (page 447) to determine whether a command is a builtin: $ type echo echo is a shell builtin slocate/locate: Searches for a File The slocate (secure locate) or locate utility searches for files on the local system: $ slocate motd /usr/share/app-install/icons/xmotd.xpm /usr/share/app-install/desktop/motd-editor.desktop /usr/share/app-install/desktop/xmotd.desktop /usr/share/base-files/motd.md5sums /usr/share/base-files/motd ... You may need to install slocate or locate; see Appendix C. Before you can use slocate or locate the updatedb utility must build or update the slocate database. Typically the database is updated once a day by a cron script (see page 649 for information on crontab and cron). If you are not on a network, skip the rest of this chapter tip If you are the only user on a system that is not connected to a network, you may want to skip the rest of this chapter. If you are not on a network but are set up to send and receive email, read "Email" on page 72. Obtaining User and System Information 67 Obtaining User and System Information This section covers utilities that provide information about who is using the system, what those users are doing, and how the system is running. To find out who is using the local system, you can employ one of several utilities that vary in the details they provide and the options they support. The oldest utility, who, produces a list of users who are logged in on the local system, the device each person is using, and the time each person logged in. The w and finger utilities show more detail, such as each user's full name and the command line each user is running. You can use the finger utility to retrieve information about users on remote systems if the local system is attached to a network. Table 3-1 on page 70 summarizes the output of these utilities. who: Lists Users on the System The who utility displays a list of users who are logged in on the local system. In Figure 3-10 the first column who displays shows that Sam, Max, and Zach are logged in. (Max is logged in from two locations.) The second column shows the device that each user's terminal, workstation, or terminal emulator is connected to. The third column shows the date and time the user logged in. An optional fourth column shows (in parentheses) the name of the system that a remote user logged in from. The information that who displays is useful when you want to communicate with a user on the local system. When the user is logged in, you can use write (page 70) to establish communication immediately. If who does not list the user or if you do not need to communicate immediately, you can send email to that person (page 72). If the output of who scrolls off the screen, you can redirect the output through a pipe (|, page 56) so that it becomes the input to less, which displays the output one screen at a time. You can also use a pipe to redirect the output through grep to look for a specific name. If you need to find out which terminal you are using or what time you logged in, you can use the command who am i: $ who am i max tty2 2009-07-25 16:42 $ who sam max zach max 2009-07-25 2009-07-25 2009-07-25 2009-07-25 Figure 3-10 tty4 tty2 tty1 pts/4 who lists who is logged in 17:18 16:42 16:39 17:27 (coffee) 68 Chapter 3 The Utilities $ finger Login max max sam zach Figure 3-11 Name Max Wild Max Wild Sam the Great Zach Brill Tty *tty2 pts/4 *tty4 *tty1 Idle 3 29 1:07 Login Time Office ... Jul 25 16:42 Jul 25 17:27 (coffee) Jul 25 17:18 Jul 25 16:39 finger I: lists who is logged in finger: Lists Users on the System You can use finger to display a list of users who are logged in on the local system. In addition to usernames, finger supplies each user's full name along with information about which device the user's terminal is connected to, how recently the user typed something on the keyboard, when the user logged in, and available contact information. If the user has logged in over the network, the name of the remote system is shown as the user's location. For example, in Figure 3-11 Max is logged in from the remote system named coffee. The asterisks (*) in front of the device names in the Tty column indicate that the user has blocked messages sent directly to his terminal (refer to "mesg: Denies or Accepts Messages" on page 71). finger can be a security risk security On systems where security is a concern, the system administrator may disable finger. This utility can reveal information that can help a malicious user break into a system. Mac OS X disables remote finger support by default. You can also use finger to learn more about an individual by specifying a username on the command line. In Figure 3-12, finger displays detailed information about Max. Max is logged in and actively using one of his terminals (tty2); he has not used his other terminal (pts/4) for 3 minutes and 7 seconds. You also learn from finger that if you want to set up a meeting with Max, you should contact Zach at extension 1693. .plan and .project Most of the information in Figure 3-12 was collected by finger from system files. The information shown after the heading Plan:, however, was supplied by Max. The finger utility searched for a file named .plan in Max's home directory and displayed its contents. $ finger max Login: max Name: Max Wild Directory: /home/max Shell: /bin/tcsh On since Fri Jul 25 16:42 (PDT) on tty2 (messages off) On since Fri Jul 25 17:27 (PDT) on pts/4 from coffee 3 minutes 7 seconds idle New mail received Sat Jul 25 17:16 2009 (PDT) Unread since Sat Jul 25 16:44 2009 (PDT) Plan: I will be at a conference in Hawaii all next week. If you need to see me, contact Zach Brill, x1693. Figure 3-12 finger II: lists details about one user Obtaining User and System Information 69 (Filenames that begin with a period, such as .plan, are not normally listed by ls and are called hidden filenames [page 82].) You may find it helpful to create a .plan file for yourself; it can contain any information you choose, such as your schedule, interests, phone number, or address. In a similar manner, finger displays the contents of the .project and .pgpkey files in your home directory. If Max had not been logged in, finger would have reported only his user information, the last time he logged in, the last time he read his email, and his plan. You can also use finger to display a user's username. For example, on a system with a user named Helen Simpson, you might know that Helen's last name is Simpson but might not guess her username is hls. The finger utility, which is not case sensitive, can search for information on Helen using her first or last name. The following commands find the information you seek as well as information on other users whose names are Helen or Simpson: $ finger HELEN Login: hls ... $ finger simpson Login: hls ... Name: Helen Simpson. Name: Helen Simpson. See page 695 for more information about finger. w: Lists Users on the System The w utility displays a list of the users who are logged in on the local system. As discussed in the section on who, the information that w displays is useful when you want to communicate with someone at your installation. The first column in Figure 3-13 shows that Max, Zach, and Sam are logged in. The second column shows the name of the device file each user's terminal is connected to. The third column shows the system that a remote user is logged in from. The fourth column shows the time each user logged in. The fifth column indicates how long each user has been idle (how much time has elapsed since the user pressed a key on the keyboard). The next two columns identify how much computer processor time each user has used during this login session and on the task that user is running. The last column shows the command each user is running. The first line the w utility displays includes the time of day, the period of time the computer has been running (in days, hours, and minutes), the number of users $ w 17:47:35 up 1 day, 8:10, USER TTY FROM sam tty4 max tty2 zach tty1 max pts/4 coffee Figure 3-13 The w utility 4 users, load average: 0.34, 0.23, 0.26 IDLE JCPU PCPU WHAT 17:18 29:14m 0.20s 0.00s vi memo 16:42 0.00s 0.20s 0.07s w 16:39 1:07 0.05s 0.00s run_bdgt 17:27 3:10m 0.24s 0.24s -bash 70 Chapter 3 The Utilities logged in, and the load average (how busy the system is). The three load average numbers represent the number of jobs waiting to run, averaged over the past 1, 5, and 15 minutes. Use the uptime utility to display just this line. Table 3-1 compares the w, who, and finger utilities. Table 3-1 Comparison of w, who, and finger Information displayed w who finger Username x x x Terminal-line identification (tty) x x x Login time (and day for old logins) x x x Login date and time Idle time x Program the user is executing x Location the user logged in from CPU time used x x x Full name (or other information from /etc/passwd) x User-supplied vanity information x System uptime and load average x Communicating with Other Users You can use the utilities discussed in this section to exchange messages and files with other users either interactively or through email. write: Sends a Message The write utility sends a message to another user who is logged in. When you and another user use write to send messages to each other, you establish two-way communication. Initially a write command displays a banner on the other user's terminal, saying that you are about to send a message (Figure 3-14). The syntax of a write command line is write username [terminal] $ write max Hi Max, are you there? o Figure 3-14 The write utility I Communicating with Other Users 71 $ write max Hi Max, are you there? o Message from on pts/0 at 16:23 ... Yes Zach, I'm here. o Figure 3-15 The write utility II The username is the username of the user you want to communicate with. The terminal is an optional device name that is useful if the user is logged in more than once. You can display the usernames and device names of all users who are logged in on the local system by using who, w, or finger. To establish two-way communication with another user, you and the other user must each execute write, specifying the other's username as the username. The write utility then copies text, line by line, from one keyboard/display to the other (Figure 3-15). Sometimes it helps to establish a convention, such as typing o (for "over") when you are ready for the other person to type and typing oo (for "over and out") when you are ready to end the conversation. When you want to stop communicating with the other user, press CONTROL-D at the beginning of a line. Pressing CONTROL-D tells write to quit, displays EOF (end of file) on the other user's terminal, and returns you to the shell. The other user must do the same. If the Message from banner appears on your screen and obscures something you are working on, press CONTROL- L or CONTROL- R to refresh the screen and remove the banner. Then you can clean up, exit from your work, and respond to the person who is writing to you. You have to remember who is writing to you, however, because the banner will no longer appear on the screen. mesg: Denies or Accepts Messages By default, messages to your screen are blocked. Give the following mesg command to allow other users to send you messages: $ mesg y If Max had not given this command before Zach tried to send him a message, Zach might have seen the following message: $ write max write: max has messages disabled You can block messages by entering mesg n. Give the command mesg by itself to display is y (for "yes, messages are allowed") or is n (for "no, messages are not allowed"). If you have messages blocked and you write to another user, write displays the following message because, even if you are allowed to write to another user, the user will not be able to respond to you: $ write max write: write: you have write permission turned off. 72 Chapter 3 The Utilities Email Email enables you to communicate with users on the local system and, if the installation is part of a network, with other users on the network. If you are connected to the Internet, you can communicate electronically with users around the world. Email utilities differ from write in that email utilities can send a message when the recipient is not logged in. In this case the email is stored until the recipient reads it. These utilities can also send the same message to more than one user at a time. Many email programs are available for Linux, including the original character-based mailx program, Mozilla/Thunderbird, pine, mail through emacs, KMail, and evolution. Another popular graphical email program is sylpheed (sylpheed.sraoss.jp/en). Two programs are available that can make any email program easier to use and more secure. The procmail program (www.procmail.org) creates and maintains email servers and mailing lists; preprocesses email by sorting it into appropriate files and directories; starts various programs depending on the characteristics of incoming email; forwards email; and so on. The GNU Privacy Guard (GPG or GNUpg; www.gnupg.org) encrypts and decrypts email and makes it almost impossible for an unauthorized person to read. Network addresses If the local system is part of a LAN, you can generally send email to and receive email from users on other systems on the LAN by using their usernames. Someone sending Max email on the Internet would need to specify his domain name (page 952) along with his username. Use this address to send email to the author of this book: Chapter Summary The utilities introduced in this chapter are a small but powerful subset of the many utilities available on a typical Linux system. Because you will use them frequently and because they are integral to the following chapters, it is important that you become comfortable using them. The utilities listed in Table 3-2 manipulate, display, compare, and print files. Table 3-2 File utilities Utility Function cp Copies one or more files (page 49) diff Displays the differences between two files (page 54) file Displays information about the contents of a file (page 56) grep Searches file(s) for a string (page 52) Chapter Summary Table 3-2 73 File utilities (continued) Utility Function head Displays the lines at the beginning of a file (page 52) lpq Displays a list of jobs in the print queue (page 51) lpr Places file(s) in the print queue (page 51) lprm Removes a job from the print queue (page 51) mv Renames a file or moves file(s) to another directory (page 50) sort Puts a file in order by lines (page 54) tail Displays the lines at the end of a file (page 53) uniq Displays the contents of a file, skipping adjacent duplicate lines (page 54) To reduce the amount of disk space a file occupies, you can compress it with the bzip2 utility. Compression works especially well on files that contain patterns, as do most text files, but reduces the size of almost all files. The inverse of bzip2—bunzip2— restores a file to its original, decompressed form. Table 3-3 lists utilities that compress and decompress files. The bzip2 utility is the most efficient of these. Table 3-3 (De)compression utilities Utility Function bunzip2 Returns a file compressed with bzip2 to its original size and format (page 61) bzcat Displays a file compressed with bzip2 (page 61) bzip2 Compresses a file (page 60) compress Compresses a file (not as well as bzip2 or gzip; page 62) gunzip Returns a file compressed with gzip or compress to its original size and format (page 62) gzip Compresses a file (not as well as bzip2; page 62) zcat Displays a file compressed with gzip (page 62) An archive is a file, frequently compressed, that contains a group of files. The tar utility (Table 3-4) packs and unpacks archives. The filename extensions .tar.bz2, .tar.gz, and .tgz identify compressed tar archive files and are often seen on software packages obtained over the Internet. Table 3-4 Archive utility Utility Function tar Creates or extracts files from an archive file (page 62) 74 Chapter 3 The Utilities The utilities listed in Table 3-5 determine the location of a utility on the local system. For example, they can display the pathname of a utility or a list of C++ compilers available on the local system. Table 3-5 Location utilities Utility Function locate Searches for files on the local system (page 66) whereis Displays the full pathnames of a utility, source code, or man page (page 65) which Displays the full pathname of a command you can run (page 65) Table 3-6 lists utilities that display information about other users. You can easily learn a user's full name, the user's login status, the login shell of the user, and other items of information maintained by the system. Table 3-6 User and system information utilities Utility Function finger Displays detailed information about users, including their full names (page 68) hostname Displays the name of the local system (page 49) w Displays detailed information about users who are logged in on the local system (page 69) who Displays information about users who are logged in on the local system (page 67) The utilities shown in Table 3-7 can help you stay in touch with other users on the local network. Table 3-7 User communication utilities Utility Function mesg Permits or denies messages sent by write (page 71) write Sends a message to another user who is logged in (page 70) Table 3-8 lists miscellaneous utilities. Table 3-8 Miscellaneous utilities Utility Function date Displays the current date and time (page 58) echo Copies its arguments (page 941) to the screen (page 57) Advanced Exercises 75 Exercises 1. Which commands can you use to determine who is logged in on a specific terminal? 2. How can you keep other users from using write to communicate with you? Why would you want to? 3. What happens when you give the following commands if the file named done already exists? $ cp to_do done $ mv to_do done 4. How can you display the following line on the screen you are working on? Hi $2, I'm number * (guess)! 5. How can you find the phone number for Ace Electronics in a file named phone that contains a list of names and phone numbers? Which command can you use to display the entire file in alphabetical order? How can you display the file without any adjacent duplicate lines? How can you display the file without any duplicate lines? 6. What happens when you use diff to compare two binary files that are not identical? (You can use gzip to create the binary files.) Explain why the diff output for binary files is different from the diff output for ASCII files. 7. Create a .plan file in your home directory. Does finger display the contents of your .plan file? 8. What is the result of giving the which utility the name of a command that resides in a directory that is not in your search path? 9. Are any of the utilities discussed in this chapter located in more than one directory on the local system? If so, which ones? 10. Experiment by calling the file utility with the names of files in /usr/bin. How many different types of files are there? 11. Which command can you use to look at the first few lines of a file named status.report? Which command can you use to look at the end of the file? Advanced Exercises 12. Re-create the colors.1 and colors.2 files used in Figure 3-8 on page 55. Test your files by running diff –u on them. Do you get the same results as in the figure? 76 Chapter 3 The Utilities 13. Try giving these two commands: $ echo cat $ cat echo Explain the differences between the output of each command. 14. Repeat exercise 5 using the file phone.gz, a compressed version of the list of names and phone numbers. Consider more than one approach to answer each question, and explain how you made your choices. 15. Find existing files or create files that a. gzip compresses by more than 80 percent. b. gzip compresses by less than 10 percent. c. Get larger when compressed with gzip. d. Use ls –l to determine the sizes of the files in question. Can you characterize the files in a, b, and c? 16. Older email programs were not able to handle binary files. Suppose that you are emailing a file that has been compressed with gzip, which produces a binary file, and the recipient is using an old email program. Refer to the man page on uuencode, which converts a binary file to ASCII. Learn about the utility and how to use it. a. Convert a compressed file to ASCII using uuencode. Is the encoded file larger or smaller than the compressed file? Explain. (If uuencode is not on the local system, you can install it using yum or aptitude [both in Appendix C]; it is part of the sharutils package.) b. Would it ever make sense to use uuencode on a file before compressing it? Explain. 4 The Filesystem In This Chapter A filesystem is a set of data structures (page 950) that usually resides on part of a disk and that holds directories of files. Filesystems store user and system data that are the basis of users' work on the system and the system's existence. This chapter discusses the organization and terminology of the Linux filesystem, defines ordinary and directory files, and explains the rules for naming them. It also shows how to create and delete directories, move through the filesystem, and use absolute and relative pathnames to access files in various directories. It includes a discussion of important files and directories as well as file access permissions and Access Control Lists (ACLs), which allow you to share selected files with other users. It concludes with a

discussion of hard and symbolic links, which can make a single file appear in more than one directory. 4Chapter4 In addition to reading this chapter, you may want to refer to the df, fsck, mkfs, and tune2fs utilities in Part V for more information on filesystems. If you are running Mac OS X, see "Filesystems" on page 927. 77 78 Chapter 4 The Filesystem Grandparent Aunt Mom Uncle Sister Brother Self Daughter 1 Grandchild 1 Figure 4-1 Daughter 2 Grandchild 2 A family tree The Hierarchical Filesystem Family tree A hierarchical structure (page 957) frequently takes the shape of a pyramid. One example of this type of structure is found by tracing a family's lineage: A couple has a child, who may in turn have several children, each of whom may have more children. This hierarchical structure is called a family tree (Figure 4-1). Directory tree Like the family tree it resembles, the Linux filesystem is called a tree. It consists of a set of connected files. This structure allows you to organize files so you can easily find any particular one. On a standard Linux system, each user starts with one directory, to which the user can add subdirectories to any desired level. By creating multiple levels of subdirectories, a user can expand the structure as needed. Subdirectories Typically each subdirectory is dedicated to a single subject, such as a person, project, or event. The subject dictates whether a subdirectory should be subdivided further. For example, Figure 4-2 shows a secretary's subdirectory named correspond. This directory contains three subdirectories: business, memos, and personal. The business directory contains files that store each letter the secretary types. If you expect many letters to go to one client, as is the case with milk_co, you can dedicate a subdirectory to that client. One major strength of the Linux filesystem is its ability to adapt to users' needs. You can take advantage of this strength by strategically organizing your files so they are most convenient and useful for you. Directory Files and Ordinary Files Like a family tree, the tree representing the filesystem is usually pictured upside down, with its root at the top. Figures 4-2 and 4-3 show that the tree "grows" Directory Files and Ordinary Files 79 correspond personal memos business milk_co letter_1 Figure 4-2 cheese_co letter_2 A secretary's directories downward from the root, with paths connecting the root to each of the other files. At the end of each path is either an ordinary file or a directory file. Special files, which can also appear at the ends of paths, provide access to operating system features. Ordinary files, or simply files, appear at the ends of paths that cannot support other paths. Directory files, also referred to as directories or folders, are the points that other paths can branch off from. (Figures 4-2 and 4-3 show some empty directories.) When you refer to the tree, up is toward the root and down is away from the root. Directories directly connected by a path are called parents (closer to the root) and children (farther from the root). A pathname is a series of names that trace a path along branches from one file to another. See page 83 for more information about pathnames. Filenames Every file has a filename. The maximum length of a filename varies with the type of filesystem; Linux supports several types of filesystems. Although most of today's filesystems allow files with names up to 255 characters long, some filesystems Directory Directory Directory Directory Ordinary File Directory Directory Ordinary File Figure 4-3 Ordinary File Ordinary File Directories and ordinary files 80 Chapter 4 The Filesystem restrict filenames to fewer characters. While you can use almost any character in a filename, you will avoid confusion if you choose characters from the following list: • • • • • • Uppercase letters (A–Z) Lowercase letters (a–z) Numbers (0–9) Underscore (_) Period (.) Comma (,) Like the children of one parent, no two files in the same directory can have the same name. (Parents give their children different names because it makes good sense, but Linux requires it.) Files in different directories, like the children of different parents, can have the same name. The filenames you choose should mean something. Too often a directory is filled with important files with such unhelpful names as hold1, wombat, and junk, not to mention foo and foobar. Such names are poor choices because they do not help you recall what you stored in a file. The following filenames conform to the suggested syntax and convey information about the contents of the file: • • • • • • Filename length correspond january davis reports 2001 acct_payable When you share your files with users on other systems, you may need to make long filenames differ within the first few characters. Systems running DOS or older versions of Windows have an 8-character filename body length limit and a 3-character filename extension length limit. Some UNIX systems have a 14-character limit and older Macintosh systems have a 31-character limit. If you keep the filenames short, they are easy to type; later you can add extensions to them without exceeding the shorter limits imposed by some filesystems. The disadvantage of short filenames is that they are typically less descriptive than long filenames. The stat utility (page 835) can display the maximum length of a filename. In the following example, the number following Namelen: is the maximum number of characters permitted in a filename on the specified filesystem (/home). $ stat -f /home | grep -i name ID: ff1f5275f648468b Namelen: 255 Type: ext2/ext3 The –f option tells stat to display filesystem status instead of file status. The –i (case insensitive) option tells grep to not consider case in its search. Long filenames enable you to assign descriptive names to files. To help you select among files without typing entire filenames, shells support filename completion. For more information about this feature, see the "Filename completion" tip on page 49. Directory Files and Ordinary Files 81 Case sensitivity You can use uppercase and/or lowercase letters within filenames, but be careful: Many filesystems are case sensitive. For example, the popular ext family of filesystems and the UFS filesystem are case sensitive, so files named JANUARY, January, and january refer to three distinct files. The FAT family of filesystems (used mostly for removable media) is not case sensitive, so those three filenames represent the same file. The HFS+ filesystem, which is the default OS X filesystem, is case preserving but not case sensitive; refer to "Case Sensitivity" on page 927 for more information. Do not use SPACEs within filenames caution Although you can use SPACEs within filenames, it is a poor idea. Because a SPACE is a special character, you must quote it on a command line. Quoting a character on a command line can be difficult for a novice user and cumbersome for an experienced user. Use periods or underscores instead of SPACEs: joe.05.04.26, new_stuff. If you are working with a filename that includes a SPACE, such as a file from another operating system, you must quote the SPACE on the command line by preceding it with a backslash or by placing quotation marks on either side of the filename. The two following commands send the file named my file to the printer. $ lpr my\ file $ lpr "my file" Filename Extensions A filename extension is the part of the filename following an embedded period. In the filenames listed in Table 4-1, filename extensions help describe the contents of the file. Some programs, such as the C programming language compiler, default to specific filename extensions; in most cases, however, filename extensions are optional. Use extensions freely to make filenames easy to understand. If you like, you can use several periods within the same filename—for example, notes.4.10.01 or files.tar.gz. Under Mac OS X, some applications use filename extensions to identify files, but many use type codes and creator codes (page 931). Table 4-1 Filename extensions Filename with extension Meaning of extension compute.c A C programming language source file compute.o The object code file for compute.c compute The executable file for compute.c memo.0410.txt A text file memo.pdf A PDF file; view with xpdf or kpdf under a GUI memo.ps A PostScript file; view with gs or kpdf under a GUI memo.Z A file compressed with compress (page 62); use uncompress or gunzip (page 62) to decompress memo.tgz or memo.tar.gz A tar (page 62) archive of files compressed with gzip (page 62) memo.gz A file compressed with gzip (page 62); view with zcat or decompress with gunzip (both on page 62) 82 Chapter 4 The Filesystem Table 4-1 Filename extensions (continued) Filename with extension Meaning of extension memo.bz2 A file compressed with bzip2 (page 60); view with bzcat or decompress with bunzip2 (both on page 61) memo.html A file meant to be viewed using a Web browser, such as Firefox photo.gif, photo.jpg, photo.jpeg, photo.bmp, photo.tif, or photo.tiff A file containing graphical information, such as a picture Hidden Filenames A filename that begins with a period is called a hidden filename (or a hidden file or sometimes an invisible file) because ls does not normally display it. The command ls –a displays all filenames, even hidden ones. Names of startup files (page 82) usually begin with a period so that they are hidden and do not clutter a directory listing. The .plan file (page 68) is also hidden. Two special hidden entries—a single and double period (. and ..)—appear in every directory (page 88). The Working Directory pwd While you are logged in on a character-based interface to a Linux system, you are always associated with a directory. The directory you are associated with is called the working directory or current directory. Sometimes this association is referred to in a physical sense: "You are in (or working in) the zach directory." The pwd (print working directory) shell builtin (page 141) displays the pathname of the working directory. Your Home Directory When you first log in, the working directory is your home directory. To display the pathname of your home directory, use pwd just after you log in (Figure 4-4). Linux home directories are typically located in /home while Mac OS X home directories are located in /Users. When used without any arguments, the ls utility displays a list of the files in the working directory. Because your home directory has been the only working directory you have used so far, ls has always displayed a list of files in your home directory. (All the files you have created up to this point were created in your home directory.) Startup Files Startup files, which appear in your home directory, give the shell and other programs information about you and your preferences. Under Mac OS X these files are login: max Password: Last login: Wed Oct 20 11:14:21 from bravo $ pwd /home/max Figure 4-4 Logging in and displaying the pathname of your home directory Pathnames 83 called configuration files or preference files (page 936). Frequently one of these files tells the shell what kind of terminal you are using (page 906) and executes the stty (set terminal) utility to establish the erase (page 29) and line kill (page 30) keys. Either you or the system administrator can put a shell startup file containing shell commands in your home directory. The shell executes the commands in this file each time you log in. Because the startup files have hidden filenames, you must use the ls –a command to see whether one is in your home directory. See page 271 (bash) and page 352 (tcsh) for more information about startup files. Pathnames Every file has a pathname, which is a trail from a directory through part of the directory hierarchy to an ordinary file or a directory. Within a pathname, a slash (/) to the right of a filename indicates that the file is a directory file. The file following the slash can be an ordinary file or a directory file. The simplest pathname is a simple filename, which points to a file in the working directory. This section discusses absolute and relative pathnames and explains how to use each. If you are running Mac OS X, refer also to page 928 for information on Carbon pathnames. Absolute Pathnames / (root) The root directory of the filesystem hierarchy does not have a name; it is referred to as the root directory and is represented by a / (slash) standing alone or at the left end of a pathname. An absolute pathname starts with a slash (/), which represents the root directory. The slash is followed by the name of a file located in the root directory. An absolute pathname continues, tracing a path through all intermediate directories, to the file identified by the pathname. String all the filenames in the path together, following each directory with a slash (/). This string of filenames is called an absolute pathname because it locates a file absolutely by tracing a path from the root directory to the file. Typically the absolute pathname of a directory does not include the trailing slash, although that format may be used to emphasize that the pathname specifies a directory (e.g., /home/zach/). The part of a pathname following the final slash is called a simple filename, filename, or basename. Figure 4-5 (next page) shows the absolute pathnames of directories and ordinary files in part of a filesystem hierarchy. Using an absolute pathname, you can list or otherwise work with any file on the local system, assuming you have permission to do so, regardless of the working directory at the time you give the command. For example, Sam can give the following command while working in his home directory to list the files in the /usr/bin directory: $ pwd /home/sam $ ls /usr/bin 7z 7za 822-date ... kwin kwin_killer_helper kwin_rules_dialog 84 Chapter 4 The Filesystem / /home /etc home tmp etc /home/hls max zach hls /home/hls/notes /home/zach bin notes /home/hls/bin/log report Figure 4-5 log Absolute pathnames ~ (Tilde) in Pathnames In another form of absolute pathname, the shell expands the characters ~/ (a tilde followed by a slash) at the start of a pathname into the pathname of your home directory. Using this shortcut, you can display your .bashrc startup file (page 272) with the following command, no matter which directory is the working directory: $ less ~/.bashrc A tilde quickly references paths that start with your or someone else's home directory. The shell expands a tilde followed by a username at the beginning of a pathname into the pathname of that user's home directory. For example, assuming he has permission to do so, Max can examine Sam's .bashrc file with the following command: $ less ~sam/.bashrc Refer to "Tilde Expansion" on page 337 for more information. Relative Pathnames A relative pathname traces a path from the working directory to a file. The pathname is relative to the working directory. Any pathname that does not begin with the root directory (represented by /) or a tilde (~) is a relative pathname. Like absolute pathnames, relative pathnames can trace a path through many directories. The simplest relative pathname is a simple filename, which identifies a file in the working directory. The examples in the next sections use absolute and relative pathnames. Significance of the Working Directory To access any file in the working directory, you need only a simple filename. To access a file in another directory, you must use a pathname. Typing a long pathname is tedious and increases the chance of making a mistake. This possibility is less likely under a GUI, where you click filenames or icons. You can choose a working directory for any particular task to reduce the need for long pathnames. Your choice of a Working with Directories 85 / .. home tmp etc working directory = . max zach hls notes ../zach bin notes bin/log report Figure 4-6 log Relative pathnames working directory does not allow you to do anything you could not do otherwise— it just makes some operations easier. When using a relative pathname, know which directory is the working directory caution The location of the file that you are accessing with a relative pathname is dependent on (is relative to) the working directory. Always make sure you know which directory is the working directory before you use a relative pathname. Use pwd to verify the directory. If you are creating a file using vim and you are not where you think you are in the file hierarchy, the new file will end up in an unexpected location. It does not matter which directory is the working directory when you use an absolute pathname. Thus, the following command always edits a file named goals in your home directory: $ vim ~/goals Refer to Figure 4-6 as you read this paragraph. Files that are children of the working directory can be referenced by simple filenames. Grandchildren of the working directory can be referenced by short relative pathnames: two filenames separated by a slash. When you manipulate files in a large directory structure, using short relative pathnames can save you time and aggravation. If you choose a working directory that contains the files used most often for a particular task, you need use fewer long, cumbersome pathnames. (The . and .. entries are explained on page 88.) Working with Directories This section discusses how to create directories (mkdir), switch between directories (cd), remove directories (rmdir), use pathnames to make your work easier, and move and copy files and directories between directories. It concludes with a section that lists and describes briefly important standard directories and files in the Linux filesystem. 86 Chapter 4 The Filesystem / home max names temp literature demo promo Figure 4-7 The file structure developed in the examples mkdir: Creates a Directory The mkdir utility creates a directory. The argument (page 941) to mkdir becomes the pathname of the new directory. The following examples develop the directory structure shown in Figure 4-7. In the figure, the directories that are added appear in a lighter shade than the others and are connected by dashes. In Figure 4-8, pwd shows that Max is working in his home directory (/home/max) and ls shows the names of the files in his home directory: demo, names, and temp. Using mkdir, Max creates a directory named literature as a child of his home directory. He uses a relative pathname (a simple filename) because he wants the literature directory to be a child of the working directory. Max could have used an absolute pathname to create the same directory: mkdir /home/max/literature or mkdir ~max/literature. The second ls in Figure 4-8 verifies the presence of the new directory. The –F option to ls displays a slash after the name of each directory and an asterisk after each executable $ pwd /home/max $ ls demo names temp $ mkdir literature $ ls demo literature names temp $ ls -F demo literature/ names temp $ ls literature $ Figure 4-8 The mkdir utility Working with Directories 87 file (shell script, utility, or application). When you call it with an argument that is the name of a directory, ls lists the contents of that directory. The final ls does not display anything because there are no files in the literature directory. The following commands show two ways to create the promo directory as a child of the newly created literature directory. The first way checks that /home/max is the working directory and uses a relative pathname: $ pwd /home/max $ mkdir literature/promo The second way uses an absolute pathname: $ mkdir /home/max/literature/promo Use the –p (parents) option to mkdir to create both the literature and promo directories with one command: $ pwd /home/max $ ls demo names temp $ mkdir -p literature/promo or $ mkdir -p /home/max/literature/promo cd: Changes to Another Working Directory The cd (change directory) utility makes another directory the working directory but does not change the contents of the working directory. Figure 4-9 shows two ways to make the /home/max/literature directory the working directory, as verified by pwd. First Max uses cd with an absolute pathname to make literature his working directory— it does not matter which is the working directory when you give a command with an absolute pathname. A pwd command confirms the change made by Max. When used without an argument, cd makes your home directory the working directory, as it was when you logged in. The second cd command in Figure 4-9 does not have an argument so it $ cd /home/max/literature $ pwd /home/max/literature $ cd $ pwd /home/max $ cd literature $ pwd /home/max/literature Figure 4-9 cd changes the working directory 88 Chapter 4 The Filesystem makes Max's home directory the working directory. Finally, knowing that he is working in his home directory, Max uses a simple filename to make the literature directory his working directory (cd literature) and confirms the change using pwd. The working directory versus your home directory tip The working directory is not the same as your home directory. Your home directory remains the same for the duration of your session and usually from session to session. Immediately after you log in, you are always working in the same directory: your home directory. Unlike your home directory, the working directory can change as often as you like. You have no set working directory, which explains why some people refer to it as the current directory. When you log in and until you change directories using cd, your home directory is the working directory. If you were to change directories to Sam's home directory, then Sam's home directory would be the working directory. The . and .. Directory Entries The mkdir utility automatically puts two entries in each directory it creates: a single period (.) and a double period (..). See Figure 4-6 on page 85. The . is synonymous with the pathname of the working directory and can be used in its place; the .. is synonymous with the pathname of the parent of the working directory. These entries are hidden because their filenames begin with a period. With the literature directory as the working directory, the following example uses .. three times: first to list the contents of the parent directory (/home/max), second to copy the memoA file to the parent directory, and third to list the contents of the parent directory again. $ pwd /home/max/literature $ ls .. demo literature names $ cp memoA .. $ ls .. demo literature memoA temp names temp After using cd to make promo (a subdirectory of literature) his working directory, Max can use a relative pathname to call vim to edit a file in his home directory. $ cd promo $ vim ../../names You can use an absolute or relative pathname or a simple filename virtually anywhere a utility or program requires a filename or pathname. This usage holds true for ls, vim, mkdir, rm, and most other Linux utilities. rmdir: Deletes a Directory The rmdir (remove directory) utility deletes a directory. You cannot delete the working directory or a directory that contains files other than the . and .. entries. If you Working with Directories 89 need to delete a directory that has files in it, first use rm to delete the files and then delete the directory. You do not have to (nor can you) delete the . and .. entries; rmdir removes them automatically. The following command deletes the promo directory: $ rmdir /home/max/literature/promo The rm utility has a –r option (rm –r filename) that recursively deletes files, including directories, within a directory and also deletes the directory itself. Use rm –r carefully, if at all caution Although rm –r is a handy command, you must use it carefully. Do not use it with an ambiguous file reference such as *. It is frighteningly easy to wipe out your entire home directory with a single short command. Using Pathnames touch Use a text editor to create a file named letter if you want to experiment with the examples that follow. Alternatively you can use touch (page 862) to create an empty file: $ cd $ pwd /home/max $ touch letter With /home/max as the working directory, the following example uses cp with a relative pathname to copy the file letter to the /home/max/literature/promo directory. (You will need to create promo again if you deleted it earlier.) The copy of the file has the simple filename letter.0610: $ cp letter literature/promo/letter.0610 If Max does not change to another directory, he can use vim as shown to edit the copy of the file he just made: $ vim literature/promo/letter.0610 If Max does not want to use a long pathname to specify the file, he can use cd to make promo the working directory before using vim: $ cd literature/promo $ pwd /home/max/literature/promo $ vim letter.0610 To make the parent of the working directory (named /home/max/literature) the new working directory, Max can give the following command, which takes advantage of the .. directory entry: $ cd .. $ pwd /home/max/literature 90 Chapter 4 The Filesystem / home zach max sam names temp literature names Figure 4-10 temp Using mv to move names and temp mv, cp: Move or Copy Files Chapter 3 discussed the use of mv to rename files. However, mv works even more generally: You can use this utility to move files from one directory to another (change the pathname of a file) as well as to change a simple filename. When used to move one or more files to a new directory, the mv command has this syntax: mv existing-file-list directory If the working directory is /home/max, Max can use the following command to move the files names and temp from the working directory to the literature directory: $ mv names temp literature This command changes the absolute pathnames of the names and temp files from /home/max/names and /home/max/temp to /home/max/literature/names and /home/max/literature/temp, respectively (Figure 4-10). Like most Linux commands, mv accepts either absolute or relative pathnames. As you work with Linux and create more files, you will need to create new directories using mkdir to keep the files organized. The mv utility is a useful tool for moving files from one directory to another as you extend your directory hierarchy. The cp utility works in the same way as mv does, except that it makes copies of the existing-file-list in the specified directory. mv: Moves a Directory Just as it moves ordinary files from one directory to another, so mv can move directories. The syntax is similar except that you specify one or more directories, not ordinary files, to move: mv existing-directory-list new-directory Working with Directories 91 / bin sbin mail Figure 4-11 var spool dev usr bin etc sbin tmp max home zach root hls A typical FHS-based Linux filesystem structure If new-directory does not exist, the existing-directory-list must contain just one directory name, which mv changes to new-directory (mv renames the directory). Although you can rename directories using mv, you cannot copy their contents with cp unless you use the –r (recursive) option. Refer to the explanations of tar (page 846) and cpio (page 644) for other ways to copy and move directories. Important Standard Directories and Files Originally files on a Linux system were not located in standard places within the directory hierarchy. The scattered files made it difficult to document and maintain a Linux system and just about impossible for someone to release a software package that would compile and run on all Linux systems. The first standard for the Linux filesystem, the FSSTND (Linux Filesystem Standard), was released early in 1994. In early 1995 work was started on a broader standard covering many UNIX-like systems: FHS (Linux Filesystem Hierarchy Standard; proton.pathname.com/fhs). More recently FHS has been incorporated in LSB (Linux Standard Base; www.linuxfoundation.org/en/LSB), a workgroup of FSG (Free Standards Group). Finally, FSG combined with Open Source Development Labs (OSDL) to form the Linux Foundation (www.linuxfoundation.org). Figure 4-11 shows the locations of some important directories and files as specified by FHS. The significance of many of these directories will become clear as you continue reading. The following list describes the directories shown in Figure 4-11, some of the directories specified by FHS, and some other directories. Most Linux distributions do not use all the directories specified by FHS. Be aware that you cannot always determine the function of a directory by its name. For example, although /opt stores addon software, /etc/opt stores configuration files for the software in /opt. / Root The root directory, present in all Linux filesystem structures, is the ancestor of all files in the filesystem. /bin Essential command binaries Holds the files needed to bring the system up and run it when it first comes up in single-user or recovery mode. /boot Static files of the boot loader Contains all files needed to boot the system. /dev Device files Contains all files that represent peripheral devices, such as disk drives, terminals, and printers. Previously this directory was filled with all possible devices. The udev utility provides a dynamic device directory that enables /dev to contain only devices that are present on the system. 92 Chapter 4 The Filesystem /etc Machine–local system configuration files Holds administrative, configuration, and other system files. One of the most important is /etc/passwd, which contains a list of all users who have permission to use the system. Mac OS X uses Open Directory (page 926) in place of /etc/passwd. /etc/opt Configuration files for add-on software packages kept in /opt /etc/X11 Machine–local configuration files for the X Window System /home User home directories Each user's home directory is typically one of many sub- directories of the /home directory. As an example, assuming that users' directories are under /home, the absolute pathname of Zach's home directory is /home/zach. On some systems the users' directories may not be found under /home but instead might be spread among other directories such as /inhouse and /clients. Under Mac OS X user home directories are typically located in /Users. /lib Shared libraries /lib/modules Loadable kernel modules /mnt Mount point for temporarily mounting filesystems /opt Add-on (optional) software packages /proc Kernel and process information virtual filesystem /root Home directory for the root account /sbin Essential system binaries Utilities used for system administration are stored in /sbin and /usr/sbin. The /sbin directory includes utilities needed during the booting process, and /usr/sbin holds utilities used after the system is up and running. In older versions of Linux, many system administration utilities were scattered through several directories that often included other system files (/etc, /usr/bin, /usr/adm, /usr/include). /sys Device pseudofilesystem /tmp Temporary files /Users User home directories Under Mac OS X, each user's home directory is typically one of many subdirectories of the /Users directory. Linux typically stores home directories in /home. /usr Second major hierarchy Traditionally includes subdirectories that contain informa- tion used by the system. Files in /usr subdirectories do not change often and may be shared by several systems. /usr/bin Most user commands Contains standard Linux/OS X utility programs—that is, binaries that are not needed in single-user or recovery mode. /usr/games Games and educational programs /usr/include Header files used by C programs /usr/lib Libraries /usr/local Local hierarchy Holds locally important files and directories that are added to the system. Subdirectories can include bin, games, include, lib, sbin, share, and src. Access Permissions 93 /usr/sbin Nonvital system administration binaries See /sbin. /usr/share Architecture-independent data Subdirectories can include dict, doc, games, info, locale, man, misc, terminfo, and zoneinfo. /usr/share/doc Documentation /usr/share/info GNU info system's primary directory /usr/share/man Online manuals /usr/src Source code /var Variable data Files with contents that vary as the system runs are kept in sub- directories under /var. The most common examples are temporary files, system log files, spooled files, and user mailbox files. Subdirectories can include cache, lib, lock, log, mail, opt, run, spool, tmp, and yp. Older versions of Linux scattered such files through several subdirectories of /usr (/usr/adm, /usr/mail, /usr/spool, /usr/tmp). /var/log Log files Contains lastlog (a record of the last login by each user), messages (system messages from syslogd), and wtmp (a record of all logins/logouts), among other log files. /var/spool Spooled application data Contains anacron, at, cron, lpd, mail, mqueue, samba, and other directories. The file /var/spool/mail is typically a link to /var/mail. Access Permissions Linux supports two methods of controlling who can access a file and how they can access it: traditional Linux access permissions and Access Control Lists (ACLs). This section describes traditional Linux access permissions. See page 99 for a discussion of ACLs, which provide finer-grained control of access permissions than do traditional access permissions. Three types of users can access a file: the owner of the file (owner), a member of a group that the file is associated with (group), and everyone else (other). A user can attempt to access an ordinary file in three ways: by trying to read from, write to, or execute it. ls –l: Displays Permissions When you call ls with the –l option and the name of one or more ordinary files, ls displays a line of information about the file. The following example displays information for two files. The file letter.0610 contains the text of a letter, and check_spell contains a shell script, a program written in a high-level shell programming language: $ ls -l letter.0610 check_spell -rwxr-xr-x 1 max pubs 852 Jul 31 13:47 check_spell -rw-r--r-- 1 max pubs 3355 Jun 22 12:44 letter.0610 e am en Fi l Da t of e an mo d t di f i m e ic a ti o n S iz e p Gr ou Ty p eo f fi le Fi l ea pe c c r m es iss s i on s AC Lf la g Lin ks Ow ne r 94 Chapter 4 The Filesystem -rwxrwxr-x+..3.max.....pubs.........2048.Aug.12.13:15.memo Figure 4-12 The columns displayed by the ls –l command From left to right, the lines that an ls –l command displays contain the following information (refer to Figure 4-12): • The type of file (first character) • The file's access permissions (the next nine characters) • The ACL flag (present if the file has an ACL, page 99) • The number of links to the file (page 104) • The name of the owner of the file (usually the person who created the file) • The name of the group the file is associated with • The size of the file in characters (bytes) • The date and time the file was created or last modified • The name of the file Type of file The type of file (first column) for letter.0610 is a hyphen (–) because it is an ordinary file. Directory files have a d in this column; see Figure V-19 on page 748 for a list of other characters that can appear in this position. Permissions The next three characters specify the access permissions for the owner of the file: r (in the first position) indicates read permission, w (in the second position) indicates write permission, and x (in the third position) indicates execute permission. A – in one of the positions indicates that the owner does not have the permission associated with that position. In a similar manner the next three characters represent permissions for the group, and the final three characters represent permissions for other (everyone else). In the preceding example, the owner of letter.0610 can read from and write to the file, whereas the group and others can only read from the file and no one is allowed to execute it. Although execute permission can be allowed for any file, it does not make sense to assign execute permission to a file that contains a document, such as a letter. The check_spell file is an executable shell script, so execute permission is appropriate for it. (The owner, group, and others have execute permission.) For more information refer to "Discussion" on page 748. chmod: Changes Access Permissions The Linux file access permission scheme lets you give other users access to the files you want to share yet keep your private files confidential. You can allow other users Access Permissions 95 to read from and write to a file (handy if you are one of several people working on a joint project). You can allow others only to read from a file (perhaps a project specification you are proposing). Or you can allow others only to write to a file (similar to an inbox or mailbox, where you want others to be able to send you mail but do not want them to read your mail). Similarly you can protect entire directories from being scanned (covered shortly). A user with root privileges can access any file on the system security There is an exception to the access permissions described in this section. Anyone who can gain root privileges has full access to all files, regardless of the file's owner or access permissions. The owner of a file controls which users have permission to access the file and how those users can access it. When you own a file, you can use the chmod (change mode) utility to change access permissions for that file. You can specify symbolic (relative) or numeric (absolute) arguments to chmod. Symbolic Arguments to chmod The following example, which uses symbolic arguments to chmod, adds (+) read and write permissions (rw) for all (a) users: $ ls -l letter.0610 -rw------- 1 max pubs 3355 Jun 22 12:44 letter.0610 $ chmod a+rw letter.0610 $ ls -l letter.0610 -rw-rw-rw- 1 max pubs 3355 Jun 22 12:44 letter.0610 You must have read permission to execute a shell script tip Because a shell needs to read a shell script (a text file containing shell commands) before it can execute the commands within that script, you must have read permission for the file containing the script to execute it. You also need execute permission to execute a shell script directly from the command line. In contrast, binary (program) files do not need to be read; they are executed directly. You need only execute permission to run a binary program. Using symbolic arguments with chmod modifies existing permissions; the change a given argument makes depends on (is relative to) the existing permissions. In the next example, chmod removes (–) read (r) and execute (x) permissions for other (o) users. The owner and group permissions are not affected. $ ls -l check_spell -rwxr-xr-x 1 max pubs 852 Jul 31 13:47 check_spell $ chmod o-rx check_spell $ ls -l check_spell -rwxr-x--- 1 max pubs 852 Jul 31 13:47 check_spell In addition to a (all) and o (other), you can use g (group) and u (user, although user refers to the owner of the file who may or may not be the user of the file at any given time) in the argument to chmod. For example, chmod a+x adds execute permission for all users (other, group, and owner) and chmod go–rwx removes all permissions for all but the owner of the file. 96 Chapter 4 The Filesystem chmod: o for other, u for owner tip When using chmod, many people assume that the o stands for owner; it does not. The o stands for other, whereas u stands for owner (user). The acronym UGO (user-group-other) may help you remember how permissions are named. Numeric Arguments to chmod You can also use numeric arguments to specify permissions with chmod. In place of the letters and symbols specifying permissions used in the previous examples, numeric arguments comprise three octal digits. (A fourth, leading digit controls setuid and setgid permissions and is discussed next.) The first digit specifies permissions for the owner, the second for the group, and the third for other users. A 1 gives the specified user(s) execute permission, a 2 gives write permission, and a 4 gives read permission. Construct the digit representing the permissions for the owner, group, or others by ORing (adding) the appropriate values as shown in the following examples. Using numeric arguments sets file permissions absolutely; it does not modify existing permissions as symbolic arguments do. In the following example, chmod changes permissions so only the owner of the file can read from and write to the file, regardless of how permissions were previously set. The 6 in the first position gives the owner read (4) and write (2) permissions. The 0s remove all permissions for the group and other users. $ chmod 600 letter.0610 $ ls -l letter.0610 -rw------- 1 max pubs 3355 Jun 22 12:44 letter.0610 Next, 7 (4 + 2 + 1) gives the owner read, write, and execute permissions. The 5 (4 + 1) gives the group and other users read and execute permissions: $ chmod 755 check_spell $ ls -l check_spell -rwxr-xr-x 1 max pubs 852 Jul 31 13:47 check_spell Refer to Table V-8 on page 628 for more examples of numeric permissions. Refer to page 278 for more information on using chmod to make a file executable and to page 626 for information on absolute arguments and chmod in general. Setuid and Setgid Permissions When you execute a file that has setuid (set user ID) permission, the process executing the file takes on the privileges of the file's owner. For example, if you run a setuid program that removes all files in a directory, you can remove files in any of the file owner's directories, even if you do not normally have permission to do so. In a similar manner, setgid (set group ID) permission gives the process executing the file the privileges of the group the file is associated with. Access Permissions 97 Minimize use of setuid and setgid programs owned by root security Executable files that are setuid and owned by root have root privileges when they run, even if they are not run by root. This type of program is very powerful because it can do anything that root can do (and that the program is designed to do). Similarly executable files that are setgid and belong to the group root have extensive privileges. Because of the power they hold and their potential for destruction, it is wise to avoid indiscriminately creating and using setuid programs owned by root and setgid programs belonging to the group root. Because of their inherent dangers, many sites minimize the use of these programs. One necessary setuid program is passwd. The following example shows a user working with root privileges and using symbolic arguments to chmod to give one program setuid privileges and another program setgid privileges. The ls –l output (page 93) shows setuid permission by displaying an s in the owner's executable position and setgid permission by displaying an s in the group's executable position: $ ls -l myprog* -rwxr-xr-x 1 root pubs 19704 Jul 31 14:30 myprog1 -rwxr-xr-x 1 root pubs 19704 Jul 31 14:30 myprog2 # chmod u+s myprog1 # chmod g+s myprog2 $ ls -l myprog* -rwsr-xr-x 1 root pubs 19704 Jul 31 14:30 myprog1 -rwxr-sr-x 1 root pubs 19704 Jul 31 14:30 myprog2 The next example uses numeric arguments to chmod to make the same changes. When you use four digits to specify permissions, setting the first digit to 1 sets the sticky bit (page 980), setting it to 2 specifies setgid permissions, and setting it to 4 specifies setuid permissions: $ ls -l myprog* -rwxr-xr-x 1 root pubs 19704 Jul 31 14:30 myprog1 -rwxr-xr-x 1 root pubs 19704 Jul 31 14:30 myprog2 # chmod 4755 myprog1 # chmod 2755 myprog2 $ ls -l myprog* -rwsr-xr-x 1 root pubs 19704 Jul 31 14:30 myprog1 -rwxr-sr-x 1 root pubs 19704 Jul 31 14:30 myprog2 Do not write setuid shell scripts security Never give shell scripts setuid permission. Several techniques for subverting them are well known. 98 Chapter 4 The Filesystem Directory Access Permissions Access permissions have slightly different meanings when they are used with directories. Although the three types of users can read from or write to a directory, the directory cannot be executed. Execute permission is redefined for a directory: It means that you can cd into the directory and/or examine files that you have permission to read from in the directory. It has nothing to do with executing a file. When you have only execute permission for a directory, you can use ls to list a file in the directory if you know its name. You cannot use ls without an argument to list the entire contents of the directory. In the following exchange, Zach first verifies that he is logged in as himself. He then checks the permissions on Max's info directory. You can view the access permissions associated with a directory by running ls with the –d (directory) and –l (long) options: $ who am i zach pts/7 Aug 21 10:02 $ ls -ld /home/max/info drwx-----x 2 max pubs 512 Aug 21 09:31 /home/max/info $ ls -l /home/max/info ls: /home/max/info: Permission denied The d at the left end of the line that ls displays indicates that /home/max/info is a directory. Max has read, write, and execute permissions; members of the pubs group have no access permissions; and other users have execute permission only, indicated by the x at the right end of the permissions. Because Zach does not have read permission for the directory, the ls –l command returns an error. When Zach specifies the names of the files he wants information about, he is not reading new directory information but rather searching for specific information, which he is allowed to do with execute access to the directory. He has read permission for notes so he has no problem using cat to display the file. He cannot display financial because he does not have read permission for it: $ ls -l /home/max/info/financial /home/max/info/notes -rw------1 max pubs 34 Aug 21 09:31 /home/max/info/financial -rw-r--r-1 max pubs 30 Aug 21 09:32 /home/max/info/notes $ cat /home/max/info/notes This is the file named notes. $ cat /home/max/info/financial cat: /home/max/info/financial: Permission denied Next Max gives others read access to his info directory: $ chmod o+r /home/max/info When Zach checks his access permissions on info, he finds that he has both read and execute access to the directory. Now ls –l can list the contents of the info directory, but Zach still cannot read financial. (This restriction is an issue of file permissions, not directory permissions.) Finally, Zach tries to create a file named newfile using ACLs: Access Control Lists 99 touch (page 862). If Max were to give him write permission to the info directory, Zach would be able to create new files in it: $ ls -ld /home/max/info drwx---r-x 2 max pubs 512 Aug 21 09:31 /home/max/info $ ls -l /home/max/info total 8 -rw------1 max pubs 34 Aug 21 09:31 financial -rw-r--r-1 max pubs 30 Aug 21 09:32 notes $ cat /home/max/info/financial cat: financial: Permission denied $ touch /home/max/info/newfile touch: cannot touch '/home/max/info/newfile': Permission denied ACLs: Access Control Lists Access Control Lists (ACLs) provide finer-grained control over which users can access specific directories and files than do traditional Linux permissions (page 93). Using ACLs you can specify the ways in which each of several users can access a directory or file. Because ACLs can reduce performance, do not enable them on filesystems that hold system files, where the traditional Linux permissions are sufficient. Also be careful when moving, copying, or archiving files: Not all utilities preserve ACLs. In addition, you cannot copy ACLs to filesystems that do not support ACLs. An ACL comprises a set of rules. A rule specifies how a specific user or group can access the file that the ACL is associated with. There are two kinds of rules: access rules and default rules. (The documentation refers to access ACLs and default ACLs, even though there is only one type of ACL: There is one type of list [ACL] and there are two types of rules that an ACL can contain.) An access rule specifies access information for a single file or directory. A default ACL pertains to a directory only; it specifies default access information (an ACL) for any file in the directory that is not given an explicit ACL. Most utilities do not preserve ACLs caution When used with the –p (preserve) or –a (archive) option, cp preserves ACLs when it copies files. The mv utility also preserves ACLs. When you use cp with the –p or –a option and it is not able to copy ACLs, and in the case where mv is unable to preserve ACLs, the utility performs the operation and issues an error message: $ mv report /tmp mv: preserving permissions for '/tmp/report': Operation not supported Other utilities, such as tar, cpio, and dump, do not support ACLs. You can use cp with the –a option to copy directory hierarchies, including ACLs. You can never copy ACLs to a filesystem that does not support ACLs or to a filesystem that does not have ACL support turned on. 100 Chapter 4 The Filesystem Enabling ACLs The following explanation of how to enable ACLs pertains to Linux. See page 933 if you are running Mac OS X. Before you can use ACLs you must install the acl software package as explained in Appendix C. Linux officially supports ACLs on ext2, ext3, and ext4 filesystems only, although informal support for ACLs is available on other filesystems. To use ACLs on an ext filesystem, you must mount the device with the acl option (no_acl is the default). For example, if you want to mount the device represented by /home so you can use ACLs on files in /home, you can add acl to its options list in /etc/fstab: $ grep home /etc/fstab LABEL=/home /home ext3 defaults,acl 1 2 After changing fstab, you need to remount /home before you can use ACLs. If no one else is using the system, you can unmount it and mount it again (working with root privileges) as long as the working directory is not in the /home hierarchy. Alternatively you can use the remount option to mount to remount /home while the device is in use: # mount -v -o remount /home /dev/hda3 on /home type ext3 (rw,acl) Working with Access Rules The setfacl utility modifies a file's ACL and getfacl displays a file's ACL. These utilities are available under Linux only. If you are running OS X you must use chmod as explained on page 933. When you use getfacl to obtain information about a file that does not have an ACL, it displays the same information as an ls –l command, albeit in a different format: $ ls -l report -rw-r--r-- 1 max max 9537 Jan 12 23:17 report $ getfacl report # file: report # owner: max # group: max user::rwgroup::r-other::r-- The first three lines of the getfacl output comprise the header; they specify the name of the file, the owner of the file, and the group the file is associated with. For more information refer to "ls –l: Displays Permissions" on page 93. The ––omit-header (or just ––omit) option causes getfacl not to display the header: $ getfacl --omit-header report user::rwgroup::r-other::r-- ACLs: Access Control Lists 101 In the line that starts with user, the two colons (::) with no name between them indicate that the line specifies the permissions for the owner of the file. Similarly, the two colons in the group line indicate that the line specifies permissions for the group the file is associated with. The two colons following other are there for consistency: No name can be associated with other. The setfacl ––modify (or –m) option adds or modifies one or more rules in a file's ACL using the following format: setfacl ––modify ugo:name:permissions file-list where ugo can be either u, g, or o to indicate that the command sets file permissions for a user, a group, or all other users, respectively; name is the name of the user or group that permissions are being set for; permissions is the permissions in either symbolic or absolute format; and file-list is the list of files the permissions are to be applied to. You must omit name when you specify permissions for other users (o). Symbolic permissions use letters to represent file permissions (rwx, r–x, and so on), whereas absolute permissions use an octal number. While chmod uses three sets of permissions or three octal numbers (one each for the owner, group, and other users), setfacl uses a single set of permissions or a single octal number to represent the permissions being granted to the user or group represented by ugo and name. See the discussion of chmod on page 94 for more information about symbolic and absolute representations of file permissions. For example, both of the following commands add a rule to the ACL for the report file that gives Sam read and write permission to that file: $ setfacl --modify u:sam:rw- report or $ setfacl --modify u:sam:6 report $ getfacl report # file: report # owner: max # group: max user::rwuser:sam:rwgroup::r-mask::rwother::r-- The line containing user:sam:rw– shows that Sam has read and write access (rw–) to the file. See page 93 for an explanation of how to read access permissions. See the following optional section for a description of the line that starts with mask. When a file has an ACL, ls –l displays a plus sign (+) following the permissions, even if the ACL is empty: $ ls -l report -rw-rw-r--+ 1 max max 9537 Jan 12 23:17 report 102 Chapter 4 The Filesystem optional Effective Rights Mask The line that starts with mask specifies the effective rights mask. This mask limits the effective permissions granted to ACL groups and users. It does not affect the owner of the file or the group the file is associated with. In other words, it does not affect traditional Linux permissions. However, because setfacl always sets the effective rights mask to the least restrictive ACL permissions for the file, the mask has no effect unless you set it explicitly after you set up an ACL for the file. You can set the mask by specifying mask in place of ugo and by not specifying a name in a setfacl command. The following example sets the effective rights mask to read for the report file: $ setfacl -m mask::r-- report The mask line in the following getfacl output shows the effective rights mask set to read (r––). The line that displays Sam's file access permissions shows them still set to read and write. However, the comment at the right end of the line shows that his effective permission is read. $ getfacl report # file: report # owner: max # group: max user::rwuser:sam:rwgroup::r-mask::r-other::r-- #effective:r-- As the next example shows, setfacl can modify ACL rules and can set more than one ACL rule at a time: $ setfacl -m u:sam:r--,u:zach:rw- report $ getfacl --omit-header report user::rwuser:sam:r-user:zach:rwgroup::r-mask::rwother::r-- The –x option removes ACL rules for a user or a group. It has no effect on permissions for the owner of the file or the group that the file is associated with. The next example shows setfacl removing the rule that gives Sam permission to access the file: $ setfacl -x u:sam report $ getfacl --omit-header report user::rwuser:zach:rwgroup::r-mask::rwother::r-- ACLs: Access Control Lists 103 You must not specify permissions when you use the –x option. Instead, specify only the ugo and name. The –b option, followed by a filename only, removes all ACL rules and the ACL itself from the file or directory you specify. Both setfacl and getfacl have many options. Use the ––help option to display brief lists of options or refer to the man pages for details. Setting Default Rules for a Directory The following example shows that the dir directory initially has no ACL. The setfacl command uses the –d (default) option to add two default rules to the ACL for dir. These rules apply to all files in the dir directory that do not have explicit ACLs. The rules give members of the pubs group read and execute permissions and give members of the admin group read, write, and execute permissions. $ ls -ld dir drwx------ 2 max max 4096 Feb 12 23:15 dir $ getfacl dir # file: dir # owner: max # group: max user::rwx group::--other::--$ setfacl -d -m g:pubs:r-x,g:admin:rwx dir The following ls command shows that the dir directory now has an ACL, as indicated by the + to the right of the permissions. Each of the default rules that getfacl displays starts with default:. The first two default rules and the last default rule specify the permissions for the owner of the file, the group that the file is associated with, and all other users. These three rules specify the traditional Linux permissions and take precedence over other ACL rules. The third and fourth rules specify the permissions for the pubs and admin groups. Next is the default effective rights mask. $ ls -ld dir drwx------+ 2 max max 4096 Feb 12 23:15 dir $ getfacl dir # file: dir # owner: max # group: max user::rwx group::--other::--default:user::rwx default:group::--default:group:pubs:r-x default:group:admin:rwx default:mask::rwx default:other::--- 104 Chapter 4 The Filesystem Remember that the default rules pertain to files held in the directory that are not assigned ACLs explicitly. You can also specify access rules for the directory itself. When you create a file within a directory that has default rules in its ACL, the effective rights mask for that file is created based on the file's permissions. In some cases the mask may override default ACL rules. In the next example, touch creates a file named new in the dir directory. The ls command shows that this file has an ACL. Based on the value of umask (see the bash man page), both the owner and the group that the file is associated with have read and write permissions for the file. The effective rights mask is set to read and write so that the effective permission for pubs is read and the effective permissions for admin are read and write. Neither group has execute permission. $ cd dir $ touch new $ ls -l new -rw-rw----+ 1 max max 0 Feb 13 00:39 new $ getfacl --omit new user::rwgroup::--group:pubs:r-x #effective:r-group:admin:rwx #effective:rwmask::rwother::--- If you change the file's traditional permissions to read, write, and execute for the owner and the group, the effective rights mask changes to read, write, and execute and the groups specified by the default rules gain execute access to the file. $ chmod 770 new $ ls -l new -rwxrwx---+ 1 max max 0 Feb 13 00:39 new $ getfacl --omit new user::rwx group::--group:pubs:r-x group:admin:rwx mask::rwx other::--- Links A link is a pointer to a file. Each time you create a file using vim, touch, cp, or by another other means, you are putting a pointer in a directory. This pointer associates a filename with a place on the disk. When you specify a filename in a command, you are indirectly pointing to the place on the disk that holds the information you want. Links 105 correspond personal memos business to_do to_do to_do to_do personal memos business Links Figure 4-13 Using links to cross-classify files Sharing files can be useful when two or more people are working on the same project and need to share some information. You can make it easy for other users to access one of your files by creating additional links to the file. To share a file with another user, first give the user permission to read from and write to the file (page 94). You may also have to change the access permissions of the parent directory of the file to give the user read, write, or execute permission (page 98). Once the permissions are appropriately set, the user can create a link to the file so that each of you can access the file from your separate directory hierarchies. A link can also be useful to a single user with a large directory hierarchy. You can create links to cross-classify files in your directory hierarchy, using different classifications for different tasks. For example, if you have the file layout depicted in Figure 4-2 on page 79, a file named to_do might appear in each subdirectory of the correspond directory—that is, in personal, memos, and business. If you find it difficult to keep track of everything you need to do, you can create a separate directory named to_do in the correspond directory. You can then link each subdirectory's to-do list into that directory. For example, you could link the file named to_do in the memos directory to a file named memos in the to_do directory. This set of links is shown in Figure 4-13. Although it may sound complicated, this technique keeps all your to-do lists conveniently in one place. The appropriate list is easily accessible in the task-related directory when you are busy composing letters, writing memos, or handling personal business. About the discussion of hard links tip Two kinds of links exist: hard links and symbolic (soft) links. Hard links are older and becoming outdated. The section on hard links is marked as optional; you can skip it, although it discusses inodes and gives you insight into the structure of the filesystem. 106 Chapter 4 The Filesystem optional Hard Links A hard link to a file appears as another file. If the file appears in the same directory as the linked-to file, the links must have different filenames because two files in the same directory cannot have the same name. You can create a hard link to a file only from within the filesystem that holds the file. ln: Creates a Hard Link The ln (link) utility (without the –s or ––symbolic option) creates a hard link to an existing file using the following syntax: ln existing-file new-link The next command shows Zach making the link shown in Figure 4-14 by creating a new link named /home/max/letter to an existing file named draft in Zach's home directory: $ pwd /home/zach $ ln draft /home/max/letter The new link appears in the /home/max directory with the filename letter. In practice, Max may need to change directory access permissions so Zach will be able to create the link. Even though /home/max/letter appears in Max's directory, Zach is the owner of the file because he created it. The ln utility creates an additional pointer to an existing file but it does not make another copy of the file. Because there is only one file, the file status information—such as access permissions, owner, and the time the file was last modified—is the same for all links; only the filenames differ. When Zach modifies /home/zach/draft, for example, Max sees the changes in /home/max/letter. / home max zach memo planning /home/max/letter and /home/zach/draft Figure 4-14 Two links to the same file: /home/max/letter and /home/zach/draft Links 107 cp Versus ln The following commands verify that ln does not make an additional copy of a file. Create a file, use ln to make an additional link to the file, change the contents of the file through one link, and verify the change through the other link: $ cat file_a This is file A. $ ln file_a file_b $ cat file_b This is file A. $ vim file_b ... $ cat file_b This is file B after the change. $ cat file_a This is file B after the change. If you try the same experiment using cp instead of ln and change a copy of the file, the difference between the two utilities will become clearer. Once you change a copy of a file, the two files are different: $ cat file_c This is file C. $ cp file_c file_d $ cat file_d This is file C. $ vim file_d ... $ cat file_d This is file D after the change. $ cat file_c This is file C. ls and link counts You can use ls with the –l option, followed by the names of the files you want to compare, to confirm that the status information is the same for two links to the same file and is different for files that are not linked. In the following example, the 2 in the links field (just to the left of max) shows there are two links to file_a and file_b (from the previous example): $ ls -l file_a file_b -rw-r--r-- 2 max pubs -rw-r--r-- 2 max pubs -rw-r--r-- 1 max pubs -rw-r--r-- 1 max pubs file_c file_d 33 May 24 10:52 33 May 24 10:52 16 May 24 10:55 33 May 24 10:57 file_a file_b file_c file_d Although it is easy to guess which files are linked to one another in this example, ls does not explicitly tell you. ls and inodes Use ls with the –i option to determine without a doubt which files are linked. The –i option lists the inode (page 959) number for each file. An inode is the control structure for a file. (HFS+, the default filesystem under Mac OS X, does not have inodes but, through an elaborate scheme, appears to have inodes.) If the two filenames have 108 Chapter 4 The Filesystem the same inode number, they share the same control structure and are links to the same file. Conversely, when two filenames have different inode numbers, they are different files. The following example shows that file_a and file_b have the same inode number and that file_c and file_d have different inode numbers: $ ls -i file_a file_b file_c file_d 3534 file_a 3534 file_b 5800 file_c 7328 file_d All links to a file are of equal value: The operating system cannot distinguish the order in which multiple links were created. When a file has two links, you can remove either one and still access the file through the remaining link. You can remove the link used to create the file, for example, and, as long as one link remains, still access the file through that link. Symbolic Links In addition to hard links, Linux supports symbolic links, also called soft links or symlinks. A hard link is a pointer to a file (the directory entry points to the inode), whereas a symbolic link is an indirect pointer to a file (the directory entry contains the pathname of the pointed-to file—a pointer to the hard link to the file). Advantages of symbolic links Symbolic links were developed because of the limitations inherent in hard links. You cannot create a hard link to a directory, but you can create a symbolic link to a directory. In most cases the Linux file hierarchy encompasses several filesystems. Because each filesystem keeps separate control information (that is, separate inode tables or filesystem structures) for the files it holds, it is not possible to create hard links between files in different filesystems. A symbolic link can point to any file, regardless of where it is located in the file structure, but a hard link to a file must be in the same filesystem as the other hard link(s) to the file. When you create links only among files in your home directory, you will not notice this limitation. A major advantage of a symbolic link is that it can point to a nonexistent file. This ability is useful if you need a link to a file that is periodically removed and recreated. A hard link keeps pointing to a "removed" file, which the link keeps alive even after a new file is created. In contrast, a symbolic link always points to the newly created file and does not interfere when you delete the old file. For example, a symbolic link could point to a file that gets checked in and out under a source code control system, a .o file that is re-created by the C compiler each time you run make, or a log file that is repeatedly archived. Although they are more general than hard links, symbolic links have some disadvantages. Whereas all hard links to a file have equal status, symbolic links do not have the same status as hard links. When a file has multiple hard links, it is analogous to a person having multiple full legal names, as many married women do. In contrast, symbolic links are analogous to nicknames. Anyone can have one or more nicknames, but these nicknames have a lesser status than legal names. The Links 109 following sections describe some of the peculiarities of symbolic links. See page 932 for information on Mac OS X Finder aliases. ln: Creates Symbolic Links The ln utility with the ––symbolic (or –s) option creates a symbolic link. The following example creates a symbolic link /tmp/s3 to the file sum in Max's home directory. When you use an ls –l command to look at the symbolic link, ls displays the name of the link and the name of the file it points to. The first character of the listing is l (for link). $ ln --symbolic /home/max/sum /tmp/s3 $ ls -l /home/max/sum /tmp/s3 -rw-rw-r-1 max max 38 Jun 12 09:51 /home/max/sum lrwxrwxrwx 1 max max 14 Jun 12 09:52 /tmp/s3 -> /home/max/sum $ cat /tmp/s3 This is sum. The sizes and times of the last modifications of the two files are different. Unlike a hard link, a symbolic link to a file does not have the same status information as the file itself. You can also use ln to create a symbolic link to a directory. When you use the ––symbolic option, ln works as expected whether the file you are creating a link to is an ordinary file or a directory. Use absolute pathnames with symbolic links tip Symbolic links are literal and are not aware of directories. A link that points to a relative pathname, including a simple filename, assumes the relative pathname is relative to the directory that the link was created in (not the directory the link was created from). In the following example, the link points to the file named sum in the /tmp directory. Because no such file exists, cat gives an error message: $ pwd /home/max $ ln --symbolic sum /tmp/s4 $ ls -l sum /tmp/s4 lrwxrwxrwx 1 max max 3 Jun 12 10:13 /tmp/s4 -> sum -rw-rw-r-1 max max 38 Jun 12 09:51 sum $ cat /tmp/s4 cat: /tmp/s4: No such file or directory optional cd and Symbolic Links When you use a symbolic link as an argument to cd to change directories, the results can be confusing, particularly if you did not realize that you were using a symbolic link. If you use cd to change to a directory that is represented by a symbolic link, the pwd shell builtin (page 141) lists the name of the symbolic link. The pwd utility 110 Chapter 4 The Filesystem (/bin/pwd) lists the name of the linked-to directory, not the link, regardless of how you got there. $ ln -s /home/max/grades /tmp/grades.old $ pwd /home/max $ cd /tmp/grades.old $ pwd /tmp/grades.old $ /bin/pwd /home/max/grades When you change directories back to the parent, you end up in the directory holding the symbolic link: $ cd .. $ pwd /tmp $ /bin/pwd /tmp Under Mac OS X, /tmp is a symbolic link to /private/tmp. When you are running OS X, after you give the cd .. command in the previous example, the working directory is /private/tmp. rm: Removes a Link When you create a file, there is one hard link to it. You can then delete the file or, using Linux terminology, remove the link with the rm utility. When you remove the last hard link to a file, the operating system releases the space the file occupied on the disk and you can no longer access the information that was stored in the file. This space is released even if symbolic links to the file remain. When there is more than one hard link to a file, you can remove a hard link and still access the file from any remaining link. Unlike DOS and Windows, Linux does not provide an easy way to undelete a file once you have removed it. A skilled hacker, however, can sometimes piece the file together with time and effort. When you remove all hard links to a file, you will not be able to access the file through a symbolic link. In the following example, cat reports that the file total does not exist because it is a symbolic link to a file that has been removed: $ ls -l sum -rw-r--r-- 1 max pubs 981 May 24 11:05 sum $ ln -s sum total $ rm sum $ cat total cat: total: No such file or directory $ ls -l total lrwxrwxrwx 1 max pubs 6 May 24 11:09 total -> sum Chapter Summary 111 When you remove a file, be sure to remove all symbolic links to it. Remove a symbolic link in the same way you remove other files: $ rm total Chapter Summary Linux has a hierarchical, or treelike, file structure that makes it possible to organize files so you can find them quickly and easily. The file structure contains directory files and ordinary files. Directories contain other files, including other directories; ordinary files generally contain text, programs, or images. The ancestor of all files is the root directory and is represented by / standing alone or at the left end of a pathname. Most Linux filesystems support 255-character filenames. Nonetheless, it is a good idea to keep filenames simple and intuitive. Filename extensions can help make filenames more meaningful. When you are logged in, you are always associated with a working directory. Your home directory is the working directory from the time you log in until you use cd to change directories. An absolute pathname starts with the root directory and contains all the filenames that trace a path to a given file. The pathname starts with a slash, representing the root directory, and contains additional slashes following all the directories in the path, except for the last directory in the case of a path that points to a directory file. A relative pathname is similar to an absolute pathname but traces the path starting from the working directory. A simple filename is the last element of a pathname and is a form of a relative pathname; it represents a file in the working directory. A Linux filesystem contains many important directories, including /usr/bin, which stores most of the Linux utility commands, and /dev, which stores device files, many of which represent physical pieces of hardware. An important standard file is /etc/passwd; it contains information about users, such as each user's ID and full name. Among the attributes associated with each file are access permissions. They determine who can access the file and how the file may be accessed. Three groups of users can potentially access the file: the owner, the members of a group, and all other users. An ordinary file can be accessed in three ways: read, write, and execute. The ls utility with the –l option displays these permissions. For directories, execute access is redefined to mean that the directory can be searched. The owner of a file or a user working with root privileges can use the chmod utility to change the access permissions of a file. This utility specifies read, write, and execute permissions for the file's owner, the group, and all other users on the system. 112 Chapter 4 The Filesystem Access Control Lists (ACLs) provide finer-grained control over which users can access specific directories and files than do traditional Linux permissions. Using ACLs you can specify the ways in which each of several users can access a directory or file. Few utilities preserve ACLs when working with files. An ordinary file stores user data, such as textual information, programs, or images. A directory is a standard-format disk file that stores information, including names, about ordinary files and other directory files. An inode is a data structure, stored on disk, that defines a file's existence and is identified by an inode number. A directory relates each of the filenames it stores to an inode. A link is a pointer to a file. You can have several links to a file so you can share the file with other users or have the file appear in more than one directory. Because only one copy of a file with multiple links exists, changing the file through any one link causes the changes to appear in all the links. Hard links cannot link directories or span filesystems, whereas symbolic links can. Table 4-2 summarizes the utilities introduced in this chapter. Table 4-2 Utilities introduced in Chapter 4 Utility Function cd Associates you with another working directory (page 87) chmod Changes access permissions on a file (page 94) getfacl Displays a file's ACL (page 100) ln Makes a link to an existing file (page 106) mkdir Creates a directory (page 86) pwd Displays the pathname of the working directory (page 82) rmdir Deletes a directory (page 88) setfacl Modifies a file's ACL (page 100)

Exercises 1. Is each of the following an absolute pathname, a relative pathname, or a simple filename? a. milk_co b. correspond/business/milk_co Exercises 113 c. /home/max d. /home/max/literature/promo e. .. f. letter.0610 2. List the commands you can use to perform these operations: a. Make your home directory the working directory b. Identify the working directory 3. If the working directory is /home/max with a subdirectory named literature, give three sets of commands that you can use to create a subdirectory named classics under literature. Also give several sets of commands that you can use to remove the classics directory and its contents. 4. The df utility displays all mounted filesystems along with information about each. Use the df utility with the –h (human-readable) option to answer the following questions. a. How many filesystems are mounted on your Linux system? b. Which filesystem stores your home directory? c. Assuming that your answer to exercise 4a is two or more, attempt to create a hard link to a file on another filesystem. What error message do you get? What happens when you attempt to create a symbolic link to the file instead? 5. Suppose you have a file that is linked to a file owned by another user. How can you ensure that changes to the file are no longer shared? 6. You should have read permission for the /etc/passwd file. To answer the following questions, use cat or less to display /etc/passwd. Look at the fields of information in /etc/passwd for the users on your system. a. Which character is used to separate fields in /etc/passwd? b. How many fields are used to describe each user? c. How many users are on the local system? d. How many different login shells are in use on your system? (Hint: Look at the last field.) e. The second field of /etc/passwd stores user passwords in encoded form. If the password field contains an x, your system uses shadow passwords and stores the encoded passwords elsewhere. Does your system use shadow passwords? 114 Chapter 4 The Filesystem 7. If /home/zach/draft and /home/max/letter are links to the same file and the following sequence of events occurs, what will be the date in the opening of the letter? a. Max gives the command vim letter. b. Zach gives the command vim draft. c. Zach changes the date in the opening of the letter to January 31, 2009, writes the file, and exits from vim. d. Max changes the date to February 1, 2009, writes the file, and exits from vim. 8. Suppose a user belongs to a group that has all permissions on a file named jobs_list, but the user, as the owner of the file, has no permissions. Describe which operations, if any, the user/owner can perform on jobs_list. Which command can the user/owner give that will grant the user/owner all permissions on the file? 9. Does the root directory have any subdirectories you cannot search as an ordinary user? Does the root directory have any subdirectories you cannot read as a regular user? Explain. 10. Assume you are given the directory structure shown in Figure 4-2 on page 79 and the following directory permissions: d--x--x--drwxr-xr-x 3 zach pubs 512 Mar 10 15:16 business 2 zach pubs 512 Mar 10 15:16 business/milk_co For each category of permissions—owner, group, and other—what happens when you run each of the following commands? Assume the working directory is the parent of correspond and that the file cheese_co is readable by everyone. a. cd correspond/business/milk_co b. ls –l correspond/business c. cat correspond/business/cheese_co Advanced Exercises 11. What is an inode? What happens to the inode when you move a file within a filesystem? 12. What does the .. entry in a directory point to? What does this entry point to in the root (/) directory? 13. How can you create a file named –i? Which techniques do not work, and why do they not work? How can you remove the file named –i? Advanced Exercises 14. Suppose the working directory contains a single file named andor. What error message do you get when you run the following command line? $ mv andor and\/or Under what circumstances is it possible to run the command without producing an error? 15. The ls –i command displays a filename preceded by the inode number of the file (page 107). Write a command to output inode/filename pairs for the files in the working directory, sorted by inode number. (Hint: Use a pipe.) 16. Do you think the system administrator has access to a program that can decode user passwords? Why or why not? (See exercise 6.) 17. Is it possible to distinguish a file from a hard link to a file? That is, given a filename, can you tell whether it was created using an ln command? Explain. 18. Explain the error messages displayed in the following sequence of commands: $ ls -l total 1 drwxrwxr-x 2 max pubs 1024 Mar $ ls dirtmp $ rmdir dirtmp rmdir: dirtmp: Directory not empty $ rm dirtmp/* rm: No match. 2 17:57 dirtmp 115 This page intentionally left blank 5 The Shell In This Chapter This chapter takes a close look at the shell and explains how to use some of its features. It discusses command-line syntax and describes how the shell processes a command line and initiates execution of a program. In addition the chapter explains how to redirect input to and output from a command, construct pipes and filters on the command line, and run a command in the background. The final section covers filename expansion and explains how you can use this feature in your everyday work. 5Chapter5 Except as noted, everything in this chapter applies to the Bourne Again (bash) and TC (tcsh) Shells. The exact wording of the shell output differs from shell to shell: What the shell you are using displays may differ slightly from what appears in this book. For shell-specific information, refer to Chapters 8 (bash) and 9 (tcsh). Chapter 10 covers writing and executing bash shell scripts. 117 118 Chapter 5 The Shell The Command Line The shell executes a program when you give it a command in response to its prompt. For example, when you give an ls command, the shell executes the utility program named ls. You can cause the shell to execute other types of programs— such as shell scripts, application programs, and programs you have written—in the same way. The line that contains the command, including any arguments, is called the command line. This book uses the term command to refer to both the characters you type on the command line and the program that action invokes. Syntax Command-line syntax dictates the ordering and separation of the elements on a command line. When you press the RETURN key after entering a command, the shell scans the command line for proper syntax. The syntax for a basic command line is command [arg1] [arg2] ... [argn] RETURN One or more SPACEs must separate elements on the command line. The command is the name of the command, arg1 through argn are arguments, and RETURN is the keystroke that terminates all command lines. The brackets in the command-line syntax indicate that the arguments they enclose are optional. Not all commands require arguments: Some commands do not allow arguments; other commands allow a variable number of arguments; and still others require a specific number of arguments. Options, a special kind of argument, are usually preceded by one or two hyphens (also called a dash or minus sign: –). Command Name Usage message Some useful Linux command lines consist of only the name of the command without any arguments. For example, ls by itself lists the contents of the working directory. Commands that require arguments typically give a short error message, called a usage message, when you use them without arguments, with incorrect arguments, or with the wrong number of arguments. Arguments On the command line each sequence of nonblank characters is called a token or word. An argument is a token, such as a filename, string of text, number, or other object that a command acts on. For example, the argument to a vim or emacs command is the name of the file you want to edit. The following command line shows cp copying the file named temp to tempcopy: $ cp temp tempcopy Arguments are numbered starting with the command itself, which is argument zero. In this example, cp is argument zero, temp is argument one, and tempcopy is argument two. The cp utility requires at least two arguments on the command line; see page 640. Argument one is the name of an existing file. Argument two is the name of the file that cp is creating or overwriting. Here the arguments are not optional; both arguments must be present for the command to work. When you do The Command Line 119 $ ls hold mark names oldstuff temp zach house max office personal test $ ls -r zach temp oldstuff names mark hold test personal office max house $ ls -x hold house mark max names office oldstuff personal temp test zach $ ls -rx zach test temp personal oldstuff office names max mark house hold Figure 5-1 Using options not supply the right number or kind of arguments, cp displays a usage message. Try typing cp and then pressing RETURN. Options An option is an argument that modifies the effects of a command. You can frequently specify more than one option, modifying the command in several different ways. Options are specific to and interpreted by the program the command line calls, not by the shell. By convention options are separate arguments that follow the name of the command and usually precede other arguments, such as filenames. Most utilities require you to prefix options with a single hyphen. However, this requirement is specific to the utility and not the shell. GNU program options are frequently preceded by two hyphens in a row. For example, ––help generates a (sometimes extensive) usage message. Figure 5-1 first shows the output of an ls command without any options. By default ls lists the contents of the working directory in alphabetical order, vertically sorted in columns. Next the –r (reverse order; because this is a GNU utility, you can also use ––reverse) option causes the ls utility to display the list of files in reverse alphabetical order, still sorted in columns. The –x option causes ls to display the list of files in horizontally sorted rows. Combining options When you need to use several options, you can usually group multiple single-letter options into one argument that starts with a single hyphen; do not put SPACEs between the options. You cannot combine options that are preceded by two hyphens in this way. Specific rules for combining options depend on the program you are running. Figure 5-1 shows both the –r and –x options with the ls utility. Together these options generate a list of filenames in horizontally sorted columns, in reverse alphabetical order. Most utilities allow you to list options in any order; thus ls –xr produces the same results as ls –rx. The command ls –x –r also generates the same list. Option arguments Some utilities have options that themselves require arguments. For example, the gcc utility has a –o option that must be followed by the name you want to give the executable file that gcc generates. Typically an argument to an option is separated from its option letter by a SPACE: $ gcc -o prog prog.c 120 Chapter 5 The Shell Displaying readable file sizes: the –h option tip Most utilities that report on file sizes specify the size of a file in bytes. Bytes work well when you are dealing with smaller files, but the numbers can be difficult to read when you are working with file sizes that are measured in megabytes or gigabytes. Use the –h (or ––human-readable) option to display file sizes in kilo-, mega-, and gigabytes. Experiment with the df –h (disk free) and ls –lh commands. Arguments that start with a hyphen Another convention allows utilities to work with arguments, such as filenames, that start with a hyphen. If a file's name is –l, the following command is ambiguous: $ ls -l This command could mean you want ls to display a long listing of all files in the working directory or a listing of the file named –l. It is interpreted as the former. Avoid creating files whose names begin with hyphens. If you do create them, many utilities follow the convention that a –– argument (two consecutive hyphens) indicates the end of the options (and the beginning of the arguments). To disambiguate the preceding command, you can type $ ls -- -l You can use an alternative format in which the period refers to the working directory and the slash indicates that the name refers to a file in the working directory: $ ls ./-l Assuming you are working in the /home/max directory, the preceding command is functionally equivalent to $ ls /home/max/-l The following command displays a long listing of this file: $ ls -l -- -l These are conventions, not hard-and-fast rules, and a number of utilities do not follow them (e.g., find). Following such conventions is a good idea and makes it easier for users to work with your program. When you write shell scripts that require options, follow the Linux option conventions. You can use xargs (page 881) in a shell script to help follow these conventions. Processing the Command Line As you enter a command line, the Linux tty device driver (part of the Linux kernel) examines each character to see whether it must take immediate action. When you press CONTROL-H (to erase a character) or CONTROL-U (to kill a line), the device driver immediately adjusts the command line as required; the shell never sees the character(s) you erased or the line you killed. Often a similar adjustment occurs when you press CONTROL-W (to erase a word). When the character you entered does not require immediate action, the device driver stores the character in a buffer and waits for additional characters. When you press RETURN, the device driver passes the command line to the shell for processing. Parsing the command line When the shell processes a command line, it looks at the line as a whole and parses (breaks) it into its component parts (Figure 5-2). Next the shell looks for the name of the command. Usually the name of the command is the first item on the command line The Command Line 121 The ––help option tip Many utilities display a (sometimes extensive) help message when you call them with an argument of ––help. All utilities developed by the GNU Project (page 3) accept this option. An example follows. $ bzip2 --help bzip2, a block-sorting file compressor. Version 1.0.5, 10-Dec-2007. usage: bunzip2 [flags and input files in any order] -h -d -z -k -f ... If --help --decompress --compress --keep --force print this message force decompression force compression keep (don't delete) input files overwrite existing output files invoked as 'bzip2', default action is to compress. as 'bunzip2', default action is to decompress. as 'bzcat', default action is to decompress to stdout. ... after the prompt (argument zero). The shell typically takes the first characters on the command line up to the first blank (TAB or SPACE) and then looks for a command with that name. The command name (the first token) can be specified on the command line Get first word and save as command name NEWLINE Execute program yes no Get more of the command line no Display not found Does program exist? Issue prompt Figure 5-2 Processing the command line 122 Chapter 5 The Shell either as a simple filename or as a pathname. For example, you can call the ls command in either of the following ways: $ ls $ /bin/ls optional The shell does not require that the name of the program appear first on the command line. Thus you can structure a command line as follows: $ >bb) instructs the shell to redirect the output of a command to the specified file instead of to the screen (Figure 5-6). The format of a command line that redirects output is command [arguments] > filename where command is any executable program (such as an application program or a utility), arguments are optional arguments, and filename is the name of the ordinary file the shell redirects the output to. Figure 5-7 uses cat to demonstrate output redirection. This figure contrasts with Figure 5-5, where standard input and standard output are associated with the keyboard and screen. The input in Figure 5-7 comes from the keyboard. The redirect output symbol on the command line causes the shell to associate cat's standard output with the sample.txt file specified on the command line. After giving the command and typing the text shown in Figure 5-7, the sample.txt file contains the text you entered. You can use cat with an argument of sample.txt to display this file. The next section shows another way to use cat to display the file. Figure 5-7 shows that redirecting standard output from cat is a handy way to create a file without using an editor. The drawback is that once you enter a line Standard Input and Standard Output 127 $ cat > sample.txt This text is being entered at the keyboard and cat is copying it to a file. Press CONTROL-D to signal the end of file. CONTROL-D $ Figure 5-7 cat with its output redirected Redirecting output can destroy a file I caution Use caution when you redirect output to a file. If the file exists, the shell will overwrite it and destroy its contents. For more information see the tip "Redirecting output can destroy a file II" on page 129. and press RETURN, you cannot edit the text. While you are entering a line, the erase and kill keys work to delete text. This procedure is useful for creating short, simple files. Figure 5-8 shows how to use cat and the redirect output symbol to catenate (join one after the other—the derivation of the name of the cat utility) several files into one larger file. The first three commands display the contents of three files: stationery, tape, and pens. The next command shows cat with three filenames as arguments. When you call it with more than one filename, cat copies the files, one at a time, to standard output. This command redirects standard output to the file supply_orders. The final cat command shows that supply_orders holds the contents of all three original files. $ cat stationery 2,000 sheets letterhead ordered: 10/7/08 $ cat tape 1 box masking tape ordered: 5 boxes filament tape ordered: 10/14/08 10/28/08 $ cat pens 12 doz. black pens ordered: 10/4/08 $ cat stationery tape pens > supply_orders $ cat supply_orders 2,000 sheets letterhead ordered: 1 box masking tape ordered: 5 boxes filament tape ordered: 12 doz. black pens ordered: $ Figure 5-8 Using cat to catenate files 10/7/08 10/14/08 10/28/08 10/4/08 128 Chapter 5 The Shell Sh e ll Standard input Shell File Standard output Command Figure 5-9 Redirecting standard input Redirecting Standard Input Just as you can redirect standard output, so you can redirect standard input. The redirect input symbol (tmp bash: tmp: cannot overwrite existing file $ set +o noclobber $ echo "hi there" > tmp $ tcsh tcsh tcsh tmp: tcsh tcsh $ $ touch tmp $ set noclobber $ echo "hi there" > tmp File exists. $ unset noclobber $ echo "hi there" > tmp Redirecting output can destroy a file II caution Depending on which shell you are using and how the environment is set up, a command such as the following may yield undesired results: $ cat orange pear > orange cat: orange: input file is output file Although cat displays an error message, the shell destroys the contents of the existing orange file. The new orange file will have the same contents as pear because the first action the shell takes when it sees the redirection symbol (>) is to remove the contents of the original orange file. If you want to catenate two files into one, use cat to put the two files into a temporary file and then use mv to rename the temporary file: $ cat orange pear > temp $ mv temp orange What happens in the next example can be even worse. The user giving the command wants to search through files a, b, and c for the word apple and redirect the output from grep (page 52) to the file a.output. Unfortunately the user enters the filename as a output, omitting the period and inserting a SPACE in its place: $ grep apple a b c > a output grep: output: No such file or directory The shell obediently removes the contents of a and then calls grep. The error message may take a moment to appear, giving you a sense the command is running correctly. Even after you see the error message, it may take a while to realize you have destroyed the contents of a. 130 Chapter 5 The Shell You can override noclobber by putting a pipe symbol (tcsh uses an exclamation point) after the redirect symbol (>|). In the following example, the user creates a file by redirecting the output of date. Next the user sets the noclobber variable and redirects output to the same file again. The shell displays an error message. Then the user places a pipe symbol after the redirect symbol and the shell allows the user to overwrite the file. $ date > tmp2 $ set -o noclobber $ date > tmp2 bash: a: cannot overwrite existing file $ date >| tmp2 $ For more information on using noclobber under tcsh, refer to page 377. Appending Standard Output to a File The append output symbol (>>) causes the shell to add new information to the end of a file, leaving existing information intact. This symbol provides a convenient way of catenating two files into one. The following commands demonstrate the action of the append output symbol. The second command accomplishes the catenation described in the preceding caution box: $ cat orange this is orange $ cat pear >> orange $ cat orange this is orange this is pear The first command displays the contents of the orange file. The second command appends the contents of the pear file to the orange file. The final cat displays the result. Do not trust noclobber caution Appending output is simpler than the two-step procedure described in the preceding caution box but you must be careful to include both greater than signs. If you accidentally use only one and the noclobber feature is not set, the shell will overwrite the orange file. Even if you have the noclobber feature turned on, it is a good idea to keep backup copies of the files you are manipulating in case you make a mistake. Although it protects you from overwriting a file using redirection, noclobber does not stop you from overwriting a file using cp or mv. These utilities include the –i (interactive) option that helps protect you from this type of mistake by verifying your intentions when you try to overwrite a file. For more information see the tip "cp can destroy a file" on page 50. The next example shows how to create a file that contains the date and time (the output from date), followed by a list of who is logged in (the output from who). The first line in Figure 5-11 redirects the output from date to the file named whoson. Then cat displays the file. Next the example appends the output from who to the whoson file. Finally cat displays the file containing the output of both utilities. Standard Input and Standard Output 131 $ date > whoson $ cat whoson Fri Mar 27 14:31:18 PST 2009 $ who >> whoson $ cat whoson Fri Mar 27 14:31:18 PST 2009 sam console Mar 27 05:00(:0) max pts/4 Mar 27 12:23(:0.0) max pts/5 Mar 27 12:33(:0.0) zach pts/7 Mar 26 08:45 (bravo.example.com) Figure 5-11 Redirecting and appending output /dev/null: Making Data Disappear The /dev/null device is a data sink, commonly referred to as a bit bucket. You can redirect output that you do not want to keep or see to /dev/null and the output will disappear without a trace: $ echo "hi there" > /dev/null $ When you read from /dev/null, you get a null string. Give the following cat command to truncate a file named messages to zero length while preserving the ownership and permissions of the file: $ ls -l messages -rw-r--r-1 max pubs 25315 Oct 24 10:55 messages $ cat /dev/null > messages $ ls -l messages -rw-r--r-1 max pubs 0 Oct 24 11:02 messages Pipes The shell uses a pipe to connect standard output of one command to standard input of another command. A pipe (sometimes referred to as a pipeline) has the same effect as redirecting standard output of one command to a file and then using that file as standard input to another command. A pipe does away with separate commands and the intermediate file. The symbol for a pipe is a vertical bar (|). The syntax of a command line using a pipe is command_a [arguments] | command_b [arguments] The preceding command line uses a pipe on a single command line to generate the same result as the following three command lines: command_a [arguments] > temp command_b [arguments] < temp rm temp In the preceding sequence of commands, the first line redirects standard output from command_a to an intermediate file named temp. The second line redirects 132 Chapter 5 The Shell $ ls > temp $ lpr temp $ rm temp or $ ls | lpr Figure 5-12 A pipe standard input for command_b to come from temp. The final line deletes temp. The command using a pipe is not only easier to type, but is more efficient because it does not create a temporary file. tr You can use a pipe with any of the Linux utilities that accept input either from a file specified on the command line or from standard input. You can also use pipes with utilities that accept input only from standard input. For example, the tr (translate; page 864) utility takes its input from standard input only. In its simplest usage tr has the following format: tr string1 string2 The tr utility accepts input from standard input and looks for characters that match one of the characters in string1. Upon finding a match, it translates the matched character in string1 to the corresponding character in string2. (The first character in string1 translates into the first character in string2, and so forth.) The tr utility sends its output to standard output. In both of the following examples, tr displays the contents of the abstract file with the letters a, b, and c translated into A, B, and C, respectively: $ cat abstract | tr abc ABC $ tr abc ABC < abstract The tr utility does not change the contents of the original file; it cannot change the original file because it does not "know" the source of its input. lpr The lpr (line printer; page 742) utility accepts input from either a file or standard input. When you type the name of a file following lpr on the command line, it places that file in the print queue. When you do not specify a filename on the command line, lpr takes input from standard input. This feature enables you to use a pipe to redirect input to lpr. The first set of commands in Figure 5-12 shows how you can use ls and lpr with an intermediate file (temp) to send a list of the files in the working directory to the printer. If the temp file exists, the first command overwrites its contents. The second set of commands uses a pipe to send the same list (with the exception of temp) to the printer. The commands in Figure 5-13 redirect the output from the who utility to temp and then display this file in sorted order. The sort utility (page 54) takes its input from the file specified on the command line or, when a file is not specified, from standard input; it sends its output to standard output. The sort command line in Figure 5-13 Standard Input and Standard Output 133 $ who > temp $ sort < temp max pts/4 max pts/5 sam console zach pts/7 $ rm temp Figure 5-13 Mar Mar Mar Mar 24 24 24 23 12:23 12:33 05:00 08:45 Using a temporary file to store intermediate results takes its input from standard input, which is redirected (temp $ lpr temp $ rm temp 3. What is a PID number? Why are these numbers useful when you run processes in the background? Which utility displays the PID numbers of the commands you are running? 4. Assume that the following files are in the working directory: $ ls intro notesa notesb ref1 ref2 ref3 section1 section2 section3 section4a section4b sentrev Give commands for each of the following, using wildcards to express filenames with as few characters as possible. a. List all files that begin with section. b. List the section1, section2, and section3 files only. c. List the intro file only. d. List the section1, section3, ref1, and ref3 files. 144 Chapter 5 The Shell 5. Refer to Part V or the man pages to determine which command will a. Output the number of lines in the standard input that contain the word a or A. b. Output only the names of the files in the working directory that contain the pattern $(. c. List the files in the working directory in reverse alphabetical order. d. Send a list of files in the working directory to the printer, sorted by size. 6. Give a command to a. Redirect standard output from a sort command to a file named phone_list. Assume the input file is named numbers. b. Translate all occurrences of the characters [and { to the character (, and all occurrences of the characters] and } to the character) in the file permdemos.c. (Hint: Refer to tr on page 864.) c. Create a file named book that contains the contents of two other files: part1 and part2. 7. The lpr and sort utilities accept input either from a file named on the command line or from standard input. a. Name two other utilities that function in a similar manner. b. Name a utility that accepts its input only from standard input. 8. Give an example of a command that uses grep a. With both input and output redirected. b. With only input redirected. c. With only output redirected. d. Within a pipe. In which of the preceding cases is grep used as a filter? 9. Explain the following error message. Which filenames would a subsequent ls display? $ ls abc abd abe abf abg abh $ rm abc ab* rm: cannot remove 'abc': No such file or directory Advanced Exercises 10. When you use the redirect output symbol (>) with a command, the shell creates the output file immediately, before the command is executed. Demonstrate that this is true. Advanced Exercises 11. In experimenting with shell variables, Max accidentally deletes his PATH variable. He decides he does not need the PATH variable. Discuss some of the problems he may soon encounter and explain the reasons for these problems. How could he easily return PATH to its original value? 12. Assume your permissions allow you to write to a file but not to delete it. a. Give a command to empty the file without invoking an editor. b. Explain how you might have permission to modify a file that you cannot delete. 13. If you accidentally create a filename that contains a nonprinting character, such as a CONTROL character, how can you remove the file? 14. Why does the noclobber variable not protect you from overwriting an existing file with cp or mv? 15. Why do command names and filenames usually not have embedded SPACEs? How would you create a filename containing a SPACE? How would you remove it? (This is a thought exercise, not recommended practice. If you want to experiment, create and work in a directory that contains only your experimental file.) 16. Create a file named answer and give the following command: $ > answers.0102 < answer cat Explain what the command does and why. What is a more conventional way of expressing this command? 145 This page intentionally left blank I PART II The Editors CHAPTER 6 The vim Editor 149 CHAPTER 7 The emacs Editor 205 147 This page intentionally left blank 6 The vim Editor In This Chapter This chapter begins with a history and description of vi, the original, powerful, sometimes cryptic, interactive, visually oriented text editor. The chapter continues with a tutorial that explains how to use vim (vi improved—a vi clone supplied with or available for most Linux distributions) to create and edit a file. Much of the tutorial and the balance of the chapter apply to vi and other vi clones. Following the tutorial, the chapter delves into the details of many vim commands and explains how to use parameters to customize vim to meet your needs. It concludes with a quick reference/summary of vim commands. 6Chapter6 149 150 Chapter 6 The vim Editor History Before vi was developed, the standard UNIX system editor was ed (available on most Linux systems), a line-oriented editor that made it difficult to see the context of your editing. Next came ex,1 a superset of ed. The most notable advantage that ex has over ed is a display-editing facility that allows you to work with a full screen of text instead of just a line. While using ex, you can bring up the display-editing facility by giving a vi (Visual mode) command. People used this display-editing facility so extensively that the developers of ex made it possible to start the editor with the display-editing facility already running, rather than having to start ex and then give a vi command. Appropriately they named the program vi. You can call the Visual mode from ex, and you can go back to ex while you are using vi. Start by running ex; give a vi command to switch to Visual mode, and give a Q command while in Visual mode to use ex. The quit command exits from ex. vi clones Linux offers a number of versions, or clones, of vi. The most popular of these clones are elvis (elvis.the-little-red-haired-girl.org), nvi (an implementation of the original vi editor, www.bostic.com/vi), vile (invisible-island.net/vile/vile.html), and vim (www.vim.org). Each clone offers additional features beyond those provided by the original vi. The examples in this book are based on vim. Several Linux distributions support multiple versions of vim. For example, Red Hat provides /bin/vi, a minimal build of vim that is compact and faster to load but offers fewer features, and /usr/bin/vim, a full-featured version of vim. If you use one of the clones other than vim, or vi itself, you may notice slight differences from the examples presented in this chapter. The vim editor is compatible with almost all vi commands and runs on many platforms, including Windows, Macintosh, OS/2, UNIX, and Linux. Refer to the vim home page (www.vim.org) for more information and a very useful Tips section. What vim is not The vim editor is not a text formatting program. It does not justify margins or provide the output formatting features of a sophisticated word processing system such as OpenOffice.org Writer. Rather, vim is a sophisticated text editor meant to be used to write code (C, HTML, Java, and so on), short notes, and input to a text formatting system, such as groff or troff. You can use fmt (page 697) to minimally format a text file you create with vim. Reading this chapter Because vim is so large and powerful, this chapter describes only some of its features. Nonetheless, if vim is completely new to you, you may find even this limited set of commands overwhelming. The vim editor provides a variety of ways to accomplish most editing tasks. A useful strategy for learning vim is to begin by learning a subset of commands to accomplish basic editing tasks. Then, as you become more comfortable with the editor, you can learn other commands that enable you to edit a file more quickly and efficiently. The following tutorial section 1. The ex program is usually a link to vi, which is a version of vim on some systems. Tutorial: Using vim to Create and Edit a File 151 introduces a basic, useful set of vim commands and features that will enable you to create and edit a file. Tutorial: Using vim to Create and Edit a File This section explains how to start vim, enter text, move the cursor, correct text, save the file to the disk, and exit from vim. The tutorial discusses three of the modes of operation of vim and explains how to switch from one mode to another. vimtutor and vim help files are not installed by default tip To run vimtutor and to get help as described on page 155, you must install the vim-runtime or vim-enhanced package; see Appendix C. vimtutor In addition to working with this tutorial, you may want to try vim's instructional program, named vimtutor. Give its name as a command to run it. Specifying a terminal Because vim takes advantage of features that are specific to various kinds of terminals, you must tell it which type of terminal or terminal emulator you are using. On many systems, and usually when you work on a terminal emulator, your terminal type is set automatically. If you need to specify your terminal type explicitly, refer to "Specifying a Terminal" on page 906. Starting vim Start vim with the following command to create and edit a file named practice: $ vim practice When you press RETURN, vim replaces the command line with a screen that looks similar to the one shown in Figure 6-1. Figure 6-1 Starting vim 152 Chapter 6 The vim Editor The tildes (~) at the left of the screen indicate that the file is empty. They disappear as you add lines of text to the file. If your screen looks like a distorted version of the one shown in Figure 6-1, your terminal type is probably not set correctly; see page 906. If you start vim with a terminal type that is not in the terminfo database, vim displays an error message and the terminal type defaults to ansi, which works on many terminals. In the following example, the user mistyped vt100 and set the terminal type to vg100: E558: Terminal entry not found in terminfo 'vg100' not known. Available builtin terminals are: builtin_ansi builtin_xterm builtin_iris-ansi builtin_dumb defaulting to 'ansi' The vi command may run vim tip On some Linux systems the command vi runs vim in vi-compatible mode (page 158). Emergency exit To reset the terminal type, press ESCAPE and then give the following command to exit from vim and display the shell prompt: :q! When you enter the colon (:), vim moves the cursor to the bottom line of the screen. The characters q! tell vim to quit without saving your work. (You will not ordinarily exit from vim this way because you typically want to save your work.) You must Figure 6-2 Starting vim without a filename Tutorial: Using vim to Create and Edit a File 153 press RETURN after you give this command. Once the shell displays a prompt, refer to "Specifying a Terminal" on page 906, and then start vim again. If you call vim without specifying a filename on the command line, vim assumes that you are a novice and tells you how to get started (Figure 6-2). The practice file is new so it does not contain any text. Thus vim displays a message similar to the one shown in Figure 6-1 on the status (bottom) line of the terminal to indicate that you are creating a new file. When you edit an existing file, vim displays the first few lines of the file and gives status information about the file on the status line. Command and Input Modes Two of vim's modes of operation are Command mode (also called Normal mode) and Input mode (Figure 6-3). While vim is in Command mode, you can give vim commands. For example, you can delete text or exit from vim. You can also command vim to enter Input mode. In Input mode, vim accepts anything you enter as text and displays it on the screen. Press ESCAPE to return vim to Command mode. By default the vim editor informs you about which mode it is in: It displays INSERT at the lower-left corner of the screen while it is in Insert mode. See the tip on page 160 if vim does not display INSERT when it is in Insert mode. The following command causes vim to display line numbers next to the text you are editing: :set number RETURN Colon (:) Last Line mode Command mode RETURN Insert, Append, Open, Replace, Change ESCAPE Input mode Figure 6-3 Modes in vim 154 Chapter 6 The vim Editor Figure 6-4 Last Line mode Entering text with vim The colon (:) in the preceding command puts vim into another mode, Last Line mode. While in this mode, vim keeps the cursor on the bottom line of the screen. When you press RETURN to finish entering the command, vim restores the cursor to its place in the text. Give the command :set nonumber RETURN to turn off line numbers. Entering Text i/a (Input mode) When you start vim, you must put it in Input mode before you can enter text. To put vim in Input mode, press the i (insert before cursor) key or the a (append after cursor) key. vim is case sensitive tip When you give vim a command, remember that the editor is case sensitive. In other words, vim interprets the same letter as two different commands, depending on whether you enter an uppercase or lowercase character. Beware of the CAPS LOCK (SHIFTLOCK) key. If you activate this key to enter uppercase text while you are in Input mode and then exit to Command mode, vim interprets your commands as uppercase letters. It can be confusing when this happens because vim does not appear to be executing the commands you are entering. If you are not sure whether vim is in Input mode, press the ESCAPE key; vim returns to Command mode if it is in Input mode or beeps, flashes, or does nothing if it is already in Command mode. You can put vim back in Input mode by pressing the i or a key again. While vim is in Input mode, you can enter text by typing on the keyboard. If the text does not appear on the screen as you type, vim is not in Input mode. Tutorial: Using vim to Create and Edit a File 155 Figure 6-5 The main vim Help screen To continue with this tutorial, enter the sample paragraph shown in Figure 6-4, pressing the RETURN key at the end of each line. If you do not press RETURN before the cursor reaches the right side of the screen or window, vim wraps the text so that it appears to start a new line. Physical lines will not correspond to programmatic (logical) lines in this situation, so editing will be more difficult. While you are using vim, you can always correct any typing mistakes you make. If you notice a mistake on the line you are entering, you can correct it before you continue (page 156); you can correct other mistakes later. When you finish entering the paragraph, press ESCAPE to return vim to Command mode. Getting Help To use vim's help system, you must install the vim-runtime package; see Appendix C. To get help while you are using vim, give the command :help [feature] followed by RETURN (you must be in Command mode when you give this command). The colon moves the cursor to the last line of the screen. If you type :help, vim displays an introduction to vim Help (Figure 6-5). Each area of the screen that displays a file, such as the two areas shown in Figure 6-5, is a vim "window." The dark band at the bottom of each vim window names the file that is displayed above it. In Figure 6-5, the help.txt file occupies most of the screen (the upper vim window). The file that is being edited (practice) occupies a few lines in the lower portion of the screen (the lower vim window). Read through the introduction to Help by scrolling the text as you read. Press j or the DOWN ARROW key to move the cursor down one line at a time; press CONTROL-D or CONTROL-U to scroll the cursor down or up half a window at a time. Give the command :q to close the Help window. 156 Chapter 6 The vim Editor Figure 6-6 Help with insert commands You can display information about the insert commands by giving the command :help insert while vim is in Command mode (Figure 6-6). Correcting Text as You Insert It The keys that back up and correct a shell command line serve the same functions when vim is in Input mode. These keys include the erase, line kill, and word kill keys (usually CONTROL-H, CONTROL-U, and CONTROL-W, respectively). Although vim may not remove deleted text from the screen as you back up over it using one of these keys, the editor does remove it when you type over the text or press RETURN. Moving the Cursor To delete, insert, and correct text, you need to move the cursor on the screen. While vim is in Command mode, you can use the RETURN key, the SPACE bar, and the ARROW keys to move the cursor. If you prefer to keep your hand closer to the center of the keyboard, if your terminal does not have ARROW keys, or if the emulator you are using does not support them, you can use the h, j, k, and l (lowercase "l") keys to move the cursor left, down, up, and right, respectively. Deleting Text x (Delete character) dw (Delete word) dd (Delete line) You can delete a single character by moving the cursor until it is over the character you want to delete and then giving the command x. You can delete a word by positioning the cursor on the first letter of the word and then giving the command dw (Delete word). You can delete a line of text by moving the cursor until it is anywhere on the line and then giving the command dd. Tutorial: Using vim to Create and Edit a File 157 Undoing Mistakes u (Undo) If you delete a character, line, or word by mistake or give any command you want to reverse, give the command u (Undo) immediately after the command you want to undo. The vim editor will restore the text to the way it was before you gave the last command. If you give the u command again, vim will undo the command you gave before the one it just undid. You can use this technique to back up over many of your actions. With the compatible parameter (page 158) set, however, vim can undo only the most recent change. :redo (Redo) If you undo a command you did not mean to undo, give a Redo command: CONTROL-R or :redo (followed by a RETURN). The vim editor will redo the undone command. As with the Undo command, you can give the Redo command many times in a row. With the compatible parameter (page 158) set, however, vim can redo only the most recent undo. Entering Additional Text i (Insert) a (Append) When you want to insert new text within existing text, move the cursor so it is on the character that follows the new text you plan to enter. Then give the i (Insert) command to put vim in Input mode, enter the new text, and press ESCAPE to return vim to Command mode. Alternatively, you can position the cursor on the character that precedes the new text and use the a (Append) command. o/O (Open) To enter one or more lines, position the cursor on the line above where you want the new text to go. Give the command o (Open). The vim editor opens a blank line below the line the cursor was on, puts the cursor on the new, empty line, and enters Input mode. Enter the new text, ending each line with a RETURN. When you are finished entering text, press ESCAPE to return vim to Command mode. The O command works in the same way as the o command, except it opens a blank line above the current line. Correcting Text To correct text, use dd, dw, or x to remove the incorrect text. Then use i, a, o, or O to insert the correct text. For example, to change the word pressing to hitting in Figure 6-4 on page 154, you might use the ARROW keys to move the cursor until it is on top of the p in pressing. Then give the command dw to delete the word pressing. Put vim in Input mode by giving an i command, enter the word hitting followed by a SPACE, and press ESCAPE. The word is changed and vim is in Command mode, waiting for another command. A shorthand for the two commands dw followed by the i command is cw (Change word). The cw command puts vim into Input mode. Page breaks for the printer tip CONTROL-L tells the printer to skip to the top of the next page. You can enter this character anywhere in a document by pressing CONTROL-L while you are in Input mode. If ^L does not appear, press CONTROL-V before CONTROL-L. 158 Chapter 6 The vim Editor Ending the Editing Session While you are editing, vim keeps the edited text in an area named the Work buffer. When you finish editing, you must write the contents of the Work buffer to a disk file so the edited text is saved and available when you next want it. Make sure vim is in Command mode, and use the ZZ command (you must use uppercase Zs) to write the newly entered text to the disk and end the editing session. After you give the ZZ command, vim returns control to the shell. You can exit with :q! if you do not want to save your work. Do not confuse ZZ with CONTROL-Z caution When you exit from vim with ZZ, make sure that you type ZZ and not CONTROL-Z (which is typically the suspend key). When you press CONTROL-Z, vim disappears from your screen, almost as though you had exited from it. In fact, vim will continue running in the background with your work unsaved. Refer to "Job Control" on page 285. If you try to start editing the same file with a new vim command, vim displays a message about a swap file (Figure 6-7, page 162). The compatible Parameter The compatible parameter makes vim more compatible with vi. By default this parameter is not set. To get started with vim you can ignore this parameter. Setting the compatible parameter changes many aspects of how vim works. For example, when the compatible parameter is set, the Undo command (page 157) can undo only the most recent change; in contrast, with the compatible parameter unset, you can call Undo repeatedly to undo many changes. This chapter notes when the compatible parameter affects a command. To obtain more details on the compatible parameter, give the command :help compatible RETURN. To display a complete list of vim's differences from the original vi, use :help vi-diff RETURN. See page 155 for a discussion of the help command. On the command line, use the –C option to set the compatible parameter and the –N option to unset it. Refer to "Setting Parameters from Within vim" on page 175 for information on changing the compatible parameter while you are running vim. Introduction to vim Features This section covers online help, modes of operation, the Work buffer, emergency procedures, and other vim features. To see which features are incorporated in a particular build, give a vim command followed by the ––version option. Online Help As covered briefly earlier, vim provides help while you are using it. Give the command :help feature to display information about feature. As you scroll through the various help texts, you will see words with a bar on either side, such as |tutor|. These Introduction to vim Features 159 words are active links: Move the cursor on top of an active link and press CONTROL-] to jump to the linked text. Use CONTROL-o (lowercase "o") to jump back to where you were in the help text. You can also use the active link words in place of feature. For example, you might see the reference |credits|; you could enter :help credits RETURN to read more about credits. Enter :q! to close a help window. Some common features that you may want to investigate by using the help system are insert, delete, and opening-window. Although opening-window is not intuitive, you will get to know the names of features as you spend more time with vim. You can also give the command :help doc-file-list to view a complete list of the help files. Although vim is a free program, the author requests that you donate the money you would have spent on similar software to help the children in Uganda (give the command :help iccf for more information). Terminology This chapter uses the following terms: Current character Current line Status line The character the cursor is on. The line the cursor is on. The last or bottom line of the screen. This line is reserved for Last Line mode and status information. Text you are editing does not appear on this line. Modes of Operation The vim editor is part of the ex editor, which has five modes of operation: • ex Command mode • ex Input mode • vim Command mode • vim Input mode • vim Last Line mode While in Command mode, vim accepts keystrokes as commands, responding to each command as you enter it. It does not display the characters you type in this mode. While in Input mode, vim accepts and displays keystrokes as text that it eventually puts into the file you are editing. All commands that start with a colon (:) put vim in Last Line mode. The colon moves the cursor to the status line of the screen, where you enter the rest of the command. In addition to the position of the cursor, there is another important difference between Last Line mode and Command mode. When you give a command in Command mode, you do not terminate the command with a RETURN. In contrast, you must terminate all Last Line mode commands with a RETURN. You do not normally use the ex modes. When this chapter refers to Input and Command modes, it means the vim modes, not the ex modes. 160 Chapter 6 The vim Editor When an editing session begins, vim is in Command mode. Several commands, including Insert and Append, put vim in Input mode. When you press the ESCAPE key, vim always reverts to Command mode. The Change and Replace commands combine the Command and Input modes. The Change command deletes the text you want to change and puts vim in Input mode so you can insert new text. The Replace command deletes the character(s) you overwrite and inserts the new one(s) you enter. Figure 6-3 on page 153 shows the modes and the methods for changing between them. Watch the mode and the CAPS LOCK key tip Almost anything you type in Command mode means something to vim. If you think vim is in Input mode when it is in Command mode, typing in text can produce confusing results. When you are learning to use vim, make sure the showmode parameter (page 188) is set (it is by default) to remind you which mode you are using. You may also find it useful to turn on the status line by giving a :set laststatus=2 command (page 187). Also keep your eye on the CAPS LOCK key. In Command mode, typing uppercase letters produces different results than typing lowercase ones. It can be disorienting to give commands and have vim give the "wrong" responses. The Display The vim editor uses the status line and several symbols to give information about what is happening during an editing session. Status Line The vim editor displays status information on the bottom line of the display area. This information includes error messages, information about the deletion or addition of blocks of text, and file status information. In addition, vim displays Last Line mode commands on the status line. Redrawing the Screen Sometimes the screen may become garbled or overwritten. When vim puts characters on the screen, it sometimes leaves @ on a line instead of deleting the line. When output from a program becomes intermixed with the display of the Work buffer, things can get even more confusing. The output does not become part of the Work buffer but affects only the display. If the screen gets overwritten, press ESCAPE to make sure vim is in Command mode, and press CONTROL-L to redraw (refresh) the screen. Tilde (~) Symbol If the end of the file is displayed on the screen, vim marks lines that would appear past the end of the file with a tilde (~) at the left of the screen. When you start editing a new file, the vim editor marks each line on the screen (except the first line) with this symbol. Correcting Text as You Insert It While vim is in Input mode, you can use the erase and line kill keys to back up over text so you can correct it. You can also use CONTROL-W to back up over words. Introduction to vim Features 161 Work Buffer The vim editor does all its work in the Work buffer. At the beginning of an editing session, vim reads the file you are editing from the disk into the Work buffer. During the editing session, it makes all changes to this copy of the file but does not change the file on the disk until you write the contents of the Work buffer back to the disk. Normally when you end an editing session, you tell vim to write the contents of the Work buffer, which makes the changes to the text final. When you edit a new file, vim creates the file when it writes the contents of the Work buffer to the disk, usually at the end of the editing session. Storing the text you are editing in the Work buffer has both advantages and disadvantages. If you accidentally end an editing session without writing out the contents of the Work buffer, your work is lost. However, if you unintentionally make some major changes (such as deleting the entire contents of the Work buffer), you can end the editing session without implementing the changes. To look at a file but not to change it while you are working with vim, you can use the view utility: $ view filename Calling the view utility is the same as calling the vim editor with the –R (readonly) option. Once you have invoked the editor in this way, you cannot write the contents of the Work buffer back to the file whose name appeared on the command line. You can always write the Work buffer out to a file with a different name. If you have installed mc (Midnight Commander), the view command calls mcview and not vim. Line Length and File Size The vim editor operates on files of any format, provided the length of a single line (that is, the characters between two NEWLINE characters) can fit into available memory. The total length of the file is limited only by available disk space and memory. Windows The vim editor allows you to open, close, and hide multiple windows, each of which allows you to edit a different file. Most of the window commands consist of CONTROL-W followed by another letter. For example, CONTROL-W s opens another window (splits the screen) that is editing the same file. CONTROL-W n opens a second window that is editing an empty file. CONTROL-W w moves the cursor between windows, and CONTROL-W q (or :q) quits (closes) a window. Give the command :help windows to display a complete list of windows commands. File Locks When you edit an existing file, vim displays the first few lines of the file, gives status information about the file on the status line, and locks the file. When you try to open a locked file with vim, you will see a message similar to the one shown in 162 Chapter 6 The vim Editor Figure 6-7 Attempting to open a locked file Figure 6-7. You will see this type of message in two scenarios: when you try to edit a file that someone is already editing (perhaps you are editing it in another window, in the background, or on another terminal) and when you try to edit a file that you were editing when vim or the system crashed. Although it is advisable to follow the instructions that vim displays, a second user can edit a file and write it out with a different filename. Refer to the next sections for more information. Abnormal Termination of an Editing Session You can end an editing session in one of two ways: When you exit from vim, you can save the changes you made during the editing session or you can abandon those changes. You can use the ZZ or :wq command from Command mode to save the changes and exit from vim (see "Ending the Editing Session" on page 158). To end an editing session without writing out the contents of the Work buffer, give the following command: :q! Use the :q! command cautiously. When you use this command to end an editing session, vim does not preserve the contents of the Work buffer, so you will lose any work you did since the last time you wrote the Work buffer to disk. The next time you edit or use the file, it will appear as it did the last time you wrote the Work buffer to disk. Sometimes you may find that you created or edited a file but vim will not let you exit. For example, if you forgot to specify a filename when you first called vim, you will get a message saying No file name when you give a ZZ command. If vim does not let you exit normally, you can use the Write command (:w) to name the file and write it to disk before you quit vim. Give the following command, substituting the name of the file for filename (remember to follow the command with a RETURN): Introduction to vim Features 163 :w filename After you give the Write command, you can use :q to quit using vim. You do not need to include the exclamation point (as in q!); it is necessary only when you have made changes since the last time you wrote the Work buffer to disk. Refer to page 183 for more information about the Write command. When you cannot write to a file tip It may be necessary to write a file using :w filename if you do not have write permission for the file you are editing. If you give a ZZ command and see the message "filename" is read only, you do not have write permission for the file. Use the Write command with a temporary filename to write the file to disk under a different filename. If you do not have write permission for the working directory, however, vim may not be able to write the file to the disk. Give the command again, using an absolute pathname of a dummy (nonexistent) file in your home directory in place of the filename. (For example, Max might give the command :w /home/max/tempor :w ~/temp.) If vim reports File exists, you will need to use :w! filename to overwrite the existing file (make sure you want to overwrite the file). Refer to page 184. Recovering Text After a Crash The vim editor temporarily stores the file you are working on in a swap file. If the system crashes while you are editing a file with vim, you can often recover its text from the swap file. When you attempt to edit a file that has a swap file, you will see a message similar to the one shown in Figure 6-7 on page 162. If someone else is editing the file, quit or open the file as a readonly file. In the following example, Max uses the –r option to check whether the swap file exists for a file named memo, which he was editing when the system crashed: $ vim -r Swap files found: In current directory: 1. .party.swp owned by: max dated: Sat Jan 26 11:36:44 2008 file name: ~max/party modified: YES user name: max host name: coffee process ID: 18439 2. .memo.swp owned by: max dated: Mon Mar 23 17:14:05 2009 file name: ~max/memo modified: no user name: max host name: coffee process ID: 27733 (still running) In directory ~/tmp: -- none -In directory /var/tmp: -- none -In directory /tmp: -- none -- Backward Backward 164 Chapter 6 The vim Editor Forward Forward Figure 6-8 Forward and backward With the –r option, vim displays a list of swap files it has saved (some may be old). If your work was saved, give the same command followed by a SPACE and the name of the file. You will then be editing a recent copy of your Work buffer. Give the command :w filename immediately to save the salvaged copy of the Work buffer to disk under a name different from the original file; then check the recovered file to make sure it is OK. Following is Max's exchange with vim as he recovers memo. Subsequently he deletes the swap file: $ vim -r memo Using swap file ".memo.swp" Original file "~/memo" Recovery completed. You should check if everything is OK. (You might want to write out this file under another name and run diff with the original file to check for changes) Delete the .swp file afterwards. Hit ENTER or type command to continue :w memo2 :q $ rm .memo.swp You must recover files on the system you were using tip The recovery feature of vim is specific to the system you were using when the crash occurred. If you are running on a cluster, you must log in on the system you were using before the crash to use the –r option successfully. Command Mode: Moving the Cursor While vim is in Command mode, you can position the cursor over any character on the screen. You can also display a different portion of the Work buffer on the screen. By manipulating the screen and cursor position, you can place the cursor on any character in the Work buffer. Command Mode: Moving the Cursor h 165 l SPACE Figure 6-9 Moving the cursor by characters You can move the cursor forward or backward through the text. As illustrated in Figure 6-8, forward means toward the right and bottom of the screen and the end of the file. Backward means toward the left and top of the screen and the beginning of the file. When you use a command that moves the cursor forward past the end (right) of a line, the cursor generally moves to the beginning (left) of the next line. When you move it backward past the beginning of a line, the cursor generally moves to the end of the previous line. Long lines Sometimes a line in the Work buffer may be too long to appear as a single line on the screen. In such a case vim wraps the current line onto the next line (unless you set the nowrap option [page 187]). You can move the cursor through the text by any Unit of Measure (that is, character, word, line, sentence, paragraph, or screen). If you precede a cursor-movement command with a number, called a Repeat Factor, the cursor moves that number of units through the text. Refer to pages 193 through page 196 for precise definitions of these terms. Moving the Cursor by Characters l/h The SPACE bar moves the cursor forward, one character at a time, toward the right side of the screen. The l (lowercase "l") key and the RIGHT ARROW key (Figure 6-9) do the same thing. For example, the command 7 SPACE or 7l moves the cursor seven characters to the right. These keys cannot move the cursor past the end of the current line to the beginning of the next line. The h and LEFT ARROW keys are similar to the l and RIGHT ARROW keys but work in the opposite direction. Moving the Cursor to a Specific Character f/F You can move the cursor to the next occurrence of a specified character on the current line by using the Find command. For example, the following command moves the cursor from its current position to the next occurrence of the character a, if one appears on the same line: fa You can also find the previous occurrence by using a capital F. The following command moves the cursor to the position of the closest previous a in the current line: Fa A semicolon (;) repeats the last Find command. 166 Chapter 6 The vim Editor B b w W belief,.really...It Moving the cursor by words Figure 6-10 Moving the Cursor by Words w/W The w (word) key moves the cursor forward to the first letter of the next word (Figure 6-10). Groups of punctuation count as words. This command goes to the next line if the next word is located there. The command 15w moves the cursor to the first character of the fifteenth subsequent word. The W key is similar to the w key but moves the cursor by blank-delimited words, including punctuation, as it skips forward. (Refer to "Blank-Delimited Word" on page 194.) b/B e/E The b (back) key moves the cursor backward to the first letter of the previous word. The B key moves the cursor backward by blank-delimited words. Similarly the e key moves the cursor to the end of the next word; E moves it to the end of the next blank-delimited word. Moving the Cursor by Lines j/k The RETURN key moves the cursor to the beginning of the next line; the j and DOWN ARROW keys move the cursor down one line to the character just below the current character (Figure 6-11). If no character appears immediately below the current character, the cursor moves to the end of the next line. The cursor will not move past the last line of text in the work buffer. The k and UP ARROW keys are similar to the j and DOWN ARROW keys but work in the opposite direction. The minus (–) key is similar to the RETURN key but works in the opposite direction. needed.as – k with.their RETURN j working..To Figure 6-11 Moving the cursor by lines Command Mode: Moving the Cursor 167 { H (M Cursor) L } Figure 6-12 Moving the cursor by sentences, paragraphs, H, M, and L Moving the Cursor by Sentences and Paragraphs)/(}/{ The) and } keys move the cursor forward to the beginning of the next sentence or the next paragraph, respectively (Figure 6-12). The (and { keys move the cursor backward to the beginning of the current sentence or paragraph, respectively. You can find more information on sentences and paragraphs starting on page 194. Moving the Cursor Within the Screen H/M/L The H (home) key positions the cursor at the left end of the top line of the screen, the M (middle) key moves the cursor to the middle line, and the L (lower) key moves it to the bottom line (Figure 6-12). Viewing Different Parts of the Work Buffer The screen displays a portion of the text that is in the Work buffer. You can display the text preceding or following the text on the screen by scrolling the display. You can also display a portion of the Work buffer based on a line number. CONTROL-D CONTROL-U Press CONTROL-D to scroll the screen down (forward) through the file so that vim displays half a screen of new text. Use CONTROL-U to scroll the screen up (backward) by the same amount. If you precede either of these commands with a number, vim scrolls that number of lines each time you press CONTROL-D or CONTROL-U for the rest of the session (unless you again change the number of lines to scroll). See page 188 for a discussion of the scroll parameter. CONTROL-F CONTROL-B The CONTROL-F (forward) and CONTROL-B (backward) keys display almost a whole screen of new text, leaving a couple of lines from the previous screen for continuity. On many keyboards you can use the PAGE DOWN and PAGE UP keys in place of CONTROL-F and CONTROL-B , respectively. 168 Chapter 6 The vim Editor I i a A This.is.a.line.of.text. Figure 6-13 Line numbers (G) The I, i, a, and A commands When you enter a line number followed by G (goto), vim positions the cursor on that line in the Work buffer. If you press G without a number, vim positions the cursor on the last line in the Work buffer. Line numbers are implicit; the file does not need to have actual line numbers for this command to work. Refer to "Line numbers" on page 187 if you want vim to display line numbers. Input Mode The Insert, Append, Open, Change, and Replace commands put vim in Input mode. While vim is in this mode, you can put new text into the Work buffer. To return vim to Command mode when you finish entering text, press the ESCAPE key. Refer to "Show mode" on page 188 if you want vim to remind you when it is in Input mode (it does by default). Inserting Text Insert (i/I) The i (Insert) command puts vim in Input mode and places the text you enter before the current character. The I command places text at the beginning of the current line (Figure 6-13). Although the i and I commands sometimes overwrite text on the screen, the characters in the Work buffer are not changed; only the display is affected. The overwritten text is redisplayed when you press ESCAPE and vim returns to Command mode. Use i or I to insert a few characters or words into existing text or to insert text in a new file. Appending Text Append (a/A) The a (Append) command is similar to the i command, except that it places the text you enter after the current character (Figure 6-13). The A command places the text after the last character on the current line. Opening a Line for Text Open (o/O) The o (Open) and O commands open a blank line within existing text, place the cursor at the beginning of the new (blank) line, and put vim in Input mode. The O command opens a line above the current line; the o command opens one below the current line. Use these commands when you are entering several new lines within existing text. Command Mode: Deleting and Changing Text 169 Replacing Text Replace (r/R) The r and R (Replace) commands cause the new text you enter to overwrite (replace) existing text. The single character you enter following an r command overwrites the current character. After you enter that character, vim returns to Command mode—you do not need to press the ESCAPE key. The R command causes all subsequent characters to overwrite existing text until you press ESCAPE to return vim to Command mode. Replacing TABs tip The Replace commands may appear to behave strangely when you replace TAB characters. TAB characters can appear as several SPACEs—until you try to replace them. A TAB is one character and is replaced by a single character. Refer to "Invisible characters" on page 187 for information on displaying TABs as visible characters. Quoting Special Characters in Input Mode CONTROL-V While you are in Input mode, you can use the Quote command, CONTROL-V, to enter any character into the text, including characters that normally have special meaning to vim. Among these characters are CONTROL-L (or CONTROL-R), which redraws the screen; CONTROL-W, which backs the cursor up a word to the left; CONTROL-M, which enters a NEWLINE; and ESCAPE , which ends Input mode. To insert one of these characters into the text, type CONTROL-V followed by the character. CONTROL-V quotes the single character that follows it. For example, to insert the sequence ESCAPE [2J into a file you are creating in vim, you would type the character sequence CONTROL-V ESCAPE [2J. This character sequence clears the screen of a DEC VT-100 and other similar terminals. Although you would not ordinarily want to type this sequence into a document, you might want to use it or another ESCAPE sequence in a shell script you are creating in vim. Refer to Chapter 10 for information about writing shell scripts. Command Mode: Deleting and Changing Text This section describes the commands to delete and replace, or change, text in the document you are editing. The Undo command is covered here because it allows you to restore deleted or changed text. Undoing Changes Undo (u/U) The u command (Undo) restores text that you just deleted or changed by mistake. A single Undo command restores only the most recently deleted text. If you delete a line and then change a word, the first Undo restores only the changed word; you have to give a second Undo command to restore the deleted line. With the compatible parameter (page 158) set, vim can undo only the most recent change. The U 170 Chapter 6 The vim Editor command restores the last line you changed to the way it was before you started changing it, even after several changes. Deleting Characters Delete character (x/X) The x command deletes the current character. You can precede the x command by a Repeat Factor (page 196) to delete several characters on the current line, starting with the current character. The X command deletes the character

to the left of the cursor. Deleting Text Delete (d/D) The d (Delete) command removes text from the Work buffer. The amount of text that d removes depends on the Repeat Factor and the Unit of Measure (page 193). After the text is deleted, vim is still in Command mode. Use dd to delete a single line tip The command d RETURN deletes two lines: the current line and the following one. Use dd to delete just the current line, or precede dd by a Repeat Factor (page 196) to delete several lines. You can delete from the current cursor position up to a specific character on the same line. To delete up to the next semicolon (;), give the command dt; (see page 173 for more information on the t command). To delete the remainder of the current line, use D or d$. Table 6-1 lists some Delete commands. Each command, except the last group that starts with dd, deletes from/to the current character. Exchange characters and lines tip If two characters are out of order, position the cursor on the first character and give the commands xp. If two lines are out of order, position the cursor on the first line and give the commands ddp. See page 181 for more information on the Put commands. Table 6-1 Delete command examples Command Result dl Deletes current character (same as the x command) d0 Deletes from beginning of line d^ Deletes from first character of line (not including leading SPACEs or TABs) dw Deletes to end of word d3w Deletes to end of third word db Deletes from beginning of word dW Deletes to end of blank-delimited word Command Mode: Deleting and Changing Text 171 Table 6-1 Delete command examples (continued) Command Result dB Deletes from beginning of blank-delimited word d7B Deletes from seventh previous beginning of blank-delimited word d) Deletes to end of sentence d4) Deletes to end of fourth sentence d(Deletes from beginning of sentence d} Deletes to end of paragraph d{ Deletes from beginning of paragraph d7{ Deletes from seventh paragraph preceding beginning of paragraph d/text Deletes up to next occurrence of word text dfc Deletes on current line up to and including next occurrence of character c dtc Deletes on current line up to next occurrence of c D Deletes to end of line d$ Deletes to end of line dd Deletes current line 5dd Deletes five lines starting with current line dL Deletes through last line on screen dH Deletes from first line on screen dG Deletes through end of Work buffer d1G Deletes from beginning of Work buffer Changing Text Change (c/C) The c (Change) command replaces existing text with new text. The new text does not have to occupy the same space as the existing text. You can change a word to several words, a line to several lines, or a paragraph to a single character. The C command replaces the text from the cursor position to the end of the line. The c command deletes the amount of text specified by the Repeat Factor and the Unit of Measure (page 193) and puts vim in Input mode. When you finish entering the new text and press ESCAPE, the old word, line, sentence, or paragraph is changed to the new one. Pressing ESCAPE without entering new text deletes the specified text (that is, it replaces the specified text with nothing). 172 Chapter 6 The vim Editor Table 6-2 lists some Change commands. Except for the last two, each command changes text from/to the current character. Table 6-2 Change command examples Command Result cl Changes current character cw Changes to end of word c3w Changes to end of third word cb Changes from beginning of word cW Changes to end of blank-delimited word cB Changes from beginning of blank-delimited word c7B Changes from beginning of seventh previous blank-delimited word c$ Changes to end of line c0 Changes from beginning of line c) Changes to end of sentence c4) Changes to end of fourth sentence c(Changes from beginning of sentence c} Changes to end of paragraph c{ Changes from beginning of paragraph c7{ Changes from beginning of seventh preceding paragraph ctc Changes on current line up to next occurrence of c C Changes to end of line cc Changes current line 5cc Changes five lines starting with current line dw works differently from cw tip The dw command deletes all characters through (including) the SPACE at the end of a word. The cw command changes only the characters in the word, leaving the trailing SPACE intact. Replacing Text Substitute (s/S) The s and S (Substitute) commands also replace existing text with new text (Table 6-3). The s command deletes the current character and puts vim into Input mode. It has the effect of replacing the current character with whatever you type until you press ESCAPE. The S command does the same thing as the cc command: It Searching and Substituting 173 changes the current line. The s command replaces characters only on the current line. If you specify a Repeat Factor before an s command and this action would replace more characters than are present on the current line, s changes characters only to the end of the line (same as C). Table 6-3 Substitute command examples Command Result s Substitutes one or more characters for current character S Substitutes one or more characters for current line 5s Substitutes one or more characters for five characters, starting with current character Changing Case The tilde (~) character changes the case of the current character from uppercase to lowercase, or vice versa. You can precede the tilde with a number to specify the number of characters you want the command to affect. For example, the command 5~ transposes the next five characters starting with the character under the cursor, but will not transpose characters past the end of the current line. Searching and Substituting Searching for and replacing a character, a string of text, or a string that is matched by a regular expression is a key feature of any editor. The vim editor provides simple commands for searching for a character on the current line. It also provides more complex commands for searching for—and optionally substituting for—single and multiple occurrences of strings or regular expressions anywhere in the Work buffer. Searching for a Character Find (f/F) You can search for and move the cursor to the next occurrence of a specified character on the current line using the f (Find) command. Refer to “Moving the Cursor to a Specific Character” on page 165. Find (t/T) The next two commands are used in the same manner as the Find commands. The t command places the cursor on the character before the next occurrence of the specified character. The T command places the cursor on the character after the previous occurrence of the specified character. A semicolon (;) repeats the last f, F, t, or T command. You can combine these search commands with other commands. For example, the command d2fq deletes the text from the current character to the second occurrence of the letter q on the current line. 174 Chapter 6 The vim Editor Searching for a String Search (//?) The vim editor can search backward or forward through the Work buffer to find a string of text or a string that matches a regular expression (Appendix A). To find the next occurrence of a string (forward), press the forward slash (/) key, enter the text you want to find (called the search string), and press RETURN. When you press the slash key, vim displays a slash on the status line. As you enter the string of text, it is also displayed on the status line. When you press RETURN, vim searches for the string. If this search is successful, vim positions the cursor on the first character of the string. If you use a question mark (?) in place of the forward slash, vim searches for the previous occurrence of the string. If you need to include a forward slash in a forward search or a question mark in a backward search, you must quote it by preceding it with a backslash (\). Two distinct ways of quoting characters tip You use CONTROL-V to quote special characters in text that you are entering into a file (page 169). This section discusses the use of a backslash (\) to quote special characters in a search string. The two techniques of quoting characters are not interchangeable. Next (n/N) The N and n keys repeat the last search but do not require you to reenter the search string. The n key repeats the original search exactly, and the N key repeats the search in the opposite direction of the original search. If you are searching forward and vim does not find the search string before it gets to the end of the Work buffer, the editor typically wraps around and continues the search at the beginning of the Work buffer. During a backward search, vim wraps around from the beginning of the Work buffer to the end. Also, vim normally performs case-sensitive searches. Refer to “Wrap scan” (page 189) and “Ignore case in searches” (page 187) for information about how to change these search parameters. Normal Versus Incremental Searches When vim performs a normal search (its default behavior), you enter a slash or question mark followed by the search string and press RETURN. The vim editor then moves the cursor to the next or previous occurrence of the string you are searching for. When vim performs an incremental search, you enter a slash or question mark. As you enter each character of the search string, vim moves the highlight to the next or previous occurrence of the string you have entered so far. When the highlight is on the string you are searching for, you must press RETURN to move the cursor to the highlighted string. If the string you enter does not match any text, vim does not highlight anything. The type of search that vim performs depends on the incsearch parameter (page 187). Give the command :set incsearch to turn on incremental searching; use noincsearch to turn it off. When you set the compatible parameter (page 158), vim turns off incremental searching. Searching and Substituting 175 Special Characters in Search Strings Because the search string is a regular expression, some characters take on a special meaning within the search string. The following paragraphs list some of these characters. See also “Extended Regular Expressions” on page 893. The first two items in the following list (^ and $) always have their special meanings within a search string unless you quote them by preceding them with a backslash (\). You can turn off the special meanings within a search string for the rest of the items in the list by setting the nomagic parameter. For more information refer to “Allow special characters in searches” on page 186. ^ Beginning-of-Line Indicator When the first character in a search string is a caret (also called a circumflex), it matches the beginning of a line. For example, the command /^the finds the next line that begins with the string the. $ End-of-Line Indicator A dollar sign matches the end of a line. For example, the command /!$ finds the next line that ends with an exclamation point and / $ matches the next line that ends with a SPACE. . Any-Character Indicator A period matches any character, anywhere in the search string. For example, the command /l..e finds line, followed, like, included, all memory, or any other word or character string that contains an l followed by any two characters and an e. To search for a period, use a backslash to quote the period (\.). \> End-of-Word Indicator This pair of characters matches the end of a word. For example, the command /s\> finds the next word that ends with an s. Whereas a backslash (\) is typically used to turn off the special meaning of a character, the character sequence \> has a special meaning, while > alone does not. \ required to complete the transaction.' Please enter the three values \ required to complete the transaction. | and & Separate Commands and Do Something Else The pipe symbol (|) and the background task symbol (&) are also command separators. They do not start execution of a command but do change some aspect of how the command functions. The pipe symbol alters the source of standard input or the destination of standard output. The background task symbol causes the shell to execute the task in the background and display a prompt immediately; you can continue working on other tasks. Each of the following command lines initiates a single job comprising three tasks: $ x | y | z $ ls -l | grep tmp | less In the first job, the shell redirects standard output of task x to standard input of task y and redirects y’s standard output to z’s standard input. Because it runs the entire job in the foreground, the shell does not display a prompt until task z runs to completion: Task z does not finish until task y finishes, and task y does not finish until task x finishes. In the second job, task x is an ls –l command, task y is grep tmp, and task z is the pager less. The shell displays a long (wide) listing of the files in the working directory that contain the string tmp, piped through less. The next command line executes tasks d and e in the background and task f in the foreground: $ d & e & f [1] 14271 [2] 14272 The shell displays the job number between brackets and the PID number for each process running in the background. It displays a prompt as soon as f finishes, which may be before d or e finishes. Before displaying a prompt for a new command, the shell checks whether any background jobs have completed. For each completed job, the shell displays its job number, the word Done, and the command line that invoked the job; the shell then displays a prompt. When the job numbers are listed, the number of the last job started is followed by a + character and the job number of the previous job is followed by a – character. 284 Chapter 8 The Bourne Again Shell Other jobs are followed by a SPACE character. After running the last command, the shell displays the following lines before issuing a prompt: [1][2]+ Done Done d e The next command line executes all three tasks as background jobs. The shell displays a prompt immediately: $ d [1] [2] [3] & e & f & 14290 14291 14292 You can use pipes to send the output from one task to the next task and an ampersand (&) to run the entire job as a background task. Again the shell displays the prompt immediately. The shell regards the commands joined by a pipe as a single job. That is, it treats all pipes as single jobs, no matter how many tasks are connected with the pipe (|) symbol or how complex they are. The Bourne Again Shell reports only one process in the background (although there are three): $ d | e | f & [1] 14295 The TC Shell shows three processes (all belonging to job 1) placed in the background: tcsh $ d | e | f & [1] 14302 14304 14306 optional ( ) Groups Commands You can use parentheses to group commands. The shell creates a copy of itself, called a subshell, for each group. It treats each group of commands as a job and creates a new process to execute each command (refer to “Process Structure” on page 306 for more information on creating subshells). Each subshell (job) has its own environment, meaning that it has its own set of variables whose values can differ from those found in other subshells. The following command line executes commands a and b sequentially in the background while executing c in the background. The shell displays a prompt immediately. $ (a ; b) & c & [1] 15520 [2] 15521 The preceding example differs from the earlier example d & e & f & in that tasks a and b are initiated sequentially, not concurrently. Similarly the following command line executes a and b sequentially in the background and, at the same time, executes c and d sequentially in the background. The subshell running a and b and the subshell running c and d run concurrently. The shell displays a prompt immediately. $ (a ; b) & (c ; d) & [1] 15528 [2] 15529 Shell Basics 285 The next script copies one directory to another. The second pair of parentheses creates a subshell to run the commands following the pipe. Because of these parentheses, the output of the first tar command is available for the second tar command despite the intervening cd command. Without the parentheses, the output of the first tar command would be sent to cd and lost because cd does not process input from standard input. The shell variables $1 and $2 represent the first and second command-line arguments (page 441), respectively. The first pair of parentheses, which creates a subshell to run the first two commands, allows users to call cpdir with relative pathnames. Without them, the first cd command would change the working directory of the script (and consequently the working directory of the second cd command). With them, only the working directory of the subshell is changed. $ cat cpdir (cd $1 ; tar -cf - .) | (cd $2 ; tar -xvf -) $ ./cpdir /home/max/sources /home/max/memo/biblio The cpdir command line copies the files and directories in the /home/max/sources directory to the directory named /home/max/memo/biblio. This shell script is almost the same as using cp with the –r option. Refer to Part V for more information on cp and tar. Job Control A job is a command pipeline. You run a simple job whenever you give the shell a command. For example, if you type date on the command line and press RETURN, you have run a job. You can also create several jobs with multiple commands on a single command line: $ find . -print | sort | lpr & grep -l max /tmp/* > maxfiles & [1] 18839 [2] 18876 The portion of the command line up to the first & is one job consisting of three processes connected by pipes: find (page 688), sort (pages 54 and 817), and lpr (pages 51 and 742). The second job is a single process running grep. The trailing & characters put each job in the background, so bash does not wait for them to complete before displaying a prompt. Using job control you can move commands from the foreground to the background (and vice versa), stop commands temporarily, and list all commands that are running in the background or stopped. jobs: Lists Jobs The jobs builtin lists all background jobs. Following, the sleep command runs in the background and creates a background job that jobs reports on: $ sleep 60 & [1] 7809 $ jobs [1] + Running sleep 60 & 286 Chapter 8 The Bourne Again Shell fg: Brings a Job to the Foreground The shell assigns a job number to each command you run in the background. For each job run in the background, the shell lists the job number and PID number immediately, just before it issues a prompt: $ xclock & [1] 1246 $ date & [2] 1247 $ Tue Dec 2 11:44:40 PST 2008 [2]+ Done date $ find /usr -name ace -print > findout & [2] 1269 $ jobs [1]- Running xclock & [2]+ Running find /usr -name ace -print > findout & Job numbers, which are discarded when a job is finished, can be reused. When you start or put a job in the background, the shell assigns a job number that is one more than the highest job number in use. In the preceding example, the jobs command lists the first job, xclock, as job 1. The date command does not appear in the jobs list because it finished before jobs was run. Because the date command was completed before find was run, the find command became job 2. To move a background job to the foreground, use the fg builtin followed by the job number. Alternatively, you can give a percent sign (%) followed by the job number as a command. Either of the following commands moves job 2 to the foreground. When you move a job to the foreground, the shell displays the command it is now executing in the foreground. $ fg 2 find /usr -name ace -print > findout or $ %2 find /usr -name ace -print > findout You can also refer to a job by following the percent sign with a string that uniquely identifies the beginning of the command line used to start the job. Instead of the preceding command, you could have used either fg %find or fg %f because both uniquely identify job 2. If you follow the percent sign with a question mark and a string, the string can match any part of the command line. In the preceding example, fg %?ace also brings job 2 to the foreground. Often the job you wish to bring to the foreground is the only job running in the background or is the job that jobs lists with a plus (+). In these cases fg without an argument brings the job to the foreground. Shell Basics 287 Suspending a Job Pressing the suspend key (usually CONTROL-Z) immediately suspends (temporarily stops) the job in the foreground and displays a message that includes the word Stopped. CONTROL-Z [2]+ Stopped find /usr -name ace -print > findout For more information refer to “Moving a Job from the Foreground to the Background” on page 135. bg: Sends a Job to the Background To move the foreground job to the background, you must first suspend the job (above). You can then use the bg builtin to resume execution of the job in the background. $ bg [2]+ find /usr -name ace -print > findout & If a background job attempts to read from the terminal, the shell stops the program and displays a message saying the job has been stopped. You must then move the job to the foreground so it can read from the terminal. $ (sleep 5; cat > mytext) & [1] 1343 $ date Tue Dec 2 11:58:20 PST 2008 [1]+ Stopped $ fg (sleep 5; cat >mytext) Remember to let the cat out! (sleep 5; cat >mytext) CONTROL-D $ In the preceding example, the shell displays the job number and PID number of the background job as soon as it starts, followed by a prompt. Demonstrating that you can give a command at this point, the user gives the command date and its output appears on the screen. The shell waits until just before it issues a prompt (after date has finished) to notify you that job 1 is stopped. When you give an fg command, the shell puts the job in the foreground and you can enter the data the command is waiting for. In this case the input needs to be terminated with CONTROL-D, which sends an EOF (end of file) signal to the shell. The shell then displays another prompt. The shell keeps you informed about changes in the status of a job, notifying you when a background job starts, completes, or stops, perhaps because it is waiting for input from the terminal. The shell also lets you know when a foreground job is suspended. Because notices about a job being run in the background can disrupt your work, the shell delays displaying these notices until just before it displays a prompt. You can set notify (page 333) to cause the shell to display these notices without delay. 288 Chapter 8 The Bourne Again Shell home sam demo names literature promo Figure 8-2 The directory structure in the examples If you try to exit from a shell while jobs are stopped, the shell issues a warning and does not allow you to exit. If you then use jobs to review the list of jobs or you immediately try to exit from the shell again, the shell allows you to exit. If huponexit (page 333) is not set (the default), stopped and background jobs keep running in the background. If it is set, the shell terminates the jobs. Manipulating the Directory Stack Both the Bourne Again and the TC Shells allow you to store a list of directories you are working with, enabling you to move easily among them. This list is referred to as a stack. It is analogous to a stack of dinner plates: You typically add plates to and remove plates from the top of the stack, so this type of stack is named a last in, first out (LIFO) stack. dirs: Displays the Stack The dirs builtin displays the contents of the directory stack. If you call dirs when the directory stack is empty, it displays the name of the working directory: $ dirs ~/literature names 21 pushd demo 12 pushd literature Figure 8-3 Creating a directory stack Shell Basics 289 pushd pushdpushd names demo names demo names demo literature literature literature Figure 8-4 pushd Using pushd to change working directories The dirs builtin uses a tilde (~) to represent the name of a user’s home directory. The examples in the next several sections assume that you are referring to the directory structure shown in Figure 8-2. pushd: Pushes a Directory on the Stack When you supply the pushd (push directory) builtin with one argument, it pushes the directory specified by the argument on the stack, changes directories to the specified directory, and displays the stack. The following example is illustrated in Figure 8-3: $ pushd ../demo ~/demo ~/literature $ pwd /home/sam/demo $ pushd ../names ~/names ~/demo ~/literature $ pwd /home/sam/names When you use pushd without an argument, it swaps the top two directories on the stack, makes the new top directory (which was the second directory) the new working directory, and displays the stack (Figure 8-4): $ pushd ~/demo ~/names ~/literature $ pwd /home/sam/demo Using pushd in this way, you can easily move back and forth between two directories. You can also use cd – to change to the previous directory, whether or not you have explicitly created a directory stack. To access another directory in the stack, call pushd with a numeric argument preceded by a plus sign. The directories in the stack are numbered starting with the top directory, which is number 0. The following pushd command continues with the previous example, changing the working directory to literature and moving literature to the top of the stack: $ pushd +2 ~/literature ~/demo ~/names $ pwd /home/sam/literature 290 Chapter 8 The Bourne Again Shell literature popd demo names Figure 8-5 Using popd to remove a directory from the stack popd: Pops a Directory Off the Stack To remove a directory from the stack, use the popd (pop directory) builtin. As the following example and Figure 8-5 show, without an argument, popd removes the top directory from the stack and changes the working directory to the new top directory: $ dirs ~/literature ~/demo ~/names $ popd ~/demo ~/names $ pwd /home/sam/demo To remove a directory other than the top one from the stack, use popd with a numeric argument preceded by a plus sign. The following example removes directory number 1, demo. Removing a directory other than directory number 0 does not change the working directory. $ dirs ~/literature ~/demo ~/names $ popd +1 ~/literature ~/names Parameters and Variables Variables Within a shell, a shell parameter is associated with a value that is accessible to the user. There are several kinds of shell parameters. Parameters whose names consist of letters, digits, and underscores are often referred to as shell variables, or simply variables. A variable name must start with a letter or underscore, not with a number. Thus A76, MY_CAT, and ___X___ are valid variable names, whereas 69TH_STREET (starts with a digit) and MY-NAME (contains a hyphen) are not. User-created variables Shell variables that you name and assign values to are user-created variables. You can change the values of user-created variables at any time, or you can make them readonly so that their values cannot be changed. You can also make user-created variables global. A global variable (also called an environment variable) is available to all shells and other programs you fork from the original shell. One naming convention is to use only uppercase letters for global variables and to use mixed-case or Parameters and Variables 291 lowercase letters for other variables. Refer to “Locality of Variables” on page 436 for more information on global variables. To assign a value to a variable in the Bourne Again Shell, use the following syntax: VARIABLE=value There can be no whitespace on either side of the equal sign (=). An example assignment follows: $ myvar=abc Under the TC Shell the assignment must be preceded by the word set and the on either side of the equal sign are optional: SPACEs $ set myvar = abc The Bourne Again Shell permits you to put variable assignments on a command line. This type of assignment creates a variable that is local to the command shell—that is, the variable is accessible only from the program the command runs. The my_script shell script displays the value of TEMPDIR. The following command runs my_script with TEMPDIR set to /home/sam/temp. The echo builtin shows that the interactive shell has no value for TEMPDIR after running my_script. If TEMPDIR had been set in the interactive shell, running my_script in this manner would have had no effect on its value. $ cat my_script echo $TEMPDIR $ TEMPDIR=/home/sam/temp ./my_script /home/sam/temp $ echo $TEMPDIR $ Keyword variables Keyword shell variables (or simply keyword variables) have special meaning to the shell and usually have short, mnemonic names. When you start a shell (by logging in, for example), the shell inherits several keyword variables from the environment. Among these variables are HOME, which identifies your home directory, and PATH, which determines which directories the shell searches and in what order to locate commands that you give the shell. The shell creates and initializes (with default values) other keyword variables when you start it. Still other variables do not exist until you set them. You can change the values of most keyword shell variables. It is usually not necessary to change the values of keyword variables initialized in the /etc/profile or /etc/csh.cshrc systemwide startup files. If you need to change the value of a bash keyword variable, do so in one of your startup files (for bash see page 271; for tcsh see page 352). Just as you can make user-created variables global, so you can make keyword variables global—a task usually done automatically in startup files. You can also make a keyword variable readonly. Positional and special parameters The names of positional and special parameters do not resemble variable names. Most of these parameters have one-character names (for example, 1, ?, and #) and 292 Chapter 8 The Bourne Again Shell are referenced (as are all variables) by preceding the name with a dollar sign ($1, $?, and $#). The values of these parameters reflect different aspects of your ongoing interaction with the shell. Whenever you give a command, each argument on the command line becomes the value of a positional parameter (page 440). Positional parameters enable you to access command-line arguments, a capability that you will often require when you write shell scripts. The set builtin (page 442) enables you to assign values to positional parameters. Other frequently needed shell script values, such as the name of the last command executed, the number of command-line arguments, and the status of the most recently executed command, are available as special parameters (page 438). You cannot assign values to special parameters. User-Created Variables The first line in the following example declares the variable named person and initializes it with the value max (use set person = max in tcsh): $ person=max $ echo person person $ echo $person max Parameter substitution Because the echo builtin copies its arguments to standard output, you can use it to display the values of variables. The second line of the preceding example shows that person does not represent max. Instead, the string person is echoed as person. The shell substitutes the value of a variable only when you precede the name of the variable with a dollar sign ($). Thus the command echo $person displays the value of the variable person; it does not display $person because the shell does not pass $person to echo as an argument. Because of the leading $, the shell recognizes that $person is the name of a variable, substitutes the value of the variable, and passes that value to echo. The echo builtin displays the value of the variable —not its name—never “knowing” that you called it with a variable. Quoting the $ You can prevent the shell from substituting the value of a variable by quoting the leading $. Double quotation marks do not prevent the substitution; single quotation marks or a backslash (\) do. $ echo $person max $ echo "$person" max $ echo '$person' $person $ echo \$person $person Parameters and Variables 293 SPACEs Because they do not prevent variable substitution but do turn off the special meanings of most other characters, double quotation marks are useful when you assign values to variables and when you use those values. To assign a value that contains SPACEs or TABs to a variable, use double quotation marks around the value. Although double quotation marks are not required in all cases, using them is a good habit. $ person="max and zach" $ echo $person max and zach $ person=max and zach bash: and: command not found When you reference a variable whose value contains TABs or multiple adjacent SPACEs, you need to use quotation marks to preserve the spacing. If you do not quote the variable, the shell collapses each string of blank characters into a single SPACE before passing the variable to the utility: $ person="max and $ echo $person max and zach $ echo "$person" max and zach Pathname expansion in assignments zach" When you execute a command with a variable as an argument, the shell replaces the name of the variable with the value of the variable and passes that value to the program being executed. If the value of the variable contains a special character, such as * or ?, the shell may expand that variable. The first line in the following sequence of commands assigns the string max* to the variable memo. The Bourne Again Shell does not expand the string because bash does not perform pathname expansion (page 136) when it assigns a value to a variable. All shells process a command line in a specific order. Within this order bash (but not tcsh) expands variables before it interprets commands. In the following echo command line, the double quotation marks quote the asterisk (*) in the expanded value of $memo and prevent bash from performing pathname expansion on the expanded memo variable before passing its value to the echo command: $ memo=max* $ echo "$memo" max* All shells interpret special characters as special when you reference a variable that contains an unquoted special character. In the following example, the shell expands the value of the memo variable because it is not quoted: $ ls max.report max.summary $ echo $memo max.report max.summary 294 Chapter 8 The Bourne Again Shell Here the shell expands the $memo variable to max*, expands max* to max.report and max.summary, and passes these two values to echo. optional Braces The $VARIABLE syntax is a special case of the more general syntax ${VARIABLE}, in which the variable name is enclosed by ${}. The braces insulate the variable name from adjacent characters. Braces are necessary when catenating a variable value with a string: $ $ $ $ PREF=counter WAY=$PREFclockwise FAKE=$PREFfeit echo $WAY $FAKE $ The preceding example does not work as planned. Only a blank line is output because, although the symbols PREFclockwise and PREFfeit are valid variable names, they are not set. By default the shell evaluates an unset variable as an empty (null) string and displays this value (bash) or generates an error message (tcsh). To achieve the intent of these statements, refer to the PREF variable using braces: $ PREF=counter $ WAY=${PREF}clockwise $ FAKE=${PREF}feit $ echo $WAY $FAKE counterclockwise counterfeit The Bourne Again Shell refers to the arguments on its command line by position, using the special variables $1, $2, $3, and so forth up to $9. If you wish to refer to arguments past the ninth argument, you must use braces: ${10}. The name of the command is held in $0 (page 441). unset: Removes a Variable Unless you remove a variable, it exists as long as the shell in which it was created exists. To remove the value of a variable but not the variable itself, assign a null value to the variable (use set person = in tcsh): $ person= $ echo $person $ You can remove a variable using the unset builtin. The following command removes the variable person: $ unset person Parameters and Variables 295 Variable Attributes This section discusses attributes and explains how to assign them to variables. readonly: Makes the Value of a Variable Permanent You can use the readonly builtin (not in tcsh) to ensure that the value of a variable cannot be changed. The next example declares the variable person to be readonly. You must assign a value to a variable before you declare it to be readonly; you cannot change its value after the declaration. When you attempt to unset or change the value of a readonly variable, the shell displays an error message: $ person=zach $ echo $person zach $ readonly person $ person=helen bash: person: readonly variable If you use the readonly builtin without an argument, it displays a list of all readonly shell variables. This list includes keyword variables that are automatically set as readonly as well as keyword or user-created variables that you have declared as readonly. See page 296 for an example (readonly and declare –r produce the same output). declare and typeset: Assign Attributes to Variables The declare and typeset builtins (two names for the same command, neither of which is available in tcsh) set attributes and values for shell variables. Table 8-3 lists five of these attributes. Table 8-3 Variable attributes (typeset or declare) Attribute Meaning –a Declares a variable as an array (page 434) –f Declares a variable to be a function name (page 327) –i Declares a variable to be of type integer (page 296) –r Makes a variable readonly; also readonly (page 295) –x Exports a variable (makes it global); also export (page 436) The following commands declare several variables and set some attributes. The first line declares person1 and assigns it a value of max. This command has the same effect with or without the word declare. $ $ $ $ declare declare declare declare person1=max -r person2=zach -rx person3=helen -x person4 296 Chapter 8 The Bourne Again Shell The readonly and export builtins are synonyms for the commands declare –r and declare –x, respectively. You can declare a variable without assigning a value to it, as the preceding declaration of the variable person4 illustrates. This declaration makes person4 available to all subshells (i.e., makes it global). Until an assignment is made to the variable, it has a null value. You can list the options to declare separately in any order. The following is equivalent to the preceding declaration of person3: $ declare -x -r person3=helen Use the + character in place of – when you want to remove an attribute from a variable. You cannot remove the readonly attribute. After the following command is given, the variable person3 is no longer exported but it is still readonly. $ declare +x person3 You can use typeset instead of declare. Listing variable attributes Without any arguments or options, declare lists all shell variables. The same list is output when you run set (page 442) without any arguments. If you use a declare builtin with options but no variable names as arguments, the command lists all shell variables that have the indicated attributes set. For example, the command declare –r displays a list of all readonly shell variables. This list is the same as that produced by the readonly command without any arguments. After the declarations in the preceding example have been given, the results are as follows: $ declare -r declare -ar BASH_VERSINFO='([0]="3" [1]="2" [2]="39" [3]="1" ...)' declare -ir EUID="500" declare -ir PPID="936" declare -r SHELLOPTS="braceexpand:emacs:hashall:histexpand:history:..." declare -ir UID="500" declare -r person2="zach" declare -rx person3="helen" The first five entries are keyword variables that are automatically declared as readonly. Some of these variables are stored as integers (–i). The –a option indicates that BASH_VERSINFO is an array variable; the value of each element of the array is listed to the right of an equal sign. Integer By default the values of variables are stored as strings. When you perform arithmetic on a string variable, the shell converts the variable into a number, manipulates it, and then converts it back to a string. A variable with the integer attribute is stored as an integer. Assign the integer attribute as follows: $ declare -i COUNT Keyword Variables Keyword variables either are inherited or are declared and initialized by the shell when it starts. You can assign values to these variables from the command line or from a startup file. Typically you want these variables to apply to all subshells you Parameters and Variables 297 start as well as to your login shell. For those variables not automatically exported by the shell, you must use export (bash; page 436) or setenv (tcsh; page 366) to make them available to child shells. HOME: Your Home Directory By default your home directory is the working directory when you log in. Your home directory is established when your account is set up; under Linux its name is stored in the /etc/passwd file. Mac OS X uses Open Directory (page 926) to store this information. $ grep sam /etc/passwd sam:x:501:501:Sam S. x301:/home/sam:/bin/bash When you log in, the shell inherits the pathname of your home directory and assigns it to the variable HOME. When you give a cd command without an argument, cd makes the directory whose name is stored in HOME the working directory: $ pwd /home/max/laptop $ echo $HOME /home/max $ cd $ pwd /home/max This example shows the value of the HOME variable and the effect of the cd builtin. After you execute cd without an argument, the pathname of the working directory is the same as the value of HOME: your home directory. Tilde (~) The shell uses the value of HOME to expand pathnames that use the shorthand tilde (~) notation (page 84) to denote a user’s home directory. The following example uses echo to display the value of this shortcut and then uses ls to list the files in Max’s laptop directory, which is a subdirectory of his home directory: $ echo ~ /home/max $ ls ~/laptop tester count lineup PATH: Where the Shell Looks for Programs When you give the shell an absolute or relative pathname rather than a simple filename as a command, it looks in the specified directory for an executable file with the specified filename. If the file with the pathname you specified does not exist, the shell reports command not found. If the file exists as specified but you do not have execute permission for it, or in the case of a shell script you do not have read and execute permission for it, the shell reports Permission denied. If you give a simple filename as a command, the shell searches through certain directories (your search path) for the program you want to execute. It looks in several directories for a file that has the same name as the command and that you have execute permission for (a compiled program) or read and execute permission for (a shell script). The PATH shell variable controls this search. 298 Chapter 8 The Bourne Again Shell The default value of PATH is determined when bash or tcsh is compiled. It is not set in a startup file, although it may be modified there. Normally the default specifies that the shell search several system directories used to hold common commands. These system directories include /bin and /usr/bin and other directories appropriate to the local system. When you give a command, if the shell does not find the executable—and, in the case of a shell script, readable—file named by the command in any of the directories listed in PATH, the shell generates one of the aforementioned error messages. Working directory The PATH variable specifies the directories in the order the shell should search them. Each directory must be separated from the next by a colon. The following command sets PATH so that a search for an executable file starts with the /usr/local/bin directory. If it does not find the file in this directory, the shell looks next in /bin, and then in /usr/bin. If the search fails in those directories, the shell looks in the ~/bin directory, a subdirectory of the user’s home directory. Finally the shell looks in the working directory. Exporting PATH makes its value accessible to subshells: $ export PATH=/usr/local/bin:/bin:/usr/bin:~/bin: A null value in the string indicates the working directory. In the preceding example, a null value (nothing between the colon and the end of the line) appears as the last element of the string. The working directory is represented by a leading colon (not recommended; see the following security tip), a trailing colon (as in the example), or two colons next to each other anywhere in the string. You can also represent the working directory explicitly with a period (.). See “PATH” on page 373 for a tcsh example. Because Linux stores many executable files in directories named bin (binary), users typically put their own executable files in their own ~/bin directories. If you put your own bin directory at the end of your PATH, as in the preceding example, the shell looks there for any commands that it cannot find in directories listed earlier in PATH. PATH and security security Do not put the working directory first in PATH when security is a concern. If you are working as root, you should never put the working directory first in PATH. It is common for root’s PATH to omit the working directory entirely. You can always execute a file in the working directory by prepending ./ to the name: ./myprog. Putting the working directory first in PATH can create a security hole. Most people type ls as the first command when entering a directory. If the owner of a directory places an executable file named ls in the directory, and the working directory appears first in a user’s PATH, the user giving an ls command from the directory executes the ls program in the working directory instead of the system ls utility, possibly with undesirable results. If you want to add directories to PATH, you can reference the old value of the PATH variable in setting PATH to a new value (but see the preceding security tip). The following command adds /usr/local/bin to the beginning of the current PATH and the bin directory in the user’s home directory (~/bin) to the end: $ PATH=/usr/local/bin:$PATH:~/bin Parameters and Variables 299 MAIL: Where Your Mail Is Kept The MAIL variable (mail under tcsh) contains the pathname of the file that holds your mail (your mailbox, usually /var/mail/name, where name is your username). If MAIL is set and MAILPATH (next) is not set, the shell informs you when mail arrives in the file specified by MAIL. In a graphical environment you can unset MAIL so the shell does not display mail reminders in a terminal emulator window (assuming you are using a graphical mail program). Most Mac OS X systems do not use local files for incoming mail; mail is typically kept on a remote mail server instead. The MAIL variable and other mail-related shell variables do not do anything unless you have a local mail server. The MAILPATH variable (not available under tcsh) contains a list of filenames separated by colons. If this variable is set, the shell informs you when any one of the files is modified (for example, when mail arrives). You can follow any of the filenames in the list with a question mark (?), followed by a message. The message replaces the you have mail message when you receive mail while you are logged in. The MAILCHECK variable (not available under tcsh) specifies how often, in seconds, the shell checks for new mail. The default is 60 seconds. If you set this variable to zero, the shell checks before each prompt. PS1: User Prompt (Primary) The default Bourne Again Shell prompt is a dollar sign ($). When you run bash with root privileges, bash typically displays a pound sign (#) prompt. The PS1 variable (prompt under tcsh, page 373) holds the prompt string that the shell uses to let you know that it is waiting for a command. When you change the value of PS1 or prompt, you change the appearance of your prompt. You can customize the prompt displayed by PS1. For example, the assignment $ PS1="[\u@\h \W \!]$ " displays the following prompt: [user@host directory event]$ where user is the username, host is the hostname up to the first period, directory is the basename of the working directory, and event is the event number (page 309) of the current command. If you are working on more than one system, it can be helpful to incorporate the system name into your prompt. For example, you might change the prompt to the name of the system you are using, followed by a colon and a SPACE (a SPACE at the end of the prompt makes the commands you enter after the prompt easier to read). This command uses command substitution (page 340) in the string assigned to PS1: $ PS1="$(hostname): " bravo.example.com: echo test test bravo.example.com: 300 Chapter 8 The Bourne Again Shell Use the following command under tcsh: tcsh $ set prompt = "`hostname`: " The first example that follows changes the prompt to the name of the local host, a SPACE, and a dollar sign (or, if the user is running with root privileges, a pound sign). The second example changes the prompt to the time followed by the name of the user. The third example changes the prompt to the one used in this book (a pound sign for root and a dollar sign otherwise): $ PS1='\h \$ ' bravo $ $ PS1='\@ \u $ ' 09:44 PM max $ $ PS1='\$ ' $ Table 8-4 describes some of the symbols you can use in PS1. See Table 9-4 on page 373 for the corresponding tcsh symbols. For a complete list of special characters you can use in the prompt strings, open the bash man page and search for the second occurrence of PROMPTING (give the command /PROMPTING and then press n). Table 8-4 PS1 symbols Symbol Display in prompt \$ # if the user is running with root privileges; otherwise, $ \w Pathname of the working directory \W Basename of the working directory \! Current event (history) number (page 313) \d Date in Weekday Month Date format \h Machine hostname, without the domain \H Full machine hostname, including the domain \u Username of the current user \@ Current time of day in 12-hour, AM/PM format \T Current time of day in 12-hour HH:MM:SS format \A Current time of day in 24-hour HH:MM format \t Current time of day in 24-hour HH:MM:SS format PS2: User Prompt (Secondary) The PS2 variable holds the secondary prompt (tcsh uses prompt2; page 374). On the first line of the next example, an unclosed quoted string follows echo. The shell assumes the command is not finished and, on the second line, gives the default secondary prompt (>). This prompt indicates the shell is waiting for the user to continue the command line. The shell waits until it receives the quotation mark that closes the string. Only then does it execute the command: Parameters and Variables 301 $ echo "demonstration of prompt string > 2" demonstration of prompt string 2 $ PS2="secondary prompt: " $ echo "this demonstrates secondary prompt: prompt string 2" this demonstrates prompt string 2 The second command changes the secondary prompt to secondary prompt: followed by a SPACE. A multiline echo demonstrates the new prompt. PS3: Menu Prompt The PS3 variable holds the menu prompt (tcsh uses prompt3; page 374) for the select control structure (page 428). PS4: Debugging Prompt The PS4 variable holds the bash debugging symbol (page 410; not under tcsh). IFS: Separates Input Fields (Word Splitting) The IFS (Internal Field Separator) shell variable (not under tcsh) specifies the characters you can use to separate arguments on a command line. It has the default value of SPACE TAB NEWLINE. Regardless of the value of IFS, you can always use one or more SPACE or TAB characters to separate arguments on the command line, provided these characters are not quoted or escaped. When you assign IFS character values, these characters can also separate fields—but only if they undergo expansion. This type of interpretation of the command line is called word splitting. Be careful when changing IFS caution Changing IFS has a variety of side effects, so work cautiously. You may find it useful to save the value of IFS before changing it. Then you can easily restore the original value if you get unexpected results. Alternatively, you can fork a new shell with a bash command before experimenting with IFS; if you get into trouble, you can exit back to the old shell, where IFS is working properly. The following example demonstrates how setting IFS can affect the interpretation of a command line: $ a=w:x:y:z $ cat $a cat: w:x:y:z: No such file or directory $ IFS=":" $ cat $a cat: w: No cat: x: No cat: y: No cat: z: No such such such such file file file file or or or or directory directory directory directory 302 Chapter 8 The Bourne Again Shell The first time cat is called, the shell expands the variable a, interpreting the string w:x:y:z as a single word to be used as the argument to cat. The cat utility cannot find a file named w:x:y:z and reports an error for that filename. After IFS is set to a colon (:), the shell expands the variable a into four words, each of which is an argument to cat. Now cat reports errors for four files: w, x, y, and z. Word splitting based on the colon (:) takes place only after the variable a is expanded. The shell splits all expanded words on a command line according to the separating characters found in IFS. When there is no expansion, there is no splitting. Consider the following commands: $ IFS="p" $ export VAR Although IFS is set to p, the p on the export command line is not expanded, so the word export is not split. The following example uses variable expansion in an attempt to produce an export command: $ IFS="p" $ aa=export $ echo $aa ex ort This time expansion occurs, so the character p in the token export is interpreted as a separator (as the echo command shows). Now when you try to use the value of the aa variable to export the VAR variable, the shell parses the $aa VAR command line as ex ort VAR. The effect is that the command line starts the ex editor with two filenames: ort and VAR. $ $aa VAR 2 files to edit "ort" [New File] Entering Ex mode. Type "visual" to go to Normal mode. :q E173: 1 more file to edit :q $ If you unset IFS, only SPACEs and TABs work as field separators. Multiple separator characters tip Although the shell treats sequences of multiple SPACE or TAB characters as a single separator, it treats each occurrence of another field-separator character as a separator. CDPATH: Broadens the Scope of cd The CDPATH variable (cdpath under tcsh) allows you to use a simple filename as an argument to the cd builtin to change the working directory to a directory other than a child of the working directory. If you have several directories you typically work out of, this variable can speed things up and save you the tedium of using cd with longer pathnames to switch among them. Parameters and Variables 303 When CDPATH or cdpath is not set and you specify a simple filename as an argument to cd, cd searches the working directory for a subdirectory with the same name as the argument. If the subdirectory does not exist, cd displays an error message. When CDPATH or cdpath is set, cd searches for an appropriately named subdirectory in the directories in the CDPATH list. If it finds one, that directory becomes the working directory. With CDPATH or cdpath set, you can use cd and a simple filename to change the working directory to a child of any of the directories listed in CDPATH or cdpath. The CDPATH or cdpath variable takes on the value of a colon-separated list of directory pathnames (similar to the PATH variable). It is usually set in the ~/.bash_profile (bash) or ~/.tcshrc (tcsh) startup file with a command line such as the following: export CDPATH=$HOME:$HOME/literature Use the following format for tcsh: setenv cdpath $HOME\:$HOME/literature These commands cause cd to search your home directory, the literature directory, and then the working directory when you give a cd command. If you do not include the working directory in CDPATH or cdpath, cd searches the working directory if the search of all the other directories in CDPATH or cdpath fails. If you want cd to search the working directory first, include a null string, represented by two colons (::), as the first entry in CDPATH: export CDPATH=::$HOME:$HOME/literature If the argument to the cd builtin is an absolute pathname—one starting with a slash (/)—the shell does not consult CDPATH or cdpath. Keyword Variables: A Summary Table 8-5 presents a list of bash keyword variables. See page 371 for information on tcsh variables. Table 8-5 bash keyword variables Variable Value BASH_ENV The pathname of the startup file for noninteractive shells (page 272) CDPATH The cd search path (page 302) COLUMNS The width of the display used by select (page 427) FCEDIT The name of the editor that fc uses by default (page 312) HISTFILE The pathname of the file that holds the history list (default: ~/.bash_history; page 308) HISTFILESIZE The maximum number of entries saved in HISTFILE (default: 500; page 308) HISTSIZE The maximum number of entries saved in the history list (default: 500; page 308) 304 Chapter 8 The Bourne Again Shell Table 8-5 bash keyword variables (continued) Variable Value HOME The pathname of the user’s home directory (page 297); used as the default argument for cd and in tilde expansion (page 84) IFS Internal Field Separator (page 301); used for word splitting (page 341) INPUTRC The pathname of the Readline startup file (default: ~/.inputrc; page 321) LANG The locale category when that category is not specifically set with an LC_* variable LC_* A group of variables that specify locale categories including LC_COLLATE, LC_CTYPE, LC_MESSAGES, and LC_NUMERIC; use the locale builtin to display a complete list with values LINES The height of the display used by select (page 427) MAIL The pathname of the file that holds a user’s mail (page 299) MAILCHECK How often, in seconds, bash checks for mail (page 299) MAILPATH A colon-separated list of file pathnames that bash checks for mail in (page 299) PATH A colon-separated list of directory pathnames that bash looks for commands in (page 297) PROMPT_COMMAND A command that bash executes just before it displays the primary prompt PS1 Prompt String 1; the primary prompt (page 299) PS2 Prompt String 2; the secondary prompt (default: '> '; page 300) PS3 The prompt issued by select (page 427) PS4 The bash debugging symbol (page 410) REPLY Holds the line that read accepts (page 448); also used by select (page 427) Special Characters Table 8-6 lists most of the characters that are special to the bash and tcsh shells. Table 8-6 Shell special characters Character Use NEWLINE Initiates execution of a command (page 282) ; Separates commands (page 282) Special Characters 305 Table 8-6 Shell special characters (continued) Character Use () Groups commands (page 284) for execution by a subshell or identifies a function (page 327) (( )) Expands an arithmetic expression (page 338) & Executes a command in the background (pages 134 and 283) | Sends standard output of the preceding command to standard input of the following command (pipe; page 283) > Redirects standard output (page 126) >> Appends standard output (page 130) /tmp/saveit $ cat /tmp/saveit cat: /tmp/saveit: No such file or directory In fact, the shell does not redirect the output—it recognizes input and output redirection before it evaluates variables. When it executes the command line, the shell checks for redirection and, finding none, evaluates the SENDIT variable. After 336 Chapter 8 The Bourne Again Shell replacing the variable with > /tmp/saveit, bash passes the arguments to echo, which dutifully copies its arguments to standard output. No /tmp/saveit file is created. The following sections provide more detailed descriptions of the steps involved in command processing. Keep in mind that double and single quotation marks cause the shell to behave differently when performing expansions. Double quotation marks permit parameter and variable expansion but suppress other types of expansion. Single quotation marks suppress all types of expansion. Brace Expansion Brace expansion, which originated in the C Shell, provides a convenient way to specify filenames when pathname expansion does not apply. Although brace expansion is almost always used to specify filenames, the mechanism can be used to generate arbitrary strings; the shell does not attempt to match the brace notation with the names of existing files. Brace expansion is turned on in interactive and noninteractive shells by default; you can turn it off with set +o braceexpand. The shell also uses braces to isolate variable names (page 294). The following example illustrates how brace expansion works. The ls command does not display any output because there are no files in the working directory. The echo builtin displays the strings that the shell generates with brace expansion. In this case the strings do not match filenames (because there are no files in the working directory). $ ls $ echo chap_{one,two,three}.txt chap_one.txt chap_two.txt chap_three.txt The shell expands the comma-separated strings inside the braces in the echo command into a SPACE-separated list of strings. Each string from the list is prepended with the string chap_, called the preamble, and appended with the string .txt, called the postscript. Both the preamble and the postscript are optional. The left-to-right order of the strings within the braces is preserved in the expansion. For the shell to treat the left and right braces specially and for brace expansion to occur, at least one comma and no unquoted whitespace characters must be inside the braces. You can nest brace expansions. Brace expansion is useful when there is a long preamble or postscript. The following example copies four files—main.c, f1.c, f2.c, and tmp.c—located in the /usr/local/src/C directory to the working directory: $ cp /usr/local/src/C/{main,f1,f2,tmp}.c . You can also use brace expansion to create directories with related names: $ ls -F file1 file2 file3 $ mkdir vrs{A,B,C,D,E} $ ls -F file1 file2 file3 vrsA/ vrsB/ vrsC/ vrsD/ vrsE/ Processing the Command Line 337 The –F option causes ls to display a slash (/) after a directory and an asterisk (*) after an executable file. If you tried to use an ambiguous file reference instead of braces to specify the directories, the result would be different (and not what you wanted): $ rmdir vrs* $ mkdir vrs[A-E] $ ls -F file1 file2 file3 vrs[A-E]/ An ambiguous file reference matches the names of existing files. In the preceding example, because it found no filenames matching vrs[A–E], bash passed the ambiguous file reference to mkdir, which created a directory with that name. Brackets in ambiguous file references are discussed on page 139. Tilde Expansion Chapter 4 introduced a shorthand notation to specify your home directory or the home directory of another user. This section provides a more detailed explanation of tilde expansion. The tilde (~) is a special character when it appears at the start of a token on a command line. When it sees a tilde in this position, bash looks at the following string of characters—up to the first slash (/) or to the end of the word if there is no slash—as a possible username. If this possible username is null (that is, if the tilde appears as a word by itself or if it is immediately followed by a slash), the shell substitutes the value of the HOME variable for the tilde. The following example demonstrates this expansion, where the last command copies the file named letter from Max’s home directory to the working directory: $ echo $HOME /home/max $ echo ~ /home/max $ echo ~/letter /home/max/letter $ cp ~/letter . If the string of characters following the tilde forms a valid username, the shell substitutes the path of the home directory associated with that username for the tilde and name. If the string is not null and not a valid username, the shell does not make any substitution: $ echo ~zach /home/zach $ echo ~root /root $ echo ~xx ~xx 338 Chapter 8 The Bourne Again Shell Tildes are also used in directory stack manipulation (page 288). In addition, ~+ is a synonym for PWD (the name of the working directory), and ~– is a synonym for OLDPWD (the name of the previous working directory). Parameter and Variable Expansion On a command line, a dollar sign ($) that is not followed by an open parenthesis introduces parameter or variable expansion. Parameters include both command-line, or positional, parameters (page 440) and special parameters (page 438). Variables include both user-created variables (page 292) and keyword variables (page 296). The bash man and info pages do not make this distinction. Parameters and variables are not expanded if they are enclosed within single quotation marks or if the leading dollar sign is escaped (i.e., preceded with a backslash). If they are enclosed within double quotation marks, the shell expands parameters and variables. Arithmetic Expansion The shell performs arithmetic expansion by evaluating an arithmetic expression and replacing it with the result. See page 368 for information on arithmetic expansion under tcsh. Under bash the syntax for arithmetic expansion is $((expression)) The shell evaluates expression and replaces $((expression)) with the result of the evaluation. This syntax is similar to the syntax used for command substitution [$(...)] and performs a parallel function. You can use $((expression)) as an argument to a command or in place of any numeric value on a command line. The rules for forming expression are the same as those found in the C programming language; all standard C arithmetic operators are available (see Table 10-8 on page 463). Arithmetic in bash is done using integers. Unless you use variables of type integer (page 296) or actual integers, however, the shell must convert stringvalued variables to integers for the purpose of the arithmetic evaluation. You do not need to precede variable names within expression with a dollar sign ($). In the following example, after read (page 447) assigns the user’s response to age, an arithmetic expression determines how many years are left until age 60: $ cat age_check #!/bin/bash echo -n "How old are you? " read age echo "Wow, in $((60-age)) years, you'll be 60!" $ ./age_check How old are you? 55 Wow, in 5 years, you'll be 60! You do not need to enclose the expression within quotation marks because bash does not perform filename expansion on it. This feature makes it easier for you to use an asterisk (*) for multiplication, as the following example shows: Processing the Command Line 339 $ echo There are $((60*60*24*365)) seconds in a non-leap year. There are 31536000 seconds in a non-leap year. The next example uses wc, cut, arithmetic expansion, and command substitution (page 340) to estimate the number of pages required to print the contents of the file letter.txt. The output of the wc (word count) utility (page 876) used with the –l option is the number of lines in the file, in columns (character positions) 1 through 4, followed by a SPACE and the name of the file (the first command following). The cut utility (page 652) with the –c1–4 option extracts the first four columns. $ wc -l letter.txt 351 letter.txt $ wc -l letter.txt | cut -c1-4 351 The dollar sign and single parenthesis instruct the shell to perform command substitution; the dollar sign and double parentheses indicate arithmetic expansion: $ echo $(($(wc -l letter.txt | cut -c1-4)/66 + 1)) 6 The preceding example sends standard output from wc to standard input of cut via a pipe. Because of command substitution, the output of both commands replaces the commands between the $(and the matching) on the command line. Arithmetic expansion then divides this number by 66, the number of lines on a page. A 1 is added because the integer division results in any remainder being discarded. Fewer dollar signs ($) tip When you use variables within $((and)), the dollar signs that precede individual variable references are optional: $ x=23 y=37 $ echo $((2*$x + 3*$y)) 157 $ echo $((2*x + 3*y)) 157 Another way to get the same result without using cut is to redirect the input to wc instead of having wc get its input from a file you name on the command line. When you redirect its input, wc does not display the name of the file: $ wc -l < letter.txt 351 It is common practice to assign the result of arithmetic expansion to a variable: $ numpages=$(($(wc -l < letter.txt)/66 + 1)) let builtin The let builtin (not available in tcsh) evaluates arithmetic expressions just as the $(( )) syntax does. The following command is equivalent to the preceding one: $ let "numpages=$(wc -l < letter.txt)/66 + 1" 340 Chapter 8 The Bourne Again Shell The double quotation marks keep the SPACEs (both those you can see and those that result from the command substitution) from separating the expression into separate arguments to let. The value of the last expression determines the exit status of let. If the value of the last expression is 0, the exit status of let is 1; otherwise, its exit status is 0. You can supply let with multiple arguments on a single command line: $ let a=5+3 b=7+2 $ echo $a $b 8 9 When you refer to variables when doing arithmetic expansion with let or $(( )), the shell does not require a variable name to begin with a dollar sign ($). Nevertheless, it is a good practice to do so for consistency, as in most places you must precede a variable name with a dollar sign. Command Substitution Command substitution replaces a command with the output of that command. The preferred syntax for command substitution under bash follows: $(command) Under bash you can also use the following, older syntax, which is the only syntax allowed under tcsh: ‘command’ The shell executes command within a subshell and replaces command, along with the surrounding punctuation, with standard output of command. In the following example, the shell executes pwd and substitutes the output of the command for the command and surrounding punctuation. Then the shell passes the output of the command, which is now an argument, to echo, which displays it. $ echo $(pwd) /home/max The next script assigns the output of the pwd builtin to the variable where and displays a message containing the value of this variable: $ cat where where=$(pwd) echo "You are using the $where directory." $ ./where You are using the /home/zach directory. Although it illustrates how to assign the output of a command to a variable, this example is not realistic. You can more directly display the output of pwd without using a variable: $ cat where2 echo "You are using the $(pwd) directory." $ ./where2 You are using the /home/zach directory. Processing the Command Line 341 The following command uses find to locate files with the name README in the directory tree rooted at the working directory. This list of files is standard output of find and becomes the list of arguments to ls. $ ls -l $(find . -name README -print) The next command line shows the older ‘command’ syntax: $ ls -l ‘find . -name README -print’ One advantage of the newer syntax is that it avoids the rather arcane rules for token handling, quotation mark handling, and escaped back ticks within the old syntax. Another advantage of the new syntax is that it can be nested, unlike the old syntax. For example, you can produce a long listing of all README files whose size exceeds the size of ./README with the following command: $ ls -l $(find . -name README -size +$(echo $(cat ./README | wc -c)c) -print) Try giving this command after giving a set –x command (page 410) to see how bash expands it. If there is no README file, you just get the output of ls –l. For additional scripts that use command substitution, see pages 406, 425, and 455. $((Versus $(tip The symbols $((constitute a single token. They introduce an arithmetic expression, not a command substitution. Thus, if you want to use a parenthesized subshell (page 284) within $(), you must insert a SPACE between the $(and the following (. Word Splitting The results of parameter and variable expansion, command substitution, and arithmetic expansion are candidates for word splitting. Using each character of IFS (page 301) as a possible delimiter, bash splits these candidates into words or tokens. If IFS is unset, bash uses its default value (SPACE-TAB-NEWLINE). If IFS is null, bash does not split words. Pathname Expansion Pathname expansion (page 136), also called filename generation or globbing, is the process of interpreting ambiguous file references and substituting the appropriate list of filenames. Unless noglob (page 333) is set, the shell performs this function when it encounters an ambiguous file reference—a token containing any of the unquoted characters *, ?, [, or]. If bash cannot locate any files that match the specified pattern, the token with the ambiguous file reference is left alone. The shell does not delete the token or replace it with a null string but rather passes it to the program as is (except see nullglob on page 333). The TC Shell generates an error message. In the first echo command in the following example, the shell expands the ambiguous file reference tmp* and passes three tokens (tmp1, tmp2, and tmp3) to echo. The echo builtin displays the three filenames it was passed by the shell. After rm 342 Chapter 8 The Bourne Again Shell removes the three tmp* files, the shell finds no filenames that match tmp* when it tries to expand it. It then passes the unexpanded string to the echo builtin, which displays the string it was passed. $ ls tmp1 tmp2 tmp3 $ echo tmp* tmp1 tmp2 tmp3 $ rm tmp* $ echo tmp* tmp* By default the same command causes the TC Shell to display an error message: tcsh $ echo tmp* echo: No match A period that either starts a pathname or follows a slash (/) in a pathname must be matched explicitly unless you have set dotglob (page 332). The option nocaseglob (page 333) causes ambiguous file references to match filenames without regard to case. Quotation marks Putting double quotation marks around an argument causes the shell to suppress pathname and all other kinds of expansion except parameter and variable expansion. Putting single quotation marks around an argument suppresses all types of expansion. The second echo command in the following example shows the variable $max between double quotation marks, which allow variable expansion. As a result the shell expands the variable to its value: sonar. This expansion does not occur in the third echo command, which uses single quotation

marks. Because neither single nor double quotation marks allow pathname expansion, the last two commands display the unexpanded argument tmp*. $ echo tmp* $max tmp1 tmp2 tmp3 sonar $ echo "tmp* $max" tmp* sonar $ echo 'tmp* $max' tmp* $max The shell distinguishes between the value of a variable and a reference to the variable and does not expand ambiguous file references if they occur in the value of a variable. As a consequence you can assign to a variable a value that includes special characters, such as an asterisk (*). Levels of expansion In the next example, the working directory has three files whose names begin with letter. When you assign the value letter* to the variable var, the shell does not expand the ambiguous file reference because it occurs in the value of a variable (in the assignment statement for the variable). No quotation marks surround the string letter*; context alone prevents the expansion. After the assignment the set builtin (with the help of grep) shows the value of var to be letter*. $ ls letter* letter1 letter2 $ var=letter* letter3 Chapter Summary 343 $ set | grep var var='letter*' $ echo '$var' $var $ echo "$var" letter* $ echo $var letter1 letter2 letter3 The three echo commands demonstrate three levels of expansion. When $var is quoted with single quotation marks, the shell performs no expansion and passes the character string $var to echo, which displays it. With double quotation marks, the shell performs variable expansion only and substitutes the value of the var variable for its name, preceded by a dollar sign. No pathname expansion is performed on this command because double quotation marks suppress it. In the final command, the shell, without the limitations of quotation marks, performs variable substitution and then pathname expansion before passing the arguments to echo. Process Substitution A special feature of the Bourne Again Shell is the ability to replace filename arguments with processes. An argument with the syntax (command) is replaced by the name of a pipe that command reads as standard input. The following example uses sort (pages 54 and 817) with the –m (merge, which works correctly only if the input files are already sorted) option to combine two word lists into a single list. Each word list is generated by a pipe that extracts words matching a pattern from a file and sorts the words in that list. $ sort -m -f > $file echo -n "Enter name of person or group: " read name echo "$name" >> $file echo >> $file cat >> $file echo "--" >> $file echo >> $file a. What do you have to do to the script to be able to execute it? b. Why does the script use the read builtin the first time it accepts input from the terminal and the cat utility the second time? 6. Assume the /home/zach/grants/biblios and /home/zach/biblios directories exist. Give Zach's working directory after he executes each sequence of commands given. Explain what happens in each case. a. $ pwd /home/zach/grants $ CDPATH=$(pwd) $ cd $ cd biblios Advanced Exercises b. $ pwd /home/zach/grants $ CDPATH=$(pwd) $ cd $HOME/biblios 7. Name two ways you can identify the PID number of the login shell. 8. Give the following command: $ sleep 30 | cat /etc/inittab Is there any output from sleep? Where does cat get its input from? What has to happen before the shell displays another prompt? Advanced Exercises 9. Write a sequence of commands or a script that demonstrates variable expansion occurs before pathname expansion. 10. Write a shell script that outputs the name of the shell executing it. 11. Explain the behavior of the following shell script: $ cat quote_demo twoliner="This is line 1. This is line 2." echo "$twoliner" echo $twoliner a. How many arguments does each echo command see in this script? Explain. b. Redefine the IFS shell variable so that the output of the second echo is the same as the first. 12. Add the exit status of the previous command to your prompt so that it behaves similarly to the following: $ [0] ls xxx ls: xxx: No such file or directory $ [1] 13. The dirname utility treats its argument as a pathname and writes to standard output the path prefix—that is, everything up to but not including the last component: $ dirname a/b/c/d a/b/c If you give dirname a simple filename (no / characters) as an argument, dirname writes a . to standard output: $ dirname simple . 347 348 Chapter 8 The Bourne Again Shell Implement dirname as a bash function. Make sure that it behaves sensibly when given such arguments as /. 14. Implement the basename utility, which writes the last component of its pathname argument to standard output, as a bash function. For example, given the pathname a/b/c/d, basename writes d to standard output: $ basename a/b/c/d d 15. The Linux basename utility has an optional second argument. If you give the command basename path suffix, basename removes the suffix and the prefix from path: $ basename src/shellfiles/prog.bash .bash prog $ basename src/shellfiles/prog.bash .c prog.bash Add this feature to the function you wrote for exercise 14. 9 The TC Shell In This Chapter

The TC Shell (tcsh) performs the same function as the Bourne Again Shell and other shells: It provides an interface between you and the Linux operating system. The TC Shell is an interactive command interpreter as well as a high-level programming language. Although you use only one shell at any given time, you should be able to switch back and forth comfortably between shells as the need arises. In fact, you may want to run different shells in different windows. Chapters 8 and 10 apply to tcsh as well as to bash so they provide a good background for this chapter. This chapter explains tcsh features that are not found in bash and those that are implemented differently from their bash counterparts. The tcsh home page is www.tcsh.org. 9Chapter9 The TC Shell is an expanded version of the C Shell (csh), which originated on Berkeley UNIX. The "T" in TC Shell comes from the TENEX and TOPS-20 operating systems, which inspired command completion and other features in the TC Shell. A number of features not found in csh are present in tcsh, including file and username completion, command-line editing, and spelling correction. As with csh, you can customize tcsh to 349 350 Chapter 9 The TC Shell make it more tolerant of mistakes and easier to use. By setting the proper shell variables, you can have tcsh warn you when you appear to be accidentally logging out or overwriting a file. Many popular features of the original C Shell are now shared by bash and tcsh. Assignment statement Although some of the functionality of tcsh is present in bash, differences arise in the syntax of some commands. For example, the tcsh assignment statement has the following syntax: set variable = value Having SPACEs on either side of the equal sign, although illegal in bash, is allowed in tcsh. By convention shell variables in tcsh are generally named with lowercase letters, not uppercase (you can use either). If you reference an undeclared variable (one that has had no value assigned to it), tcsh generates an error message, whereas bash does not. Finally the default tcsh prompt is a greater than sign (>), but it is frequently set to a single $ character followed by a SPACE. The examples in this chapter use a prompt of tcsh $ to avoid confusion with the bash prompt. Do not use tcsh as a programming language tip If you have used UNIX and are comfortable with the C or TC Shell, you may want to use tcsh as your login shell. However, you may find that the TC Shell is not as good a programming language as bash. If you are going to learn only one shell programming language, learn bash. The Bourne Again Shell and dash (page 270), which is a subset of bash, are used throughout Linux to program many system administration scripts. Shell Scripts The TC Shell can execute files containing tcsh commands, just as the Bourne Again Shell can execute files containing bash commands. Although the concepts of writing and executing scripts in the two shells are similar, the methods of declaring and assigning values to variables and the syntax of control structures are different. You can run bash and tcsh scripts while using any one of the shells as a command interpreter. Various methods exist for selecting the shell that runs a script. Refer to "#! Specifies a Shell" on page 280 for more information. If the first character of a shell script is a pound sign (#) and the following character is not an exclamation point (!), the TC Shell executes the script under tcsh. If the first character is anything other than #, tcsh calls the sh link to dash or bash to execute the script. echo: getting rid of the RETURN tip The tcsh echo builtin accepts either a –n option or a trailing \c to get rid of the RETURN that echo normally displays at the end of a line. The bash echo builtin accepts only the –n option (refer to "read: Accepts User Input" on page 447). Entering and Leaving the TC Shell 351 Shell game tip When you are working with an interactive TC Shell, if you run a script in which # is not the first character of the script and you call the script directly (without preceding its name with tcsh), tcsh calls the sh link to dash or bash to run the script. The following script was written to be run under tcsh but, when called from a tcsh command line, is executed by bash. The set builtin (page 442) works differently under bash and tcsh. As a result the following example (from page 371) issues a prompt but does not wait for you to respond: tcsh $ cat user_in echo -n "Enter input: " set input_line = "$&2" echo "cat >$i >>$userline$userline>>> finished processing line #", NR print "" } $ gawk -f g5 < alpha line # 1 aaaaaaaaa >>>> finished processing line # 1 line # 2 bbbbbbbbb skip this line: ccccccccc previous line began with: bbbbbbbbb >>>> finished processing line # 3 line # 4 ddddddddd >>>> finished processing line # 4 Coprocess: Two-Way I/O A coprocess is a process that runs in parallel with another process. Starting with version 3.1, gawk can invoke a coprocess to exchange information directly with a background process. A coprocess can be useful when you are working in a client/server environment, setting up an SQL (page 980) front end/back end, or exchanging data with a remote system over a network. The gawk syntax identifies a coprocess by preceding the name of the program that starts the background process with a |& operator. Advanced gawk Programming 561 Only gawk supports coprocesses tip The awk and mawk utilities do not support coprocesses. Only gawk supports coprocesses. The coprocess command must be a filter (i.e., it reads from standard input and writes to standard output) and must flush its output whenever it has a complete line rather than accumulating lines for subsequent output. When a command is invoked as a coprocess, it is connected via a two-way pipe to a gawk program so you can read from and write to the coprocess. to_upper When used alone the tr utility (page 864) does not flush its output after each line. The to_upper shell script is a wrapper for tr that does flush its output; this filter can be run as a coprocess. For each line read, to_upper writes the line, translated to uppercase, to standard output. Remove the # before set –x if you want to_upper to display debugging output. $ cat to_upper #!/bin/bash #set -x while read arg do echo "$arg" | tr '[a-z]' '[A-Z]' done $ echo abcdef | ./to_upper ABCDEF The g6 program invokes to_upper as a coprocess. This gawk program reads standard input or a file specified on the command line, translates the input to uppercase, and writes the translated data to standard output. $ cat g6 { print $0 |& "to_upper" "to_upper" |& getline hold print hold } $ gawk -f g6 < alpha AAAAAAAAA BBBBBBBBB CCCCCCCCC DDDDDDDDD The g6 program has one compound statement, enclosed within braces, comprising three statements. Because there is no pattern, gawk executes the compound statement once for each line of input. In the first statement, print $0 sends the current record to standard output. The |& operator redirects standard output to the program named to_upper, which is running as a coprocess. The quotation marks around the name of the program are required. The second statement redirects standard output from to_upper to a getline statement, which copies its standard input to the variable named hold. The third statement, print hold, sends the contents of the hold variable to standard output. 562 Chapter 12 The AWK Pattern Processing Language Getting Input from a Network Building on the concept of a coprocess, gawk can exchange information with a process on another system via an IP network connection. When you specify one of the special filenames that begins with /inet/, gawk processes the request using a network connection. The format of these special filenames is /inet/protocol/local-port/remote-host/remote-port where protocol is usually tcp but can be udp, local-port is 0 (zero) if you want gawk to pick a port (otherwise it is the number of the port you want to use), remote-host is the IP address (page 960) or fully qualified domain name (page 955) of the remote host, and remote-port is the port number on the remote host. Instead of a port number in local-port and remote-port, you can specify a service name such as http or ftp. The g7 program reads the rfc-retrieval.txt file from the server at www.rfc-editor.org. On www.rfc-editor.org the file is located at /rfc/rfc-retrieval.txt. The first statement in g7 assigns the special filename to the server variable. The filename specifies a TCP connection, allows the local system to select an appropriate port, and connects to www.rfc-editor.org on port 80. You can use http in place of 80 to specify the standard HTTP port. The second statement uses a coprocess to send a GET request to the remote server. This request includes the pathname of the file gawk is requesting. A while loop uses a coprocess to redirect lines from the server to getline. Because getline has no variable name as an argument, it saves its input in the current record buffer $0. The final print statement sends each record to standard output. Experiment with this script, replacing the final print statement with gawk statements that process the file. $ cat g7 BEGIN { # set variable named server # to special networking filename server = "/inet/tcp/0/www.rfc-editor.org/80" # use coprocess to send GET request to remote server print "GET /rfc/rfc-retrieval.txt" |& server # while loop uses coprocess to redirect # output from server to getline while (server |& getline) print $0 } Exercises 563 $ gawk -f g7 Where and how to get new RFCs ============================ RFCs may be obtained via FTP or HTTP or email from many RFC repositories. The official repository for RFCs is: ... Chapter Summary AWK is a pattern-scanning and processing language that searches one or more files for records (usually lines) that match specified patterns. It processes lines by performing actions, such as writing the record to standard output or incrementing a counter, each time it finds a match. AWK has several implementations, including awk, gawk, and mawk. An AWK program consists of one or more lines containing a pattern and/or action in the following format: pattern { action } The pattern selects lines from the input. An AWK program performs the action on all lines that the pattern selects. If a program line does not contain a pattern, AWK selects all lines in the input. If a program line does not contain an action, AWK copies the selected lines to standard output. An AWK program can use variables, functions, arithmetic operators, associative arrays, control statements, and C's printf statement. Advanced AWK programming takes advantage of getline statements to fine-tune input, coprocesses to enable gawk to exchange data with other programs (gawk only), and network connections to exchange data with programs running on remote systems on a network (gawk only). Exercises 1. Write an AWK program that numbers each line in a file and sends its output to standard output. 2. Write an AWK program that displays the number of characters in the first field followed by the first field and sends its output to standard output. 564 Chapter 12 The AWK Pattern Processing Language 3. Write an AWK program that uses the cars file (page 541), displays all cars priced at more than $5,000, and sends its output to standard output. 4. Use AWK to determine how many lines in /usr/share/dict/words contain the string abul. Verify your answer using grep. Advanced Exercises 5. Experiment with pgawk (available only with gawk). What does it do? How can it be useful? 6. Write a gawk (not awk or mawk) program named net_list that reads from the rfc-retrieval.txt file on www.rfc-editor.org (see "Getting Input from a Network" on page 562) and displays a the last word on each line in all uppercase letters. 7. Expand the net_list program developed in Exercise 6 to use to_upper (page 561) as a coprocess to display the list of cars with only the make of the cars in uppercase. The model and subsequent fields on each line should appear as they do in the cars file. 8. How can you get gawk (not awk or mawk) to neatly format—that is, "pretty print"—a gawk program file? (Hint: See the gawk man page.) 13 The sed Editor In This Chapter

The sed (stream editor) utility is a batch (noninteractive) editor. It transforms an input stream that can come from a file or standard input. It is frequently used as a filter in a pipe. Because it makes only one pass through its input, sed is more efficient than an interactive editor such as ed. Most Linux distributions provide GNU sed; Mac OS X supplies BSD sed. This chapter applies to both versions. 13Chapter13

565565 566 Chapter 13 The sed Editor Syntax A sed command line has the following syntax: sed [–n] program [file-list] sed [–n] –f program-file [file-list] The sed utility takes its input from files you specify on the command line or from standard input. Output from sed goes to standard output. Arguments The program is a sed program included on the command line. The first format allows you to write simple, short sed programs without creating a separate file to hold the sed program. The program-file in the second format is the pathname of a file containing a sed program (see "Editor Basics"). The file-list contains pathnames of the ordinary files that sed processes; these are the input files. When you do not specify a file-list, sed takes its input from standard input. Options Options preceded by a double hyphen (––) work under Linux (GNU sed) only. Options named with a single letter and preceded by a single hyphen work under Linux (GNU sed) and OS X (BSD sed). ––file program-file –f program-file Causes sed to read its program from the file named program-file instead of from the command line. You can use this option more than once on the command line. Summarizes how to use sed. L ––help ––in-place[=suffix] –i[suffix] Edits files in place. Without this option sed sends its output to standard output. With this option sed replaces the file it is processing with its output. When you specify a suffix, sed makes a backup of the original file. The backup has the original filename with suffix appended. You must include a period in suffix if you want a period to appear between the original filename and suffix. ––quiet or ––silent –n Causes sed not to copy lines to standard output except as specified by the Print (p) instruction or flag. Editor Basics 567 Editor Basics A sed program consists of one or more lines with the following syntax: [address[,address]] instruction [argument-list] The addresses are optional. If you omit the address, sed processes all lines of input. The instruction is an editing instruction that modifies the text. The addresses select the line(s) the instruction part of the command operates on. The number and kinds of arguments in the argument-list depend on the instruction. If you want to put several sed commands on one line, separate the commands with semicolons (;). The sed utility processes input as follows: 1. Reads one line of input from file-list or standard input. 2. Reads the first instruction from the program or program-file. If the address(es) select the input line, acts on the input line as the instruction specifies. 3. Reads the next instruction from the program or program-file. If the address(es) select the input line, acts on the input line (possibly modified by the previous instruction) as the instruction specifies. 4. Repeats step 3 until it has executed all instructions in the program or program-file. 5. Starts over with step 1 if there is another line of input; otherwise, sed is finished. Addresses A line number is an address that selects a line. As a special case, the line number $ represents the last line of input. A regular expression (Appendix A) is an address that selects those lines containing a string that the regular expression matches. Although slashes are often used to delimit these regular expressions, sed permits you to use any character other than a backslash or NEWLINE for this purpose. Except as noted, zero, one, or two addresses (either line numbers or regular expressions) can precede an instruction. If you do not specify an address, sed selects all lines, causing the instruction to act on every line of input. Specifying one address causes the instruction to act on each input line the address selects. Specifying two addresses causes the instruction to act on groups of lines. In this case the first address selects the first line in the first group. The second address selects the next 568 Chapter 13 The sed Editor subsequent line that it matches; this line is the last line in the first group. If no match for the second address is found, the second address points to the end of the file. After selecting the last line in a group, sed starts the selection process over, looking for the next line the first address matches. This line is the first line in the next group. The sed utility continues this process until it has finished going through the entire file. Instructions Pattern space The sed utility has two buffers. The following commands work with the Pattern space, which initially holds the line of input that sed just read. The other buffer, the Hold space, is discussed on page 570. a (append) The Append instruction appends one or more lines to the currently selected line. If you precede an Append instruction with two addresses, it appends to each line that is selected by the addresses. If you do not precede an Append instruction with an address, it appends to each input line. An Append instruction has the following format: [address[,address]] a\ text \ text \ ... text You must end each line of appended text, except the last, with a backslash, which quotes the following NEWLINE. The appended text concludes with a line that does not end with a backslash. The sed utility always writes out appended text, regardless of whether you use a –n flag on the command line. It even writes out the text if you delete the line to which you are appending the text. c (change) The Change instruction is similar to Append and Insert except it changes the selected lines so that they contain the new text. When you specify an address range, Change replaces the range of lines with a single occurrence of the new text. d (delete) The Delete instruction causes sed not to write out the lines it selects and not to finish processing the lines. After sed executes a Delete instruction, it reads the next input line and then begins anew with the first instruction from the program or program-file. i (insert) The Insert instruction is identical to the Append instruction except it places the new text before the selected line. N (next without write) The Next (N) instruction reads the next input line and appends it to the current line. An embedded NEWLINE separates the original line and the new line. You can use the N command to remove NEWLINEs from a file. See the example on page 575. Editor Basics 569 n (next) The Next (n) instruction writes out the currently selected line if appropriate, reads the next input line, and starts processing the new line with the next instruction from the program or program-file. p (print) The Print instruction writes the selected lines to standard output, writing the lines immediately, and does not reflect the effects of subsequent instructions. This instruction overrides the –n option on the command line. q (quit) The Quit instruction causes sed to terminate immediately. r file (read) The Read instruction reads the contents of the specified file and appends it to the selected line. A single instruction. SPACE and the name of the input file must follow a Read s (substitute) The Substitute instruction in sed is similar to that in vim (page 176). It has the following format: [address[,address]] s/pattern/replacement-string/[g][p][w file] The pattern is a regular expression (Appendix A) that traditionally is delimited by a slash (/); you can use any character other than a SPACE or NEWLINE. The replacementstring starts immediately following the second delimiter and must be terminated by the same delimiter. The final (third) delimiter is required. The replacement-string can contain an ampersand (&), which sed replaces with the matched pattern. Unless you use the g flag, the Substitute instruction replaces only the first occurrence of the pattern on each selected line. The g (global) flag causes the Substitute instruction to replace all nonoverlapping occurrences of the pattern on the selected lines. The p (print) flag causes sed to send all lines on which it makes substitutions to standard output. This flag overrides the –n option on the command line. The w (write) flag is similar to the p flag but sends its output to the file specified by file. A single SPACE and the name of the output file must follow a w flag. w file (write) This instruction is similar to the Print instruction except it sends the output to the file specified by file. A single SPACE and the name of the output file must follow a Write instruction. Control Structures ! (NOT) Causes sed to apply the following instruction, located on the same line, to each of the lines not selected by the address portion of the instruction. For example, 3!d deletes all lines except line 3 and $!p displays all lines except the last. { } (group instructions) When you enclose a group of instructions within a pair of braces, a single address or address pair selects the lines on which the group of instructions operates. Use semicolons (;) to separate multiple commands appearing on a single line. 570 Chapter 13 The sed Editor Branch instructions The GNU sed info page identifies the branch instructions as "Commands for sed gurus" and suggests that if you need them you might be better off writing your program in awk or Perl. : label Identifies a location within a sed program. The label is useful as a target for the b and t branch instructions. b [label] Unconditionally transfers control to label. Without label, skips the rest of the instructions for the current line of input and reads the next line of input. t [label] Transfers control to label only if a Substitute instruction has been successful since the most recent line of input was read (conditional branch). Without label, skips the rest of the instructions for the current line of input and reads the next line of input. The Hold Space The commands reviewed up to this point work with the Pattern space, a buffer that initially holds the line of input that sed just read. The Hold space can hold data while you manipulate data in the Pattern space; it is a temporary buffer. Until you place data in the Hold space, it is empty. This section discusses commands that move data between the Pattern space and the Hold space. g Copies the contents of the Hold space to the Pattern space. The original contents of the Pattern space is lost. G Appends a NEWLINE and the contents of the Hold space to the Pattern space. h Copies the contents of the Pattern space to the Hold space. The original contents of the Hold space is lost. H Appends a NEWLINE and the contents of the Pattern space to the Hold space. x Exchanges the contents of the Pattern space and the Hold space. Examples lines data file The following examples use the lines file for input: $ cat lines Line one. The second line. The third. This is line four. Five. This is the sixth sentence. This is line seven. Eighth and last. Unless you instruct it not to, sed sends all lines—selected or not—to standard output. When you use the –n option on the command line, sed sends only certain lines, such as those selected by a Print (p) instruction, to standard output. Examples 571 The following command line displays all lines in the lines file that contain the word line (all lowercase). In addition, because there is no –n option, sed displays all the lines of input. As a result, sed displays the lines that contain the word line twice. $ sed '/line/ p' lines Line one. The second line. The second line. The third. This is line four. This is line four. Five. This is the sixth sentence. This is line seven. This is line seven. Eighth and last. The preceding command uses the address /line/, a regular expression that is a simple string. The sed utility selects each of the lines that contains a match for that pattern. The Print (p) instruction displays each of the selected lines. The following command uses the –n option, so sed displays only the selected lines: $ sed -n '/line/ p' lines The second line. This is line four. This is line seven. In the next example, sed displays part of a file based on line numbers. The Print instruction selects and displays lines 3 through 6. $ sed -n '3,6 p' lines The third. This is line four. Five. This is the sixth sentence. The next command line uses the Quit instruction to cause sed to display only the beginning of a file. In this case sed displays the first five lines of lines just as a head –5 lines command would. $ sed '5 q' lines Line one. The second line. The third. This is line four. Five. program-file When you need to give sed more complex or lengthy instructions, you can use a program-file. The print3_6 program performs the same function as the command line in the second preceding example. The –f option tells sed to read its program from the file named following this option. 572 Chapter 13 The sed Editor $ cat print3_6 3,6 p $ sed -n -f print3_6 lines The third. This is line four. Five. This is the sixth sentence. Append The next program selects line 2 and uses an Append instruction to append a NEWLINE and the text AFTER. to the selected line. Because the command line does not include the –n option, sed copies all lines from the input file lines. $ cat append_demo 2 a\ AFTER. $ sed -f append_demo lines Line one. The second line. AFTER. The third. This is line four. Five. This is the sixth sentence. This is line seven. Eighth and last. Insert The insert_demo program selects all lines containing the string This and inserts a NEWLINE and the text BEFORE. before the selected lines. $ cat insert_demo /This/ i\ BEFORE. $ sed -f insert_demo lines Line one. The second line. The third. BEFORE. This is line four. Five. BEFORE. This is the sixth sentence. BEFORE. This is line seven. Eighth and last. Change The next example demonstrates a Change instruction with an address range. When you specify a range of lines for a Change instruction, it does not change each line within the range but rather changes the block of lines to a single occurrence of the new text. Examples 573 $ cat change_demo 2,4 c\ SED WILL INSERT THESE\ THREE LINES IN PLACE\ OF THE SELECTED LINES. $ sed -f change_demo lines Line one. SED WILL INSERT THESE THREE LINES IN PLACE OF THE SELECTED LINES. Five. This is the sixth sentence. This is line seven. Eighth and last. Substitute The next example demonstrates a Substitute instruction. The sed utility selects all lines because the instruction has no address. On each line subs_demo replaces the first occurrence of line with sentence. The p flag displays each line where a substitution occurs. The command line calls sed with the –n option, so sed displays only the lines the program explicitly specifies. $ cat subs_demo s/line/sentence/p $ sed -n -f subs_demo lines The second sentence. This is sentence four. This is sentence seven. The next example is similar to the preceding one except that a w flag and filename (temp) at the end of the Substitute instruction cause sed to create the file named temp. The command line does not include the –n option, so it displays all lines in addition to writing the changed lines to temp. The cat utility displays the contents of the file temp. The word Line (starting with an uppercase L) is not changed. $ cat write_demo1 s/line/sentence/w temp $ sed -f write_demo1 lines Line one. The second sentence. The third. This is sentence four. Five. This is the sixth sentence. This is sentence seven. Eighth and last. $ cat temp The second sentence. This is sentence four. This is sentence seven. 574 Chapter 13 The sed Editor The following bash script changes all occurrences of REPORT to report, FILE to file, and PROCESS to process in a group of files. Because it is a shell script and not a sed program file, you must have read and execute permission to the sub file to execute it as a command (page 278). The for structure (page 412) loops through the list of files on the command line. As it processes each file, the script displays each filename before processing the file with sed. This program uses embedded sed commands that span multiple lines. Because the NEWLINEs between the commands are quoted (they appear between single quotation marks), sed accepts multiple commands on a single, extended command line (within the shell script). Each Substitute instruction includes a g (global) flag to take care of the case where a string occurs more than once on a line. $ cat sub for file do echo $file mv $file $$.subhld sed 's/REPORT/report/g s/FILE/file/g s/PROCESS/process/g' $$.subhld > $file done rm $$.subhld $ sub file1 file2 file3 file1 file2 file3 In the next example, a Write instruction copies part of a file to another file (temp2). The line numbers 2 and 4, separated by a comma, select the range of lines sed is to copy. This program does not alter the lines. $ cat write_demo2 2,4 w temp2 $ sed -n -f write_demo2 lines $ cat temp2 The second line. The third. This is line four. The program write_demo3 is similar to write_demo2 but precedes the Write instruction with the NOT operator (!), causing sed to write to the file those lines not selected by the address. $ cat write_demo3 2,4 !w temp3 $ sed -n -f write_demo3 lines Examples 575 $ cat temp3 Line one. Five. This is the sixth sentence. This is line seven. Eighth and last. Next (n) The following example demonstrates the Next (n) instruction. When it processes the selected line (line 3), sed immediately starts processing the next line without displaying line 3. $ cat next_demo1 3 n p $ sed -n -f next_demo1 lines Line one. The second line. This is line four. Five. This is the sixth sentence. This is line seven. Eighth and last. The next example uses a textual address. The sixth line contains the string the, so the Next (n) instruction causes sed not to display it. $ cat next_demo2 /the/ n p $ sed -n -f next_demo2 lines Line one. The second line. The third. This is line four. Five. This is line seven. Eighth and last. Next (N) The following example is similar to the preceding example except it uses the uppercase Next (N) instruction in place of the lowercase Next (n) instruction. Here the Next (N) instruction appends the next line to the line that contains the string the. In the lines file, sed appends line 7 to line 6 and embeds a NEWLINE between the two lines. The Substitute command replaces the embedded NEWLINE with a SPACE. The Substitute command does not affect other lines because they do not contain embedded NEWLINEs; rather, they are terminated by NEWLINEs. See page 926 for an example of the Next (N) instruction in a sed script running under OS X. $ cat Next_demo3 /the/ N s// / p 576 Chapter 13 The sed Editor $ sed -n -f Next_demo3 lines Line one. The second line. The third. This is line four. Five. This is the sixth sentence. This is line seven. Eighth and last. The next set of examples uses the compound.in file to demonstrate how sed instructions work together. $ cat compound.in 1. The words on this 2. The words on this 3. The words on this 4. The words on this page... page... page... page... The following example substitutes the string words with text on lines 1, 2, and 3 and the string text with TEXT on lines 2, 3, and 4. The example also selects and deletes line 3. The result is text on line 1, TEXT on line 2, no line 3, and words on line 4. The sed utility makes two substitutions on lines 2 and 3: text for words and TEXT for text. Then sed deletes line 3. $ cat compound 1,3 s/words/text/ 2,4 s/text/TEXT/ 3 d $ sed -f compound compound.in 1. The text on this page... 2. The TEXT on this page... 4. The words on this page... The ordering of instructions within a sed program is critical. Both Substitute instructions are applied to the second line in the following example, as in the previous example, but the order in which the substitutions occur changes the result. $ cat compound2 2,4 s/text/TEXT/ 1,3 s/words/text/ 3 d $ sed -f compound2 compound.in 1. The text on this page... 2. The text on this page... 4. The words on this page... In the next example, compound3 appends two lines to line 2. The sed utility displays all lines from the file once because no –n option appears on the command line. The Print instruction at the end of the program file displays line 3 an additional time. Examples 577 $ cat compound3 2 a\ This is line 2a.\ This is line 2b. 3 p $ sed -f compound3 compound.in 1. The words on this page... 2. The words on this page... This is line 2a. This is line 2b. 3. The words on this page... 3. The words on this page... 4. The words on this page... The next example shows that sed always displays appended text. Here line 2 is deleted but the Append instruction still displays the two lines that were appended to it. Appended lines are displayed even if you use the –n option on the command line. $ cat compound4 2 a\ This is line 2a.\ This is line 2b. 2 d $ sed -f compound4 compound.in 1. The words on this page... This is line 2a. This is line 2b. 3. The words on this page... 4. The words on this page... The next example uses a regular expression as the pattern. The regular expression in the following instruction (^.) matches one character at the beginning of every line that is not empty. The replacement string (between the second and third slashes) contains a backslash escape sequence that represents a TAB character (\t) followed by an ampersand (&). The ampersand takes on the value of what the regular expression matched. $ sed 's/^./\t&/' lines Line one. The second line. The third. ... This type of substitution is useful for indenting a file to create a left margin. See Appendix A for more information on regular expressions. You can also use the simpler form s/^/\t/ to add TABs to the beginnings of lines. In addition to placing TABs at the beginnings of lines with text on them, this instruction places a TAB at the beginning of every empty line—something the preceding command does not do. 578 Chapter 13 The sed Editor You may want to put the preceding sed instruction into a shell script so you do not have to remember it (and retype it) each time you want to indent a file. The chmod utility gives you read and execute permission to the ind file. $ cat ind sed 's/^./\t&/' $* $ chmod u+rx ind $ ind lines Line one. The second line. The third. ... Stand-alone script When you run the preceding shell script, it creates two processes: It calls a shell, which in turn calls sed. You can eliminate the overhead associated with the shell process by putting the line #!/bin/sed –f (page 280) at the beginning of the script, which runs the sed utility directly. You need read and execute permission to the file holding the script. $ cat ind2 #!/bin/sed -f s/^./\t&/ In the following sed program, the regular expression (two SPACEs followed by *$) matches one or more SPACEs at the end of a line. This program removes trailing SPACEs at the ends of lines, which is useful for cleaning up files you created using vim. $ cat cleanup sed 's/ *$//' $* The cleanup2 script runs the same sed command as cleanup but stands alone: It calls sed directly with no intermediate shell. $ cat cleanup2 #!/bin/sed -f s/ *$// Hold space The next sed program makes use of the Hold space to exchange pairs of lines in a file. $ h n p g p cat s1 # Copy Pattern space (line just read) to Hold space. # Read the next line of input into Pattern space. # Output Pattern space. # Copy Hold space to Pattern space. # Output Pattern space (which now holds the previous line). $ sed -nf s1 lines The second line. Line one. This is line four. Examples 579 The third. This is the sixth sentence. Five. Eighth and last. This is line seven. The commands in the s1 program process pairs of input lines. This program reads a line and stores it; reads another line and displays it; and then retrieves the stored line and displays it. After processing a pair of lines, the program starts over with the next pair of lines. The next sed program adds a blank line after each line in the input file (i.e., it doublespaces a file). $ sed 'G' lines Line one. The second line. The third. This is line four. $ The G instruction appends a NEWLINE and the contents of the Hold space to the Pattern space. Unless you put something in the Hold space, it will be empty. Thus the G instruction appends a NEWLINE to each line of input before sed displays the line(s) from the Pattern space. The s2 sed program reverses the order of the lines in a file just as the tac utility does. $ cat s2 2,$G # On all but the first line, append a NEWLINE and the # contents of the Hold space to the Pattern space. h # Copy the Pattern space to the Hold space. $!d # Delete all except the last line. $ sed -f s2 lines Eighth and last. This is line seven. This is the sixth sentence. Five. This is line four. The third. The second line. Line one. This program comprises three commands: 2,$G, h, and $!d. To understand this script it is important to understand how the address of the last command works: The $ is the address of the last line of input and the ! negates the address. The result 580 Chapter 13 The sed Editor is an address that selects all lines except the last line of input. In the same fashion you could replace the first command with 1!G: It would select all lines except the first line for processing; the results would be the same. Here is what happens as s2 processes the lines file: 1. The sed utility reads the first line of input (Line one.) into the Pattern space. a. The 2,$G does not process the first line of input—because of its address the G instruction starts processing at the second line. b. The h copies Line one. from the Pattern space to the Hold space. c. The $!d deletes the contents of the Pattern space. Because there is nothing in the Pattern space, sed does not display anything. 2. The sed utility reads the second line of input (The second line.) into the Pattern space. a. The 2,$G adds what is in the Hold space (Line one.) to the Pattern space. The Pattern space now contains The second line.NEWLINELine one. b. The h copies what is in the Pattern space to the Hold space. c. The $!d deletes the second line of input. Because it is deleted, sed does not display it. 3. The sed utility reads the third line of input (The third.) into the Pattern space. a. The 2,$G adds what is in the Hold space (The second line.NEWLINELine one.) to the Pattern space. The Pattern space now has The third.NEWLINE The second line.NEWLINELine one. b. The h copies what is in the Pattern space to the Hold space. c. The $!d deletes the contents of the Pattern space. Because there is nothing in the Pattern space, sed does not display anything. ... 8. The sed utility reads the eighth (last) line of input into the Pattern space. a. The 2,$G adds what is in the Hold space to the Pattern space. The Pattern space now contains all the lines from lines in reverse order. b. The h copies what is in the Pattern space to the Hold space. This step is not necessary for the last line of input but does not alter the program's output. c. The $!d does not process the last line of input. Because of its address the d instruction does not delete the last line. d. The sed utility displays the contents of the Pattern space. Exercises 581 Chapter Summary The sed (stream editor) utility is a batch (noninteractive) editor. It takes its input from files you specify on the command line or from standard input. Unless you redirect the output from sed, it goes to standard output. A sed program consists of one or more lines with the following syntax: [address[,address]] instruction [argument-list] The addresses are optional. If you omit the address, sed processes all lines of input. The instruction is the editing instruction that modifies the text. The addresses select the line(s) the instruction part of the command operates on. The number and kinds of arguments in the argument-list depend on the instruction. In addition to basic instructions, sed includes some powerful advanced instructions. One set of these instructions allows sed programs to store data temporarily in a buffer called the Hold space. Other instructions provide unconditional and conditional branching in sed programs. Exercises 1. Write a sed command that copies a file to standard output, removing all lines that begin with the word Today. 2. Write a sed command that copies only those lines of a file that begin with the word Today to standard output. 3. Write a sed command that copies a file to standard output, removing all blank lines (i.e., lines with no characters on them). 4. Write a sed program named ins that copies a file to standard output, changing all occurrences of cat to dog and preceding each modified line with a line that says following line is modified: 5. Write a sed program named div that copies a file to standard output, copies the first five lines to a file named first, and copies the rest of the file to a file named last. 6. Write a sed command that copies a file to standard output, replacing a single SPACE as the first character on a line with a 0 (zero) only if the SPACE is immediately followed by a number (0–9). For example: abc abc 85c 55b 000 →abc → abc →085c →55b →0000 7. How can you use sed to triple-space (i.e., add two blank lines after each line in) a file? This page intentionally left blank 14 The rsync Secure Copy Utility In This Chapter

The rsync (remote synchronization) utility copies an ordinary file or directory hierarchy locally or from the local system to or from another system on a network. By default, this utility uses OpenSSH to transfer files and the same authentication mechanism as OpenSSH; therefore it provides the same security as OpenSSH. The rsync utility prompts for a password when it needs one. Alternatively, you can use the rsyncd daemon as a transfer agent. 14Chapter14

583 584 Chapter 14 The rsync Secure Copy Utility Syntax An rsync command line has the following syntax: rsync [options] [[user@]from-host:]source-file [[user@]to-host:][destination-file] The rsync utility copies files, including directory hierarchies, on the local system or between the local system and a remote system. Arguments The from-host is the name of the system you are copying files from; the to-host is the name of the system you are copying files to. When you do not specify a host, rsync assumes the local system. The user on either system defaults to the user who is giving the command on the local system; you can specify a different user with user@. Unlike scp, rsync does not permit copying between remote systems. The source-file is the ordinary or directory file you are copying; the destination-file is the resulting copy. You can specify files as relative or absolute pathnames. On the local system, relative pathnames are relative to the working directory; on a remote system, relative pathnames are relative to the specified or implicit user's home directory. When the source-file is a directory, you must use the ––recursive or ––archive option to copy its contents. When the destination-file is a directory, each of the source files maintains its simple filename. If the source-file is a single file, you can omit destination-file; the copied file will have the same simple filename as source-file (useful only when copying to or from a remote machine). A trailing slash (/) on source-file is critical caution When source-file is a directory, a trailing slash in source-file causes rsync to copy the contents of the directory. The slash is equivalent to /*; it tells rsync to ignore the directory itself and copy the files within the directory. Without a trailing slash, rsync copies the directory. See page 587. Options The Mac OS X version of rsync accepts long options tip Options for rsync preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––acls –A Preserves ACLs (page 99) of copied files. ––archive –a Copies files including dereferenced symbolic links, device files, and special files recursively, preserving ownership, group, permissions, and modification times Options 585 associated with the files. Using this option is the same as specifying the ––devices, ––specials, ––group, ––links, ––owner, ––perms, ––recursive, and ––times options. This option does not include the ––acls, ––hard-links, or ––xattrs options; you must specify these options in addition to ––archive if you want to use them. See page 588 for an example. ––backup –b Renames files that otherwise would be deleted or overwritten. By default, the rsync utility renames files by appending a tilde (~) to existing filenames. See ––backup-dir=dir if you want rsync to put these files in a specified directory instead of renaming them. See also ––link-dest=dir. When used with the ––backup option, moves files that otherwise would be deleted or overwritten into the directory named dir. After moving the older version of the file in dir, rsync copies the newer version of the file from source-file to destination-file. ––backup-dir=dir The directory named dir is located on the same system as destination-file. If dir is a relative pathname, it is relative to destination-file. ––copy-unsafe-links (partial dereference) For each file that is a symbolic link that refers to a file outside the source-file hierarchy, copies the file the link points to, not the symbolic link itself. Without this option rsync copies all symbolic links, even if it does not copy the file the link refers to. –D same as ––devices ––specials. ––delete Deletes files in the destination-file that are not in the source-file. This option can easily remove files you did not intend to remove; see the caution box on page 589. ––devices Copies device files (root user only). ––dry-run Runs rsync without writing to disk. With the ––verbose option, this option reports on what rsync would have done had it been run without this option. Useful with the ––delete option. ––group –g Preserves group associations of copied files. ––hard-links –H Preserves hard links of copied files. ––links –l (lowercase "l"; no dereference) For each file that is a symbolic link, copies the symbolic link, not the file the link points to, even if the file the link points to is not in the source-file. ––link-dest=dir If rsync would normally copy a file—that is, if the file exists in source-file but not in destination-file or is changed in destination-file—rsync looks in the directory named dir for the same file. If it finds an exact copy of the file in dir, rsync makes a hard link from the file in dir to destination-file. If it does not 586 Chapter 14 The rsync Secure Copy Utility find an exact copy, rsync copies the file to destination-file. See page 592 for an example. The directory named dir is located on the same system as destination-file. If dir is a relative pathname, it is relative to destination-file. ––owner –o Preserves the owner of copied files (root user only). ––perms –p Preserves the permissions of copied files. ––recursive ––specials ––times ––update ––verbose –r Recursively descends a directory specified in source-file and copies all files in the directory hierarchy. See page 587 for an example. Copies special files. –t Preserves the modification times of copied files. This option also speeds up copying files because it causes rsync not to copy a file that has the same modification time and size in both the source-file and the destination-file. See page 587 for an example. –u Skips files that are newer in the destination-file than in the source-file. –v Displays information about what rsync is doing. This option is useful with the ––dry-run option. See page 587 for an example. ––xattrs –X Preserves the extended attributes of copied files. This option is not available with all compilations of rsync. ––compress –z Compresses files while copying them. Notes The rsync utility has many options. This chapter describes a few of them; see the rsync man page for a complete list. OpenSSH By default, rsync copies files to and from a remote system using OpenSSH. The remote system must be running an OpenSSH server. If you can use ssh to log in on the remote system, you can use rsync to copy files to or from that system. If ssh requires you to enter a password, rsync will require a password. See "Copying Files to and from a Remote System" on page 590 for examples. See "Discussion" on page 829 for more information on setting up and using OpenSSH. rsyncd daemon If you use a double colon (::) in place of a single colon (:) following the name of a remote system, rsync connects to the rsyncd daemon on the remote system (it does not use OpenSSH). See the rsync man page for more information. More Information man page: rsync rsync home page: www.samba.org/rsync Backup information: www.mikerubel.org/computers/rsync_snapshots Examples 587 Backup tools: www.rsnapshot.org, backuppc.sourceforge.net File synchronization: alliance.seas.upenn.edu/~bcpierce Examples ––recursive ––verbose The first example shows rsync making a copy of a directory using the ––recursive and ––verbose options. Both the source and destination directories are in the working directory. $ ls -l memos total 12 -rw-r--r-- 1 max max 1500 Jun -rw-r--r-- 1 max max 6001 Jun 6 14:24 0606 8 16:16 0608 $ rsync --recursive --verbose memos memos.copy building file list ... done created directory memos.copy memos/ memos/0606 memos/0608 sent 7671 bytes received 70 bytes 15482.00 bytes/sec total size is 7501 speedup is 0.97 $ ls -l memos.copy total 4 drwxr-xr-x 2 max max 4096 Jul 20 17:42 memos In the preceding example, rsync copies the memos directory to the memos.copy directory. As the following ls command shows, rsync changed the modification times on the copied files to the time it made the copies: $ ls -l memos.copy/memos total 12 -rw-r--r-- 1 max max 1500 Jul 20 17:42 0606 -rw-r--r-- 1 max max 6001 Jul 20 17:42 0608 Using a Trailing Slash (/) on source-file Whereas the previous example copied a directory to another directory, you may want to copy the contents of a directory to another directory. A trailing slash (/) on the source-file causes rsync to act as though you had specified a trailing /* and causes rsync to copy the contents of the specified directory. A trailing slash on the destination-file has no effect. ––times The next example makes another copy of the memos directory, using ––times to preserve modification times of the copied files. It uses a trailing slash on memos 588 Chapter 14 The rsync Secure Copy Utility to copy the contents of the memos directory—not the directory itself—to memos.copy2. $ rsync --recursive --verbose --times memos/ memos.copy2 building file list ... done created directory memos.copy2 ./ 0606 0608 sent 7665 bytes received 70 bytes 15470.00 bytes/sec total size is 7501 speedup is 0.97 $ $ ls -l memos.copy2 total 12 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 ––archive The ––archive option causes rsync to copy directories recursively, dereferencing symbolic links (copying the files the links point to, not the symbolic links themselves), preserving modification times, ownership, group association of the copied files, and more. This option does not preserve hard links; use the ––hard-links option for that purpose. See page 584 for more information on ––archive. The following commands perform the same functions as the previous one: $ rsync --archive --verbose memos/ memos.copy2 $ rsync -av memos/ memos.copy2 Removing Files ––delete ––dry-run The ––delete option causes rsync to delete from destination-file files that are not in source-file. Together, the ––dry-run and ––verbose options report on what an rsync command would do without the ––dry-run option, without rsync taking any action. With the ––delete option, the ––dry-run and ––verbose options can help you avoid removing files you did not intend to remove. This combination of options marks files rsync would remove with the word deleting. The next example uses these options in addition to the ––archive option. $ ls -l memos memos.copy3 memos: total 20 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 5911 Jun 10 12:02 0610 memos.copy3: total 16 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 1200 Jul 21 10:17 notes Examples 589 $ rsync --archive --verbose --delete --dry-run memos/ memos.copy3 building file list ... done deleting notes ./ 0610 sent 125 bytes received 32 bytes 314.00 bytes/sec total size is 13412 speedup is 85.43 The rsync utility reports deleting notes, indicating which file it would remove if you ran it without the ––dry-run option. It also reports it would copy the 0610 file. Test to make sure ––delete is going to do what you think it will do caution The ––delete option can easily delete an entire directory tree if you omit a needed slash (/) or include an unneeded slash in source-file. Use ––delete with the ––dry-run and ––verbose options to test an rsync command. If you get tired of using the long versions of options, you can use the single-letter versions. The next rsync command is the same as the previous one (there is no short version of the ––delete option): $ rsync -avn --delete memos/ memos.copy3 The next example runs the same rsync command, omitting the ––dry-run option. The ls command shows the results of the rsync command: The ––delete option causes rsync to remove the notes file from the destination-file (memos.copy3) because it is not in the source-file (memos). In addition, rsync copies the 0610 file. $ rsync --archive --verbose --delete memos/ memos.copy3 building file list ... done deleting notes ./ 0610 sent 6076 bytes received 48 bytes 12248.00 bytes/sec total size is 13412 speedup is 2.19 $ ls -l memos memos.copy3 memos: total 20 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 5911 Jun 10 12:02 0610 memos.copy3: total 20 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 5911 Jun 10 12:02 0610 590 Chapter 14 The rsync Secure Copy Utility To this point, the examples have copied files locally, in the working directory. To copy files to other directories, replace the simple filenames with relative or absolute pathnames. On the local system, relative pathnames are relative to the working directory; on a remote system, relative pathnames are relative to the user's home directory. For example, the following command copies memos from the working directory to the /backup directory on the local system: $ rsync --archive --verbose --delete memos/ /backup Copying Files to and from a Remote System To copy files to or from a remote system, that system must be running an OpenSSH server or another transport mechanism rsync can connect to. For more information refer to "Notes" on page 586. To specify a file on a remote system, preface the filename with the name of the remote system and a colon. Relative pathnames on the remote system are relative to the user's home directory. Absolute pathnames are absolute (i.e., they are relative to the root directory). See page 83 for more information on relative and absolute pathnames. In the next example, Max copies the memos directory from the working directory on the local system to the holdfiles directory in his working directory on the remote system named coffee. The ssh utility runs an ls command on coffee to show the result of the rsync command. The rsync and ssh utilities do not request a password because Max has set up OpenSSH-based utilities to log in automatically on coffee (page 830). $ rsync --archive memos/ coffee:holdfiles $ ssh coffee 'ls -l holdfiles' total 20 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 5911 Jun 10 12:02 0610 When copying from a remote system to the local system, place the name of the remote system before source-file: $ rsync --archive coffee:holdfiles/ ~/memo.copy4 $ rsync --archive coffee:holdfiles/ /home/max/memo.copy5 Both of these commands copy the holdfiles directory from Max's working directory on coffee to his home directory on the local system. Under Mac OS X, replace /home with /Users. Mirroring a Directory You can use rsync to maintain a copy of a directory. Because it is an exact copy, this type of copy is called a mirror. The mirror directory must be on an OpenSSH server (you must be able to connect to it using an OpenSSH utility such as ssh). If you want to run this script using crontab (page 649), you must set up OpenSSH so Examples 591 you can log in on the remote system automatically (without providing a password; page 830). ––compress ––update The next example introduces the rsync ––compress and ––update options. The ––compress option causes rsync to compress files as it copies them, usually making the transfer go more quickly. In some cases, such as a setup with a fast network connection and a slower CPU, compressing files can slow down the transfer. The ––update option keeps rsync from overwriting a newer file with an older one. As with all shell scripts, you must have read and execute access to the mirror script. To make it easier to read, each option in this script appears on a line by itself. Each line of each command except the last is terminated with a SPACE and a backslash (\). The SPACE separates one option from the next; the backslash quotes the following NEWLINE so the shell passes all arguments to rsync and does not interpret the NEWLINEs as the end of the command. $ cat mirror rsync \ --archive \ --verbose \ --compress \ --update \ --delete \ ~/mirrordir/ coffee:mirrordir $ ./mirror > mirror.out The mirror command in the example redirects output to mirror.out for review. Remove the ––verbose option if you do not want the command to produce any output except for errors. The rsync command in mirror copies the mirrordir directory hierarchy from the user's home directory on the local system to the user's home directory on the remote (server) system. In this example the remote system is named coffee. Because of the ––update option, rsync will not overwrite newer versions of files on the server with older versions of the same files from the local system. Although this option is not required if files on the server system are never changed manually, it can save you from grief if you accidentally update files on or add files to the server system. The ––delete option causes rsync to remove files on the server system that are not present on the local system. Making Backups After performing an initial full backup, rsync is able to perform subsequent incremental backups efficiently with regard to running time and storage space. By definition, an incremental backup stores only those files that have changed since the last backup; these are the only files that rsync needs to copy. As the following example shows, rsync, without using extra disk space, can make each incremental backup appear to be a full backup by creating hard links between the incremental backup and the unchanged files in the initial full backup. 592 Chapter 14 The rsync Secure Copy Utility ––link-dest=dir The rsync ––link-dest=dir option makes backups easy and efficient. It presents the user and/or system administrator with snapshots that appear to be full backups while taking minimal extra space in addition to the initial backup. The dir directory is always located on the machine holding the destination-file. If dir is a relative pathname, the pathname is relative to the destination-file. See page 585 for a description of this option. Following is a simple rsync command that uses the ––link-dest=dir option: $ rsync --archive --link-dest=../backup source/ destination When you run this command, rsync descends the source directory, examining each file it finds. For each file in the source directory, rsync looks in the destination directory to find an exact copy of the file. • If it finds an exact copy of the file in the destination directory, rsync continues with the next file. • If it does not find an exact copy of the file in the destination directory, rsync looks in the backup directory to find an exact copy of the file. • If it finds an exact copy of the file in the backup directory, rsync makes a hard link from the file in the backup directory to the destination directory. • If it does not find an exact copy of the file in the backup directory, rsync copies the file from the source directory to the destination directory. Next is a simple example showing how to use rsync to make full and incremental backups using the ––link-dest=dir option. Although the backup files reside on the local system, they could easily be located on a remote system. As specified by the two arguments to rsync in the bkup script, rsync copies the memos directory to the bu.0 directory. The ––link-dest=dir option causes rsync to check whether each file it needs to copy exists in bu.1. If it does, rsync creates a link to the bu.1 file instead of copying it. The bkup script rotates three backup directories named bu.0, bu.1, and bu.2 and calls rsync. The script removes bu.2, moves bu.1 to bu.2, and moves bu.0 to bu.1. The first time you run the script, rsync copies all the files from memos because they do not exist in bu.0 or bu.1. $ cat bkup rm -rf bu.2 mv bu.1 bu.2 mv bu.0 bu.1 rsync --archive --link-dest=../bu.1 memos/ bu.0 Before you run bkup for the first time, bu.0, bu.1, and bu.2 do not exist. Because of the –f option, rm does not display an error message when it tries to remove the Examples 593 nonexistent bu.2 directory. Until bkup creates bu.0 and bu.1, mv displays error messages saying there is No such file or directory. In the following example, ls shows the bkup script and the contents of the memos directory. After running bkup, ls shows the contents of memos and of the new bu.0 directory; bu.0 holds exact copies of the files in memos. The rsync utility created no links because there were no files in bu.1: The directory did not exist. $ ls -l * -rwxr-xr-x 1 max max 87 Jul 22 11:23 bkup memos: total 20 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 5911 Jun 10 12:02 0610 $ ./bkup mv: cannot stat 'bu.1': No such file or directory mv: cannot stat 'bu.0': No such file or directory $ ls -l * -rwxr-xr-x 1 max max 87 Jul 22 11:23 bkup bu.0: total 20 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 5911 Jun 10 12:02 0610 memos: total 20 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 5911 Jun 10 12:02 0610 After working with the files in memos, ls shows 0610 has been removed and newfile has been added: $ ls -l memos total 20 -rw-r--r-- 1 max max 2100 Jul 22 14:31 0606 -rw-r--r-- 1 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 5251 Jul 22 14:32 newfile After running bkup again, bu.0 holds the same files as memos and bu.1 holds the files that bu.0 held before running bkup. The 0608 file has not changed, so rsync, with the ––link-dest=dir option, has not copied it but rather has made a link from the copy in bu.1 to the copy in bu.0, as indicated by the 2 ls displays between the permissions and max. 594 Chapter 14 The rsync Secure Copy Utility $ ./bkup mv: cannot stat 'bu.1': No such file or directory $ ls -l bu.0 bu.0: total 20 -rw-r--r-- 1 -rw-r--r-- 2 -rw-r--r-- 1 bu.1 max max 2100 Jul 22 14:31 0606 max max 6001 Jun 8 16:16 0608 max max 5251 Jul 22 14:32 newfile bu.1: total 20 -rw-r--r-- 1 max max 1500 Jun 6 14:24 0606 -rw-r--r-- 2 max max 6001 Jun 8 16:16 0608 -rw-r--r-- 1 max max 5911 Jun 10 12:02 0610 The beauty of this setup is that each incremental backup occupies only the space needed to hold files that have changed. Files that have not changed are stored as links, which take up minimal disk space. Yet users and the system administrator have access to a directory that appears to hold a full backup. You can run a backup script such as bkup once an hour, once a day, or as often as you like. Storage space permitting, you can have as many backup directories as you like. If rsync does not require a password, you can automate this process by using crontab (page 649). Chapter Summary The rsync utility copies an ordinary file or directory hierarchy locally or between the local system and a remote system on a network. By default, this utility uses openSSH to transfer files and the same authentication mechanism as openSSH; therefore it provides the same security as openSSH. The rsync utility prompts for a password when it needs one. Exercises 1. List three features of rsync. 2. Write an rsync command that copies the backmeup directory from your home directory on the local system to the /tmp directory on coffee, preserving file ownership, permissions, and modification times. Write a command that will copy the same directory to your home directory on coffee. Do not assume your working directory on the local system is your home directory. Exercises 595 3. You are writing an rsync command that includes the ––delete option. Which options would you use to test the command without copying or removing any files? 4. What does the ––archive option do? Why is it useful? 5. When running a script such as bkup (page 592) to back up files on a remote system, how could you rotate (rename) files on a remote system? 6. What effect does a trailing slash (/) on the source-file have? This page intentionally left blank I PART V Command Reference 597 This page intentionally left blank Command Reference 599 V V Command Reference Command Reference The following tables list the utilities covered in this part of the book grouped by function and alphabetically within function. Although most of these are true utilities (programs that are separate from the shells), some are built into the shells (shell builtins). The sample utility on page 605 shows the format of the description of each utility in this part of the book. Utilities That Display and Manipulate Files

 Network Utilities

a network—page 852 Command Reference Utilities That Display and Alter Status cd Changes to another working directory—page 620 chgrp Changes the group associated with a file—page 622 chmod Changes the access mode (permissions) of a file—page 626 chown Changes the owner of a file and/or the group the file is associated with—page 631 date Displays or sets the system time and date—page 655 df Displays disk space usage—page 661 dscl Displays and manages Directory Service information—page 674 O dmesg Displays kernel messages—page 673 du Displays information on disk usage by directory hierarchy and/or file—page 677 file Displays the classification of a file—page 686 finger Displays information about users—page 695 GetFileInfo Displays file attributes—page 717 O kill Terminates a process by PID—page 729 killall Terminates a process by name—page 731 nice Changes the priority of a command—page 773 nohup Runs a command that keeps running after you log out—page 775 ps Displays process status—page 796 renice Changes the priority of a process—page 802 SetFile Sets file attributes—page 813 O sleep Creates a process that sleeps for a specified interval—page 815 stat Displays information about files—page 835 stty Displays or sets terminal parameters—page 838 sysctl Displays and alters kernel variables—page 842 O top Dynamically displays process status—page 858 umask Establishes the file-creation permissions mask—page 870 w Displays information about system users—page 874 which Shows where in PATH a command is located—page 877 who Displays information about logged-in users—page 879 601 602 Command Reference Utilities That Are Programming Tools awk Searches for and processes patterns in a file—page 531 configure Configures source code automatically—page 638 gawk Searches for and processes patterns in a file—page 531 gcc Compiles C and C++ programs—page 712 make Keeps a set of programs current—page 753 mawk Searches for and processes patterns in a file—page 531 Miscellaneous Utilities at Executes commands at a specified time—page 611 cal Displays a calendar—page 617 crontab Maintains crontab files—page 649 diskutil Checks, modifies, and repairs local volumes—page 668 O echo Displays a message—page 680 expr Evaluates an expression—page 682 fsck Checks and repairs a filesystem—page 699 launchctl Controls the launchd daemon—page 733 O mkfs Creates a filesystem on a device—page 764 Mtools Uses DOS-style commands on files and directories—page 767 tee Copies standard input to standard output and one or more files—page 851 test Evaluates an expression—page 854 tr Replaces specified characters—page 864 tty Displays the terminal pathname—page 867 tune2fs Changes parameters on an ext2, ext3, or ext4 filesystem—page 868 xargs Converts standard input to command lines—page 881 Standard Multiplicative Suffixes Several utilities allow you to use the suffixes listed in Table V-1 following byte counts. You can precede a multiplicative suffix with a number that is a multiplier. For example, 5K means 5 × 210. The absence of a multiplier indicates that the Command Reference 603 multiplicative suffix is to be multiplied by 1. The utilities that allow these suffixes are marked as such. BLOCKSIZE Table V-1 Multiplicative suffixes Suffix Multiplicative value Suffix Multiplicative value KB 1,000 (103) PB 1015 K 1,024 (210) P 250 MB 1,000,000 (106) EB 1018 M 1,048,576 (220) E 260 GB 1,000,000,000 (109) ZB 1021 G 1,073,741,824 (230) Z 270 TB 1012 YB 1024 T 240 Y 280 Under Mac OS X, some utilities use the BLOCKSIZE environment variable to set a default block size. You can set BLOCKSIZE to a value that is a number of bytes or to a value that uses one of the K, M, or G suffixes. The text identifies utilities that use BLOCKSIZE. Common Options Several GNU utilities share the options listed in Table V-2. The utilities that use these options are marked as such. Table V-2 Common command-line options Option Effect – A single hyphen appearing in place of a filename instructs the utility to accept standard input in place of the file. –– A double hyphen marks the end of the options on a command line. You can follow this option with an argument that begins with a hyphen. Without this option the utility assumes that an argument that begins with a hyphen is an option. ––help Displays a help message for the utility. Some of these messages are quite long; using a pipe, you can send the output through less to display it one screen at a time. For example, you could give the command ls ––help | less. Alternatively, you can send the output through a pipe to grep if you are looking for specific information. For example, you could give the following command to get information on the –d option to ls: ls ––help | grep –– –d. See the preceding entry in this table for information on the double hyphen. ––version Displays version information for the utility. 604 Command Reference The sample Utility The following description of the sample utility shows the format that this part of the book uses to describe the utilities. These descriptions are similar to the man page descriptions (pages 33 and 759); however, most users find the descriptions in this book easier to read and understand. They emphasize the most useful features of the utilities and often leave out the more obscure features. For information about the less commonly used features, refer to the man and info pages or call the utility with the ––help option, which works with many utilities. sample O 605 sample O ←OS X in an oval indicates this utility runs under Mac OS X only. Brief description of what the utility does sample [options] arguments Following the syntax line is a description of the utility. The syntax line shows how to run the utility from the command line. Options and arguments enclosed in brackets ([]) are not required. Enter words that appear in this italic typeface as is. Words that you must substitute when you type appear in this bold italic typeface. Words listed as arguments to a command identify single arguments (for example, source-file) or groups of similar arguments (for example, directory-list). Arguments This section describes the arguments you can use when you run the utility. The argument itself, as shown in the preceding syntax line, is printed in this bold italic typeface. Options This section lists some of the options you can use with the command. Unless otherwise specified, you must precede options with one or two hyphens. Most commands accept a single hyphen before multiple options (page 119). Options in this section are ordered alphabetically by short (single-hyphen) options. If an option has only a long version (two hyphens), it is ordered by its long option. Following are some sample options: ––delimiter=dchar –d dchar This option includes an argument. The argument is set in a bold italic typeface in both the heading and the description. You substitute another word (filename, string of characters, or other value) for any arguments you see in this typeface. Type characters that are in bold type (such as the ––delimiter and –d) as is. ––make-dirs –m This option has a long and a short version. You can use either option; they are equivalent. This option description ends with Linux in a box, indicating it is available under Linux only. Options not followed by Linux or OS X are available under both operating systems. L –t (table of contents) This simple option is preceded by a single hyphen and not followed by arguments. It has no long version. The table of contents appearing in parentheses at the beginning of the description is a cue, suggestive of what the option letter stands for. This option description ends with OS X in a box, indicating it is available under OS X only. Options not followed by Linux or OS X are available under both operating systems. O sample O sample O 606 sample O Discussion This optional section describes how to use the utility and identifies any quirks it may have. Notes This section contains miscellaneous notes—some important and others merely interesting. Examples This section contains examples illustrating how to use the utility. This section is a tutorial, so it takes a more casual tone than the preceding sections of the description. aspell 607 Checks a file for spelling errors aspell check [options] filename aspell list [options] < filename aspell config aspell help The aspell utility checks the spelling of words in a document against a standard dictionary. You can use aspell interactively: It displays each misspelled word in context, together with a menu that gives you the choice of accepting the word as is, choosing one of aspell's suggested replacements for the word, inserting the word into your personal dictionary, or replacing the word with one you enter. You can also use aspell in batch mode so it reads from standard input and writes to standard output. The aspell utility is available under Linux only. L aspell is not like other utilities regarding its input tip Unlike many other utilities, aspell does not accept input from standard input when you do not specify a filename on the command line. Instead, the action specifies where aspell gets its input. Actions You must choose one and only one action when you run aspell. check –c Runs aspell as an interactive spelling checker. Input comes from a single file named on the command line. Refer to "Discussion" on page 608. config Displays aspell's configuration, both default and current values. Send the output through a pipe to less for easier viewing, or use grep to find the option you are looking for (for example, aspell config | grep backup). help –? Displays an extensive page of help. Send the output through a pipe to less for easier viewing. list –l Runs aspell in batch mode (noninteractively) with input coming from standard input and output going to standard output. Arguments The filename is the name of the file you want to check. The aspell utility accepts this argument only when you use the check (–c) action. With the list (–l) action, input must come from standard input. Options The aspell utility has many options. A few of the more commonly used ones are listed in this section; see the aspell man page for a complete list. Default values of many options are determined when aspell is compiled (see the config action). You can specify options on the command line, as the value of the ASPELL_CONF shell variable, or in your personal configuration file (~/.aspell.conf). A user working aspell aspell 608 aspell with root privileges can create a global configuration file (/etc/aspell.conf). Put one option per line in a configuration file; separate options with a semicolon (;) in ASPELL_CONF. Options on the command line override those in ASPELL_CONF, which override those in your personal configuration file, which override those in the global configuration file. The following list contains two types of options: Boolean and value. The Boolean options turn a feature on (enable the feature) or off (disable the feature). Precede a Boolean option with dont– to turn it off. For example, ––ignore-case turns the ignore-case feature on and ––dont-ignore-case turns it off. Value options assign a value to a feature. Follow the option with an equal sign and a value—for example, ––ignore=4. For all options in a configuration file or in the ASPELL_CONF variable, drop the leading hyphens (ignore-case or dont-ignore-case). aspell options and leading hyphens caution The way you specify options differs depending on whether you are specifying them on the command line, using the ASPELL_CONF shell variable, or in a configuration file. On the command line, prefix long options with two hyphens (for example, ––ignore-case or ––dont-ignore-case). In ASPELL_CONF and configuration files, drop the leading hyphens (for example, ignore-case or dont-ignore-case). ––dont-backup ––ignore=n Does not create a backup file named filename.bak (default is ––backup when action is check). Ignores words with n or fewer characters (default is 1). ––ignore-case Ignores the case of letters in words being checked (default is ––dont-ignore-case). ––lang=cc Specifies the two-letter language code (cc). The language code defaults to the value of LC_MESSAGES (page 304). ––mode=mod Specifies a filter to use. Select mod from url (default), none, sgml, and others. The modes work as follows: url: skips URLs, hostnames, and email addresses; none: turns off all filters; sgml: skips SGML, HTML, XHTML, and XML commands. ––strip-accents Removes accent marks from all the words in the dictionary before checking words (default is ––dont-strip-accents). Discussion The aspell utility has two basic modes of operation: batch and interactive. You specify batch mode by using the list or –l action. In batch mode, aspell accepts the document you want to check for spelling errors as standard input and sends the list of potentially misspelled words to standard output. You specify interactive mode by using the check or –c action. In interactive mode, aspell displays a screen with the potentially misspelled word highlighted in context aspell 609 and a menu of choices. See "Examples" for an illustration. The menu includes various commands (Table V-3) as well as some suggestions of similar, correctly spelled words. You either enter one of the numbers from the menu to select a suggested word to replace the word in question or enter a letter to give a command. Table V-3 Notes aspell commands Command Action SPACE Takes no action and goes on to next the misspelled word. n Replaces the misspelled word with suggested word number n. a Adds the "misspelled" word to your personal dictionary. b Aborts aspell; does not save changes. i or I (letter "i") Ignores the misspelled word. I (uppercase "I") ignores all occurrences of this word; i ignores this occurrence only and is the same as SPACE. l (lowercase "l") Changes the "misspelled" word to lowercase and adds it to your personal dictionary. r or R Replaces the misspelled word with the word you enter at the bottom of the screen. R replaces all occurrences of this word; r replaces this occurrence only. x Saves the file as corrected so far and exits from aspell. For more information refer to the aspell home page located at aspell.net and to the /usr/share/doc/aspell directory. The aspell utility is not a foolproof way of finding spelling errors. It also does not check for misused, properly spelled words (such as red instead of read). Spelling from emacs You can make it easy to use aspell from emacs by adding the following line to your ~/.emacs file (page 250). This line causes emacs' ispell functions to call aspell: (setq-default ispell-program-name "aspell") Spelling from vim Similarly, you can make it easy to use aspell from vim by adding the following line to your ~/.vimrc file (page 185): map ^T :w!:!aspell check %:e! % When you enter this line in ~/.vimrc using vim, enter the ^T as CONTROL-V CONTROL-T (page 169). With this line in ~/.vimrc, CONTROL-T brings up aspell to spell check the file you are editing with vim. Examples The following examples use aspell to correct the spelling in the memo.txt file: $ cat memo.txt Here's a document for teh aspell utilitey to check. It obviously needs proofing quiet badly. 610 aspell The first example uses aspell with the check action and no options. The appearance of the screen for the first misspelled word, teh, is shown. At the bottom of the screen is the menu of commands and suggested words. Each of the numbered words differs slightly from the misspelled word: $ aspell check memo.txt Here's a document for teh aspell utilitey to check. It obviously needs proofing quiet badly. == 1) the 6) th 2) Te 7) tea 3) tech 8) tee 4) Th 9) Ted 5) eh 0) tel i) Ignore I) Ignore all r) Replace R) Replace all a) Add l) Add Lower b) Abort x) Exit == ? Enter one of the menu choices in response to the preceding display; aspell will do your bidding and move the highlight to the next misspelled word (unless you choose to abort or exit). In this case, entering 1 (one) would change teh to the in the file. The next example uses the list action to display a list of misspelled words. The word quiet is not in the list—it is not properly used but is properly spelled. $ aspell list < memo.txt teh aspell utilitey obviosly The last example also uses the list action. It shows a quick way to check the spelling of a word or two with a single command. The user gives the aspell list command and then enters seperate temperature into aspell's standard input (the keyboard). After the user presses RETURN and CONTROL-D (to mark the end of file), aspell writes the misspelled word to standard output (the screen): $ aspell list seperate temperatureRETURN CONTROL-D seperate at 611 at at [options] time [date | +increment] atq atrm job-list batch [options] [time] The at and batch utilities execute commands at a specified time. They accept commands from standard input or, with the –f option, from a file. Commands are executed in the same environment as the at or batch command. Unless redirected, standard output and standard error from commands are emailed to the user who ran the at or batch command. A job is the group of commands that is executed by one call to at. The batch utility differs from at in that it schedules jobs so they run when the CPU load on the system is low. The atq utility displays a list of at jobs you have queued; atrm cancels pending at jobs. Arguments The time is the time of day when at runs the job. You can specify the time as a one-, two-, or four-digit number. One- and two-digit numbers specify an hour, and four-digit numbers specify an hour and minute. You can also give the time in the form hh:mm. The at utility assumes a 24-hour clock unless you place am or pm immediately after the number, in which case it uses a 12-hour clock. You can also specify time as now, midnight, noon, or teatime (4:00 PM). The date is the day of the week or day of the month when you want at to execute the job. When you do not specify a day, at executes the job today if the hour you specify in time is greater than the current hour. If the hour is less than the current hour, at executes the job tomorrow. You specify a day of the week by spelling it out or abbreviating it to three letters. You can also use the words today and tomorrow. Use the name of a month followed by the number of the day in the month to specify a date. You can follow the month and day number with a year. The increment is a number followed by one of the following (both plural and singular are allowed): minutes, hours, days, or weeks. The at utility adds the increment to time. You cannot specify an increment for a date. When using atrm, job-list is a list of one or more at job numbers. You can list job numbers by running at with the –l option or by using atq. Options The batch utility accepts options under OS X only. The –c, –d, and –l options are not for use when you initiate a job with at; use these options to determine the status of a job or to cancel a job only. at Executes commands at a specified time 612 at –c job-list (cat) Displays the environment and commands specified by the job numbers in job-list. –d job-list (delete) Cancels jobs that you submitted with at. The job-list is a list of one or more at job numbers to cancel. If you do not remember the job number, use the –l option or run atq to list your jobs and their numbers. Using this option with at has the same effect as running atrm. This option is deprecated under OS X; use the –r option instead. –f file (file) Specifies that commands come from file instead of standard input. This option is useful for long lists of commands or commands that are executed repeatedly. –l (list) Displays a list of your at jobs. Using this option with at has the same effect as running atq. –m (mail) Sends you email after a job is run, even when nothing is sent to standard output or standard error. When a job generates output, at always emails it to you, regardless of this option. –r job-list (remove) Same as the –d option. O Notes The at utility uses /bin/sh to execute commands. Under Linux, this file is typically a link to bash or dash. The shell saves the environment variables and the working directory at the time you submit an at job so they are available when at executes commands. at.allow and at.deny A user running with root privileges can always use at. The Linux /etc/at.allow (OS X uses /var/at/at.allow) and Linux /etc/at.deny (OS X uses /var/at/at.deny) files, which should be readable and writable by root only (mode 600), control which ordinary, local users can use at. When at.deny exists and is empty, all users can use at. When at.deny does not exist, only those users listed in at.allow can use at. Users listed in at.deny cannot use at unless they are also listed in at.allow. Under Linux, jobs you submit using at are run by the atd daemon. This daemon stores jobs in /var/spool/at or /var/spool/cron/atjobs and stores their output in /var/spool/at/spool or /var/spool/cron/atspool. These files should be set to mode 700 and owned by the user named daemon. Under Mac OS X, jobs you submit using at are run by atrun, which is called every 30 seconds by launchd. The atrun utility stores jobs in /var/at/jobs and stores their output in /var/at/spool, both of which should be set to mode 700 and owned by the user named daemon. at 613 Under OS X 10.4 and later, the atrun daemon is disabled by default. Working as a privileged user, you can enable and disable atrun with the following commands: # launchctl load -w /System/Library/LaunchDaemons/com.apple.atrun.plist # launchctl unload -w /System/Library/LaunchDaemons/com.apple.atrun.plist See launchctl (page 733) for more information. Examples You can use any of the following techniques to paginate and print long_file tomorrow at 2:00 AM. The first example executes the command directly from the command line; the last two examples use the pr_tonight file, which contains the necessary command, and execute that command using at. Prompts and output from different versions of at differ. $ at 2am at> pr long_file | lpr at>CONTROL-D job 8 at 2009-08-17 02:00 $ cat pr_tonight #!/bin/bash pr long_file | lpr $ at -f pr_tonight 2am job 9 at 2009-08-17 02:00 $ at 2am < pr_tonight job 10 at 2009-08-17 02:00 If you execute commands directly from the command line, you must signal the end of the commands by pressing CONTROL-D at the beginning of a line. After you press CONTROL-D, at displays a line that begins with job followed by the job number and the time at will execute the job. If you run atq after the preceding commands, it displays a list of jobs in its queue: $ atq 8 9 10 2009-08-17 02:00 a 2009-08-17 02:00 a 2009-08-17 02:00 a The following command removes job number 9 from the queue: $ atrm 9 $ atq 8 2009-08-17 02:00 a 10 2009-08-17 02:00 a The next example executes cmdfile at 3:30 PM (1530 hours) one week from today: $ at -f cmdfile 1530 +1 week job 12 at 2009-08-23 15:30 614 at The next at command executes a job at 7 PM on Thursday. This job uses find to create an intermediate file, redirects the output sent to standard error, and prints the file. $ at 7pm Thursday at> find / -name "core" -print >report.out 2>report.err at> lpr report.out at>CONTROL-D job 13 at 2009-08-18 19:00 The final example shows some of the output generated by the –c option when at is queried about the preceding job. Most of the lines show the environment; only the last few lines execute the commands: $ at -c 13 #!/bin/sh # atrun uid=500 gid=500 # mail sam 0 umask 2 PATH=/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:.; export PATH PWD=/home/sam/book.examples/99/cp; export PWD EXINIT=set\ ai\ aw; export EXINIT LANG=C; export LANG PS1=\\\$\ ; export PS1 ... cd /home/sam/book\.examples/99/cp || { echo 'Execution directory inaccessible' >&2 exit 1 } find / -name "core" -print >report.out 2>report.err lpr report.out bzip2 615 Compresses or decompresses files bzip2 [options] [file-list] bunzip2 [options] [file-list] bzcat [options] [file-list] bzip2recover [file] The bzip2 utility compresses files; bunzip2 restores files compressed with bzip2; and bzcat displays files compressed with bzip2. Arguments The file-list is a list of one or more files (no directories) that are to be compressed or decompressed. If file-list is empty or if the special option – is present, bzip2 reads from standard input. The ––stdout option causes bzip2 to write to standard output. Options Under Linux, bzip2, bunzip2, and bzcat accept the common options described on page 603. The Mac OS X version of bzip2 accepts long options tip Options for bzip2 preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––stdout –c Writes the results of compression or decompression to standard output. ––decompress –d Decompresses a file compressed with bzip2. This option with bzip2 is equivalent to the bunzip2 command. ––fast or ––best –n Sets the block size when compressing a file. The n is a digit from 1 to 9, where 1 (––fast) generates a block size of 100 kilobytes and 9 (––best) generates a block size of 900 kilobytes. The default level is 9. The options ––fast and ––best are provided for compatibility with gzip and do not necessarily yield the fastest or best compression. ––force –f Forces compression even if a file already exists, has multiple links, or comes directly from a terminal. The option has a similar effect with bunzip2. ––keep –k Does not delete input files while compressing or decompressing them. ––quiet –q Suppresses warning messages; does display critical messages. ––test ––verbose –t Verifies the integrity of a compressed file. Displays nothing if the file is OK. –v For each file being compressed, displays the name of the file, the compression ratio, the percentage of space saved, and the sizes of the decompressed and compressed files. bzip2 bzip2 616 bzip2 Discussion The bzip2 and bunzip2 utilities work similarly to gzip and gunzip; see the discussion of gzip (page 725) for more information. Normally bzip2 does not overwrite a file; you must use ––force to overwrite a file during compression or decompression. Notes See page 62 for additional information on and examples of tar. The bzip2 home page is bzip.org. The bzip2 utility does a better job of compressing files than does gzip. Use the ––bzip2 modifier with tar (page 847) to compress archive files with bzip2. bzcat file-list Works like cat except it uses bunzip2 to decompress file-list as it copies files to stan- dard output. bzip2recover Attempts to recover a damaged file that was compressed with bzip2. Examples In the following example, bzip2 compresses a file and gives the resulting file the same name with a .bz2 filename extension. The –v option displays statistics about the compression. $ ls -l total 728 -rw-r--r-- 1 sam sam 737414 Feb 20 19:05 bigfile $ bzip2 -v bigfile bigfile: 3.926:1, 2.037 bits/byte, 74.53% saved, 737414 in, 187806 out $ ls -l total 188 -rw-r--r-- 1 sam sam 187806 Feb 20 19:05 bigfile.bz2 Next touch creates a file with the same name as the original file; bunzip2 refuses to overwrite the file in the process of decompressing bigfile.bz2. The ––force option enables bunzip2 to overwrite the file. $ touch bigfile $ bunzip2 bigfile.bz2 bunzip2: Output file bigfile already exists. $ bunzip2 --force bigfile.bz2 $ ls -l total 728 -rw-r--r-- 1 sam sam 737414 Feb 20 19:05 bigfile cal 617 cal cal Displays a calendar cal [options] [[month] year] The cal utility displays a calendar. Arguments Options The arguments specify the month and year for which cal displays a calendar. The month is a decimal integer from 1 to 12 and the year is a decimal integer. Without any arguments, cal displays a calendar for the current month. When you specify a single argument, it is taken to be the year. –j (Julian) Displays Julian days—a calendar that numbers the days consecutively from January 1 (1) through December 31 (365 or 366). –m (Monday) Makes Monday the first day of the week. Without this option, Sunday is the first day of the week. L –y (year) Displays a calendar for the current year. L –3 (three months) Displays the previous, current, and next months. L Notes Do not abbreviate the year. The year 05 is not the same as 2005. The ncal (new cal) utility displays a more compact calendar. Examples The following command displays a calendar for August 2007: $ cal 8 2007vim: August 2007 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 Next is a Julian calendar for 1949: $ cal -j 1949 1949 Su Mo 2 9 16 23 30 ... 3 10 17 24 31 January Tu We Th Fr 4 11 18 25 7 14 21 28 5 12 19 26 6 13 20 27 Sa 1 8 15 22 29 Su Mo 37 44 51 58 38 45 52 59 February Tu We Th 32 33 34 39 40 41 46 47 48 53 54 55 Fr 35 42 49 56 Sa 36 43 50 57 618 cat cat Joins and displays files cat cat [options] [file-list] The cat utility copies files to standard output. You can use cat to display the contents of one or more text files on the screen. Arguments The file-list is a list of the pathnames of one or more files that cat processes. If you do not specify an argument or if you specify a hyphen (–) in place of a filename, cat reads from standard input. Options Under Linux, cat accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––show-all –A Same as –vET. L ––number-nonblank –b Numbers all lines that are not blank as they are written to standard output. –e (end) Same as –vE. ––show-ends ––number ––squeeze-blank –E Marks the ends of lines with dollar signs. L –n Numbers all lines as they are written to standard output. –s Removes extra blank lines so there are never two or more blank lines in a row. –t (tab) Same as –vT. ––show-tabs –T Displays TABs as ^I. L ––show-nonprinting –v Displays CONTROL characters with the caret notation (^M) and displays characters that have the high bit set (META characters) with the M- notation (page 216). This option does not convert TABs and LINEFEEDs. Use –T (––show-tabs) if you want to display TABs as ^I. LINEFEEDs cannot be displayed as anything but themselves; otherwise, the line would be too long. Notes See page 125 for a discussion of cat, standard input, and standard output. Use the od utility (page 776) to display the contents of a file that does not contain text (for example, an executable program file). Use the tac utility to display lines of a text file in reverse order. See the tac info page for more information. cat 619 The name cat is derived from one of the functions of this utility, catenate, which means to join together sequentially, or end to end. Set noclobber to avoid overwriting a file caution Despite cat's warning message, the shell destroys the input file (letter) before invoking cat in the following example: $ cat memo letter > letter cat: letter: input file is output file You can prevent overwriting a file in this situation by setting the noclobber variable (pages 129 and 377). Examples The following command displays the contents of the memo text file on the terminal: $ cat memo ... The next example catenates three text files and redirects the output to the all file: $ cat page1 letter memo > all You can use cat to create short text files without using an editor. Enter the following command line, type the text you want in the file, and press CONTROL-D on a line by itself: $ cat > new_file ... (text) ... CONTROL-D In this case cat takes input from standard input (the keyboard) and the shell redirects standard output (a copy of the input) to the file you specify. The CONTROL-D signals the EOF (end of file) and causes cat to return control to the shell. In the next example, a pipe sends the output from who to standard input of cat. The shell redirects cat's output to the file named output; after the commands have finished executing, output contains the contents of the header file, the output of who, and footer. The hyphen on the command line causes cat to read standard input after reading header and before reading footer. $ who | cat header - footer > output 620 cd cd Changes to another working directory cd cd [options] [directory] The cd builtin makes directory the working directory. Arguments The directory is the pathname of the directory you want to be the new working directory. Without an argument, cd makes your home directory the working directory. Using a hyphen in place of directory changes to the previous working directory. Options The following options are available under bash and dash only. –L (no dereference) If directory is a symbolic link, cd makes the symbolic link the working directory (default). –P (dereference) If directory is a symbolic link, cd makes the directory the symbolic link points to the working directory. Notes The cd command is a bash, dash, and tcsh builtin. See page 87 for a discussion of cd. Without an argument, cd makes your home directory the working directory; it uses the value of the HOME (bash; page 297) or home (tcsh; page 372) variable to determine the pathname of your home directory. With an argument of a hyphen, cd makes the previous working directory the working directory. It uses the value of the OLDPWD (bash) or owd (tcsh) variable to determine the pathname of the previous working directory. The CDPATH (bash; page 302) or cdpath (tcsh; page 372) variable contains a colon-separated list of directories that cd searches. Within the list a null directory name (::) or a period (:.:) represents the working directory. If CDPATH or cdpath is not set, cd searches only the working directory for directory. If this variable is set and directory is not an absolute pathname (does not begin with a slash), cd searches the directories in the list; if the search fails, cd searches the working directory. See page 302 for a discussion of CDPATH. Examples A cd command without an argument makes a user's home directory the working directory. In the following example, cd makes Max's home directory the working directory and the pwd builtin verifies the change. cd 621 $ pwd /home/max/literature $ cd $ pwd /home/max Under Mac OS X, home directories are stored in /Users, not /home. The next command makes the /home/max/literature directory the working directory: $ cd /home/max/literature $ pwd /home/max/literature Next the cd utility makes a subdirectory of the working directory the new working directory: $ cd memos $ pwd /home/max/literature/memos Finally cd uses the .. reference to the parent of the working directory (page 88) to make the parent the new working directory: $ cd .. $ pwd /home/max/literature 622 chgrp chgrp chgrp Changes the group associated with a file chgrp [options] group file-list chgrp [options] ––reference=rfile file-list L The chgrp utility changes the group associated with one or more files. The second format works under Linux only. Arguments The group is the name or numeric group ID of the new group. The file-list is a list of the pathnames of the files whose group association is to be changed. The rfile is the pathname of a file whose group is to become the new group associated with file-list. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––changes –c Displays a message for each file whose group is changed. L ––dereference For each file that is a symbolic link, changes the group of the file the link points to, not the symbolic link itself. Under Linux, this option is the default. L ––quiet or ––silent –f Suppresses warning messages about files whose permissions prevent you from changing their group IDs. ––no-dereference –h For each file that is a symbolic link, changes the group of the symbolic link, not the file the link points to. –H (partial dereference) For each file that is a symbolic link, changes the group of the file the link points to, not the symbolic link itself. This option affects files specified on the command line; it does not affect files found while descending a directory hierarchy. This option treats files that are not symbolic links normally and works with –R only. See page 623 for an example of the use of the –H versus –L options. –L (dereference) For each file that is a symbolic link, changes the group of the file the link points to, not the symbolic link itself. This option affects all files, treats files that are not symbolic links normally, and works with –R only. See page 623 for an example of the use of the –H versus –L options. –P (no dereference) For each file that is a symbolic link, changes the group of the symbolic link, not the file the link points to (default). This option affects all files, treats files that are not symbolic links normally, and works with –R only. See page 625 for an example of the use of the –P option. chgrp 623 ––recursive ––reference=rfile ––verbose Notes –R Recursively descends a directory specified in file-list and changes the group ID on all files in the directory hierarchy. Changes the group of the files in file-list to that of rfile. L –v For each file, displays a message saying whether its group was retained or changed. Only the owner of a file or a user working with root privileges can change the group association of a file. Unless you are working with root privileges, you must belong to the specified group to change the group ID of a file to that group. See page 631 for information on how to use chown to change the group associated with, as well as the owner of, a file. Examples The following command changes the group that the manuals file is associated with; the new group is pubs. $ chgrp pubs manuals –H versus –L The following examples demonstrate the difference between the –H and –L options. Initially all files and directories in the working directory are associated with the zach group: $ ls -lR .: total 12 -rw-r--r-- 1 zach zach 102 Jul drwxr-xr-x 2 zach zach 4096 Jul drwxr-xr-x 2 zach zach 4096 Jul 2 12:31 bb 2 15:34 dir1 2 15:33 dir4 ./dir1: total 4 -rw-r--r-- 1 zach zach 102 Jul lrwxrwxrwx 1 zach zach 7 Jul 2 12:32 dd 2 15:33 dir4.link -> ../dir4 ./dir4: total 8 -rw-r--r-- 1 zach zach 125 Jul -rw-r--r-- 1 zach zach 375 Jul 2 15:33 gg 2 15:33 hh When you call chgrp with the –R and –H options (–H does not work without –R), chgrp dereferences only symbolic links you list on the command line, which includes symbolic links found in directories you list on the command line. That means chgrp changes the group association of the files these links point to. It does not dereference symbolic links it finds as it descends into directory hierarchies, nor does it change symbolic links themselves. While descending the dir1 hierarchy, chgrp does not change dir4.link, but it does change dir4, the directory dir4.link points to. 624 chgrp $ chgrp -RH pubs bb dir1 $ ls -lR .: total 12 -rw-r--r-- 1 zach pubs 102 Jul drwxr-xr-x 2 zach pubs 4096 Jul drwxr-xr-x 2 zach pubs 4096 Jul 2 12:31 bb 2 15:34 dir1 2 15:33 dir4 ./dir1: total 4 -rw-r--r-- 1 zach pubs 102 Jul lrwxrwxrwx 1 zach zach 7 Jul 2 12:32 dd 2 15:33 dir4.link -> ../dir4 ./dir4: total 8 -rw-r--r-- 1 zach zach 125 Jul -rw-r--r-- 1 zach zach 375 Jul 2 15:33 gg 2 15:33 hh The –H option under Mac OS X caution The chgrp –H option works slightly differently under Mac OS X than it does under Linux. Under OS X, chgrp –RH changes the group of the symbolic link it finds in a directory listed on the command line and does not change the file the link points to. (It does not dereference the symbolic link.) When you run the preceding example under OS X, the group association of dir4 is not changed, but the group association of dir4.link is. If your program depends on how the –H option functions with a utility under OS X, test the option with that utility to determine exactly how it works. When you call chgrp with the –R and –L options (–L does not work without –R), chgrp dereferences all symbolic links: those you list on the command line and those it finds as it descends the directory hierarchy. It does not change the symbolic links themselves. This command changes the files dir4.link points to: $ chgrp -RL pubs bb dir1 $ ls -lR .: total 12 -rw-r--r-- 1 zach pubs 102 Jul drwxr-xr-x 2 zach pubs 4096 Jul drwxr-xr-x 2 zach pubs 4096 Jul 2 12:31 bb 2 15:34 dir1 2 15:33 dir4 ./dir1: total 4 -rw-r--r-- 1 zach pubs 102 Jul lrwxrwxrwx 1 zach zach 7 Jul 2 12:32 dd 2 15:33 dir4.link -> ../dir4 ./dir4: total 8 -rw-r--r-- 1 zach pubs 125 Jul -rw-r--r-- 1 zach pubs 375 Jul 2 15:33 gg 2 15:33 hh chgrp 625 –P When you call chgrp with the –R and –P options (–P does not work without –R), chgrp does not dereference symbolic links. It does change the group of the symbolic link itself. $ ls -l bb* -rw-r--r-- 1 zach zach 102 Jul lrwxrwxrwx 1 zach zach 2 Jul 2 12:31 bb 2 16:02 bb.link -> bb $ chgrp -PR pubs bb.link $ ls -l bb* -rw-r--r-- 1 zach zach 102 Jul lrwxrwxrwx 1 zach pubs 2 Jul 2 12:31 bb 2 16:02 bb.link -> bb 626 chmod chmod chmod Changes the access mode (permissions) of a file chmod [options] who operator permission file-list chmod [options] mode file-list chmod [options] ––reference=rfile file-list symbolic absolute referential L The chmod utility changes the ways in which a file can be accessed by the owner of the file, the group the file is associated with, and/or all other users. You can specify the new access mode absolutely or symbolically. Under Linux you can also specify the mode referentially (third format). Under Mac OS X, you can use chmod to modify ACLs (page 932). Arguments Arguments specify which files are to have their modes changed in what ways. The rfile is the pathname of a file whose permissions are to become the new permissions of file-list. Symbolic You can specify multiple sets of symbolic modes (who operator permission) by separating each set from the next with a comma. The chmod utility changes the access permission for the class of users specified by who. The class of users is designated by one or more of the letters specified in the who column of Table V-4. Table V-4 Symbolic mode user class specification who User class Meaning u User Owner of the file g Group Group the file is associated with o Other All other users a All Use in place of ugo Table V-5 lists the symbolic mode operators. Table V-5 Symbolic mode operators operator Meaning + Adds the permission for the specified user class – Removes the permission for the specified user class = Sets the permission for the specified user class; resets all other permissions for that user class The access permission is specified by one or more of the letters listed in Table V-6. chmod 627 Table V-6 Symbolic mode permissions permission Meaning r Sets read permission w Sets write permission x Sets execute permission s Sets the user ID or group ID (depending on the who argument) to that of the owner of the file while the file is being executed (For more information see page 96.) t Sets the sticky bit (Only a user working with root privileges can set the sticky bit, and it can be used only with u; see page 980.) X Makes the file executable only if it is a directory or if another user class has execute permission u Sets the specified permissions to those of the owner g Sets the specified permissions to those of the group o Sets the specified permissions to those of others Absolute You can use an octal number to specify the access mode. Construct the number by ORing the appropriate values from Table V-7. To OR two or more octal numbers from this table, just add them. (Refer to Table V-8 on the next page for examples.) Table V-7 Absolute mode specifications mode Meaning 4000 Sets the user ID when the program is executed (page 96) 2000 Sets the group ID when the program is executed (page 96) 1000 Sticky bit (page 980) 0400 Owner can read the file 0200 Owner can write to the file 0100 Owner can execute the file 0040 Group can read the file 0020 Group can write to the file 0010 Group can execute the file 0004 Others can read the file 0002 Others can write to the file 0001 Others can execute the file 628 chmod Table V-8 lists some typical modes. Table V-8 Options Examples of absolute mode specifications Mode Meaning 0777 Owner, group, and others can read, write, and execute the file 0755 Owner can read, write, and execute the file; group and others can read and execute the file 0711 Owner can read, write, and execute the file; group and others can execute the file 0644 Owner can read and write the file; group and others can read the file 0640 Owner can read and write the file, group can read the file, and others cannot access the file Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––changes –c Displays a message for each file whose permissions are changed. L ––quiet or ––silent –f Suppresses warning messages about files whose permissions prevent chmod from changing the permissions of the file. –H (partial dereference) For each file that is a symbolic link, changes permissions of the file the link points to, not the symbolic link itself. This option affects files specified on the command line; it does not affect files found while descending a directory hierarchy. This option treats files that are not symbolic links normally and works with –R only. See page 623 for an example of the use of the –H versus –L options. O –L (dereference) For each file that is a symbolic link, changes permissions of the file the link points to, not the symbolic link itself. This option affects all files, treats files that are not symbolic links normally, and works with –R only. See page 623 for an example of the use of the –H versus –L options. O –P (no dereference) For each file that is a symbolic link, changes permissions of the symbolic link, not the file the link points to. This option affects all files, treats files that are not symbolic links normally, and works with –R only. See page 625 for an example of the use of the –P option. O ––recursive ––reference=rfile –R Recursively descends a directory specified in file-list and changes the permissions on all files in the directory hierarchy. Changes the permissions of the files in file-list to that of rfile. L chmod 629 ––verbose Notes –v For each file, displays a message saying that its permissions were changed (even if they were not changed) and specifying the permissions. Use ––changes to display messages only when permissions are actually changed. Only the owner of a file or a user working with root privileges can change the access mode, or permissions, of a file. When you use symbolic arguments, you can omit the permission from the command line when the operator is =. This omission takes away all permissions for the specified user class. See the second example in the next section. Under Linux, chmod never changes the permissions of symbolic links. Under Linux, chmod dereferences symbolic links found on the command line. In other words, chmod changes the permissions of files that symbolic links found on the command line point to; chmod does not affect files found while descending a directory hierarchy. This behavior mimics the behavior of the Mac OS X –H option. See page 94 for another discussion of chmod. See page 933 for a discussion of using chmod under Mac OS X to change ACLs. Examples See page 932 for examples of using chmod to change ACLs under Mac OS X. The following examples show how to use the chmod utility to change the permissions of the file named temp. The initial access mode of temp is shown by ls. See "Discussion" on page 748 for information about the ls display. $ ls -l temp -rw-rw-r-- 1 max pubs 57 Jul 12 16:47 temp When you do not follow an equal sign with a permission, chmod removes all permissions for the specified user class. The following command removes all access permissions for the group and all other users so only the owner has access to the file: $ chmod go= temp $ ls -l temp -rw------- 1 max pubs 57 Jul 12 16:47 temp The next command changes the access modes for all users (owner, group, and others) to read and write. Now anyone can read from or write to the file. $ chmod a=rw temp $ ls -l temp -rw-rw-rw- 1 max pubs 57 Jul 12 16:47 temp Using an absolute argument, a=rw becomes 666. The next command performs the same function as the previous one: $ chmod 666 temp 630 chmod The next command removes write access permission for other users. As a result, members of the pubs group can read from and write to the file, but other users can only read from the file: $ chmod o-w temp $ ls -l temp -rw-rw-r-- 1 max pubs 57 Jul 12 16:47 temp The following command yields the same result, using an absolute argument: $ chmod 664 temp The next command adds execute access permission for all users: $ chmod others temp $ ls -l temp -rwxrwxr-x 1 max pubs 57 Jul 12 16:47 temp If temp is a shell script or other executable file, all users can now execute it. (The operating system requires read and execute access to execute a shell script but only execute access to execute a binary file.) The absolute command that yields the same result is $ chmod 775 temp The final command uses symbolic arguments to achieve the same result as the preceding one. It sets permissions to read, write, and execute for the owner and the group, and to read and execute for other users. A comma separates the sets of symbolic modes. $ chmod ug=rwx,o=rx temp chown 631 Changes the owner of a file and/or the group the file is associated with chown [options] owner file-list chown [options] owner:group file-list chown [options] owner: file-list chown [options] :group file-list chown [options] ––reference=rfile file-list L The chown utility changes the owner of a file and/or the group the file is associated with. Only a user working with root privileges can change the owner of a file. Only a user working with root privileges or the owner of a file who belongs to the new group can change the group a file is associated with. The last format works under Linux only. Arguments The owner is the username or numeric user ID of the new owner. The file-list is a list of the pathnames of the files whose ownership and/or group association you want to change. The group is the group name or numeric group ID of the new group the file is to be associated with. The rfile is the pathname of a file whose owner and/or group association is to become the new owner and/or group association of file-list. Table V-9 shows the ways you can specify the new owner and/or group. Table V-9 Options ––changes Specifying the new owner and/or group Argument Meaning owner The new owner of file-list; the group is not changed owner:group The new owner of and new group associated with file-list owner: The new owner of file-list; the group associated with file-list is changed to the new owner's login group :group The new group associated with file-list; the owner is not changed Under Linux, chown accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. –c Displays a message for each file whose ownership/group is changed. L ––dereference Changes the ownership/group of the files symbolic links point to, not the symbolic links themselves. Under Linux, this option is the default. L ––quiet or ––silent –f Suppresses error messages about files whose ownership and/or group association chown cannot change. chown chown 632 chown –H (partial dereference) For each file that is a symbolic link, changes the owner and/or group association of the file the link points to, not the symbolic link itself. This option affects files specified on the command line; it does not affect files found while descending a directory hierarchy. This option treats files that are not symbolic links normally and works with –R only. See page 623 for an example of the use of the –H versus –L options. ––no-dereference –h For each file that is a symbolic link, changes the owner and/or group associa- tion of the symbolic link, not the file the link points to. –L (dereference) For each file that is a symbolic link, changes the owner and/or group association of the file the link points to, not the symbolic link itself. This option affects all files, treats files that are not symbolic links normally, and works with –R only. See page 623 for an example of the use of the –H versus –L options. –P (no dereference) For each file that is a symbolic link, changes the owner and/or group association of the symbolic link, not the file the link points to. This option affects all files, treats files that are not symbolic links normally, and works with –R only. See page 625 for an example of the use of the –P option. ––recursive –R When you include directories in the file-list, this option descends the directory hierarchy, setting the specified owner and/or group association for all files in the hierarchy. ––reference=rfile Changes the owner and/or group association of the files in the file-list to that of rfile. L ––verbose –v Displays for each file a message saying whether its owner and/or group association was retained or changed. Notes The chown utility clears setuid and setgid bits when it changes the owner of a file. Examples The following command changes the owner of the chapter1 file in the manuals directory. The new owner is Sam. # chown sam manuals/chapter1 The following command makes Max the owner of, and Max's login group the group associated with, all files in the /home/max/literature directory and in all its subdirectories. # chown -R max: /home/max/literature Under Mac OS X, home directories are stored in /Users, not /home. The next command changes the ownership of the files in literature to max and the group associated with these files to pubs: # chown max:pubs /home/max/literature/* chown 633 The final example changes the group association of the files in manuals to pubs without altering their ownership. The owner of the files, who is executing this command, must belong to the pubs group. $ chown :pubs manuals/* 634 cmp cmp Compares two files cmp cmp [options] file1 [file2 [skip1 [skip2]]] The cmp utility displays the differences between two files on a byte-by-byte basis. If the files are the same, cmp is silent. If the files differ, cmp displays the byte and line number of the first difference. Arguments The file1 and file2 arguments are the pathnames of the files that cmp compares. If file2 is omitted, cmp uses standard input instead. Using a hyphen (–) in place of file1 or file2 causes cmp to read standard input instead of that file. The skip1 and skip2 arguments are decimal numbers indicating the number of bytes to skip in each file before beginning the comparison. You can use the standard multiplicative suffixes after skip1 and skip2; see Table V-1 on page 603. Options ––print–bytes Under Linux and OS X, cmp accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. –b Displays more information, including filenames, byte and line numbers, and the octal and ASCII values of the first differing byte. ––ignore–initial=n1[:n2] –i n1[:n2] Without n2, skips the first n1 bytes in both files before beginning the comparison. With n1 and n2, skips the first n1 bytes in file1 and skips the first n2 bytes in file2 before beginning the comparison. You can follow n1 and/or n2 with one of the multiplicative suffixes listed in Table V-1 on page 603. ––verbose –l (lowercase "l") Instead of stopping at the first byte that differs, continues comparing the two files and displays both the location and the value of each byte that differs. Locations are displayed as decimal byte count offsets from the beginning of the files; byte values are displayed in octal. The comparison terminates when an EOF is encountered on either file. ––silent or ––quiet –s Suppresses output from cmp; only sets the exit status (see "Notes"). Notes Byte and line numbering start at 1. The cmp utility does not display a message if the files are identical; it only sets the exit status. This utility returns an exit status of 0 if the files are the same and an exit status of 1 if they are different. An exit status greater than 1 means an error occurred. When you use skip1 (and skip2), the offset values cmp displays are based on the byte where the comparison began. cmp 635 Under Mac OS X, cmp compares data forks (page 929) of a file only. If you want to compare resource forks, you can manually compare the ..namedfork/rsrc files (page 930) for the target files. Unlike diff (page 663), cmp works with binary as well as ASCII files. Examples The examples use the files a and b shown below. These files have two differences. The first difference is that the word lazy in file a is replaced by lasy in file b. The second difference is more subtle: A TAB character appears just before the NEWLINE character in file b. $ cat a The quick brown fox jumped over the lazy dog. $ cat b The quick brown fox jumped over the lasy dog.TAB The first example uses cmp without any options to compare the two files. The cmp utility reports that the files are different and identifies the offset from the beginning of the files where the first difference is found: $ cmp a b a b differ: char 39, line 1 You can display the values of the bytes at that location by adding the –b (––print–bytes) option: $ cmp --print-bytes a b a b differ: char 39, line 1 is 172 z 163 s The –l option displays all bytes that differ between the two files. Because this option creates a lot of output if the files have many differences, you may want to redirect the output to a file. The following example shows the two differences between files a and b. The –b option displays the values for the bytes as well. Where file a has a CONTROL–J (NEWLINE), file b has a CONTROL–I (TAB). The message saying that the end of file on file a has been reached indicates that file b is longer than file a. $ cmp -lb a b 39 172 z 163 s 46 12 ^J 11 ^I cmp: EOF on a In the next example, the ––ignore–initial option is used to skip over the first difference in the files. The cmp utility now reports on the second difference. The difference is put at character 7, which is the 46th character in the original file b (7 characters past the ignored 39 characters). $ cmp --ignore-initial=39 a b a b differ: char 7, line 1 You can use skip1 and skip2 in place of the ––ignore–initial option used in the preceding example: $ cmp a b 39 39 a b differ: char 7, line 1 636 comm comm comm Compares sorted files comm [options] file1 file2 The comm utility displays a line-by-line comparison of two sorted files. The first of the three columns it displays lists the lines found only in file1, the second column lists the lines found only in file2, and the third lists the lines common to both files. Arguments The file1 and file2 arguments are pathnames of the files that comm compares. Using a hyphen (–) in place of file1 or file2 causes comm to read standard input instead of that file. Options You can combine the options. With no options, comm produces three-column output. –1 Does not display column 1 (does not display lines found only in file1). –2 Does not display column 2 (does not display lines found only in file2). –3 Does not display column 3 (does not display lines found in both files). Notes If the files have not been sorted, comm will not work properly. Lines in the second column are preceded by one are preceded by two TAB s. TAB, and those in the third column The exit status indicates whether comm completed normally (0) or abnormally (not 0). Examples The following examples use two files, c and d. As with all input to comm, the files have already been sorted: $ cat c bbbbb ccccc ddddd eeeee fffff $ cat d aaaaa ddddd eeeee ggggg hhhhh Refer to sort on page 817 for information on sorting files. comm 637 The following example calls comm without any options, so it displays three columns. The first column lists those lines found only in file c, the second column lists those found in d, and the third lists the lines found in both c and d: $ comm c d aaaaa bbbbb ccccc ddddd eeeee fffff ggggg hhhhh The next example shows the use of options to prevent comm from displaying columns 1 and 2. The result is column 3, a list of the lines common to files c and d: $ comm -12 c d ddddd eeeee 638 configure configure configure Configures source code automatically ./configure [options] The configure script is part of the GNU Configure and Build System. Software developers who supply source code for their products face the problem of making it easy for relatively naive users to build and install their software package on a wide variety of machine architectures, operating systems, and system software. To facilitate this process many software developers supply a shell script named configure with their source code. When you run configure, it determines the capabilities of the local system. The data collected by configure is used to build the makefiles with which make (page 753) builds the executables and libraries. You can adjust the behavior of configure by specifying command-line options and environment variables. Options The Mac OS X version of configure accepts long options tip Options for configure preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––disable-feature Works in the same manner as ––enable-feature except it disables support for feature. ––enable-feature The feature is the name of a feature that can be supported by the software being configured. For example, configuring the Z Shell source code with the command configure ––enable-zsh-mem configures the source code to use the special memory allocation routines provided with zsh instead of using the system memory allocation routines. Check the README file supplied with the software distribution to see the choices available for feature. ––help Displays a detailed list of all options available for use with configure. The contents of this list depends on the software you are installing. ––prefix=directory By default configure builds makefiles that install software in the /usr/local directory hierarchy (when you give the command make install). To install software into a different directory, replace directory with the pathname of the desired directory. ––with-package The package is the name of an optional package that can be included with the software you are configuring. For example, if you configure the source code for the Windows emulator wine with the command configure ––with-dll, the source code is configured to build a shared library of Windows emulation support. Check the README file supplied with the software you are installing to configure 639 see the choices available for package. The command configure ––help usually displays the choices available for package. Discussion The GNU Configure and Build System allows software developers to distribute software that can configure itself to be built on a variety of systems. It builds a shell script named configure, which prepares the software distribution to be built and installed on a local system. The configure script searches the local system to find the dependencies for the software distribution and constructs the appropriate makefiles. Once you have run configure, you can build the software with a make command and install the software with a make install command. The configure script determines which C compiler to use (usually gcc) and specifies a set of flags to pass to that compiler. You can set the environment variables CC and CFLAGS to override these values. See the "Examples" section. Notes Each package that uses the GNU autoconfiguration utility provides its own custom copy of configure, which the software developer created using the GNU autoconf utility (www.gnu.org/software/autoconf). Read the README and INSTALL files that are provided with the software you are installing for information about the available options. The configure scripts are self-contained and run correctly on a wide variety of systems. You do not need any special system resources to use configure. The configure utility will exit with an error message if a dependency is not installed. Examples The simplest way to call configure is to cd to the base directory for the software you are installing and then run the following command: $ ./configure The ./ is prepended to the command name to ensure you are running the configure script supplied with the software you are installing. For example, to cause configure to build makefiles that pass the flags –Wall and –O2 to gcc, give the following command from bash: $ CFLAGS="-Wall -O2" ./configure If you are using tcsh, give the following command: tcsh $ env CFLAGS="-Wall -O2" ./configure 640 cp cp Copies files cp cp [options] source-file destination-file cp [options] source-file-list destination-directory The cp utility copies one or more files. It can either make a copy of a single file (first format) or copy one or more files to a directory (second format). With the –R option, cp can copy directory hierarchies. Arguments The source-file is the pathname of the file that cp makes a copy of. The destinationfile is the pathname that cp assigns to the resulting copy of the file. The source-file-list is a list of one or more pathnames of files that cp makes copies of. The destination-directory is the pathname of the directory in which cp places the copied files. With this format, cp gives each copied file the same simple filename as its source-file. The –R option enables cp to copy directory hierarchies recursively from the sourcefile-list into the destination-directory. Options Under Linux, cp accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––archive –a Attempts to preserve the owner, group, permissions, access date, and modification date of source file(s) while copying recursively without dereferencing symbolic links. Same as –dpR. L ––backup –b If copying a file would remove or overwrite an existing file, this option makes a backup copy of the file that would be overwritten. The backup copy has the same name as the destination-file with a tilde (~) appended to it. When you use both ––backup and ––force, cp makes a backup copy when you try to copy a file over itself. For more backup options, search for Backup options in the core utils info page. L –d For each file that is a symbolic link, copies the symbolic link, not the file the link points to. Also preserves hard links in destination-files that exist between corresponding source-files. This option is equivalent to ––no-dereference and ––preserve=links. L ––force –f When the destination-file exists and cannot be opened for writing, causes cp to try to remove destination-file before copying source-file. This option is useful when the user copying a file does not have write permission to an existing destination-file but does have write permission to the directory containing the cp 641 destination-file. Use this option with –b to back up a destination file before removing or overwriting it. –H (partial dereference) For each file that is a symbolic link, copies the file the link points to, not the symbolic link itself. This option affects files specified on the command line; it does not affect files found while descending a directory hierarchy. This option treats files that are not symbolic links normally. Under OS X works with –R only. See page 623 for an example of the use of the –H versus –L options. ––interactive –i Prompts you whenever cp would overwrite a file. If you respond with a string that starts with y or Y, cp copies the file. If you enter anything else, cp does not copy the file. ––dereference –L (dereference) For each file that is a symbolic link, copies the file the link points to, not the symbolic link itself. This option affects all files and treats files that are not symbolic links normally. Under OS X works with –R only. See page 623 for an example of the use of the –H versus –L options. ––no-dereference –P (no dereference) For each file that is a symbolic link, copies the symbolic link, not the file the link points to. This option affects all files and treats files that are not symbolic links normally. Under OS X works with –R only. See page 625 for an example of the use of the –P option. ––preserve[=attr] –p Creates a destination-file with the same owner, group, permissions, access date, and modification date as the source-file. The –p option does not take an argument. Without attr, ––preserve works as described above. The attr is a commaseparated list that can include mode (permissions and ACLs), ownership (owner and group), timestamps (access and modification dates), links (hard links), and all (all attributes). ––parents Copies a relative pathname to a directory, creating directories as needed. See the "Examples" section. L ––recursive –R or –r Recursively copies directory hierarchies including ordinary files. Under Linux, the ––no-dereference (–d) option is implied: With the –R, –r, or ––recursive option, cp copies the links (not the files the links point to). The –r and ––recursive options are available under Linux only. ––update –u Copies only when the destination-file does not exist or when it is older than the source-file (i.e., this option will not overwrite a newer destination file). L ––verbose –v Displays the name of each file as cp copies it. 642 cp Notes Under Linux, cp dereferences symbolic links unless you use one or more of the –R, –r, ––recursive, –P, –d, or ––no-dereference options. As explained on the previous page, under Linux the –H option dereferences only symbolic links listed on the command line. Under Mac OS X, without the –R option, cp always dereferences symbolic links; with the –R option, cp does not dereference symbolic links (–P is the default) unless you specify –H or –L. Many options are available for cp under Linux. See the coreutils info page for a complete list. If the destination-file exists before you execute a cp command, cp overwrites the file, destroying the contents but leaving the access privileges, owner, and group associated with the file as they were. If the destination-file does not exist, cp uses the access privileges of the source-file. The user who copies the file becomes the owner of the destination-file and the user's login group becomes the group associated with the destination-file. Using the –p option without any arguments causes cp to attempt to set the owner, group, permissions, access date, and modification date to match those of the source-file. Unlike with the ln utility (page 740), the destination-file that cp creates is independent of its source-file. Under Mac OS X version 10.4 and later, cp copies extended attributes (page 928). Examples The first command makes a copy of the file letter in the working directory. The name of the copy is letter.sav. $ cp letter letter.sav The next command copies all files with filenames ending in .c into the archives directory, which is a subdirectory of the working directory. Each copied file retains its simple filename but has a new absolute pathname. The –p (––preserve) option causes the copied files in archives to have the same owner, group, permissions, access date, and modification date as the source files. $ cp -p *.c archives The next example copies memo from Sam's home directory to the working directory: $ cp ~sam/memo . The next example runs under Linux and uses the ––parents option to copy the file memo/thursday/max to the dir directory as dir/memo/thursday/max. The find utility shows the newly created directory hierarchy. cp 643 $ cp --parents memo/thursday/max dir $ find dir dir dir/memo dir/memo/thursday dir/memo/thursday/max The following command copies the files named memo and letter into another directory. The copies have the same simple filenames as the source files (memo and letter) but have different absolute pathnames. The absolute pathnames of the copied files are /home/sam/memo and /home/sam/letter, respectively. $ cp memo letter /home/sam The final command demonstrates one use of the –f (––force) option. Max owns the working directory and tries unsuccessfully to copy one over another file (me) that he does not have write permission for. Because he has write permission to the directory that holds me, Max can remove the file but cannot write to it. The –f (––force) option unlinks, or removes, me and then copies one to the new file named me. $ ls -ld drwxrwxr-x 2 max max 4096 Oct 21 22:55 . $ ls -l -rw-r--r-1 root root 3555 Oct 21 22:54 me -rw-rw-r-1 max max 1222 Oct 21 22:55 one $ cp one me cp: cannot create regular file 'me': Permission denied $ cp -f one me $ ls -l -rw-r--r-1 max max 1222 Oct 21 22:58 me -rw-rw-r-1 max max 1222 Oct 21 22:55 one If Max had used the –b (––backup) option in addition to –f (––force), cp would have created a backup of me named me~. Refer to "Directory Access Permissions" on page 98 for more information. 644 cpio cpio Creates an archive, restores files from an archive, or copies a directory hierarchy cpio cpio ––create|–o [options] cpio ––extract|–i [options] [pattern-list] cpio ––pass-through|–p [options] destination-directory The cpio utility has three modes of operation: Create (copy-out) mode places multiple files into a single archive file, extract (copy-in) mode restores files from an archive, and pass-through (copy-pass) mode copies a directory hierarchy. The archive file cpio creates can be saved on disk, tape, other removable media, or a remote system. Create mode reads a list of ordinary or directory filenames from standard input and writes the resulting archive file to standard output. You can use this mode to create an archive. Extract mode reads an archive from standard input and extracts files from that archive. You can restore all files from the archive or only those files whose names match a pattern. Pass-through mode reads a list of names of ordinary or directory files from standard input and copies the files to a specified directory. Arguments By default cpio in extract mode extracts all files found in the archive. You can choose to extract files selectively by supplying a pattern-list. If the name of a file in the archive matches one of the patterns in pattern-list, cpio extracts that file; otherwise, it ignores the file. The patterns in a cpio pattern-list are similar to shell wildcards (page 136) except that pattern-list match slashes (/) and a

leading period (.) in a filename. In pass-through mode you must give the name of the destination-directory as an argument to cpio. Options A major option specifies the mode in which cpio operates: create, extract, or pass-through. Major Options You must include exactly one of these options. Options preceded by a double hyphen (––) work under Linux only. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––extract –i (copy-in mode) Reads the archive from standard input and extracts files. Without a pattern-list on the command line, cpio extracts all files from the archive. With a pattern-list, cpio extracts only files with names that match one of the patterns in pattern-list. The following example extracts from the device mounted on /dev/sde1 only those files whose names end in .c: $ cpio -i *.c < /dev/sde1 The backslash prevents the shell from expanding the argument to cpio. * before it passes the cpio ––create 645 –o (copy-out mode) Constructs an archive from the files named on standard input. These files may be ordinary or directory files, and each must appear on a separate line. The archive is written to standard output as it is built. The find utility frequently generates the filenames that cpio uses. The following command builds an archive of the entire local system and writes it to the device mounted on /dev/sde1: $ find / -depth -print | cpio -o >/dev/sde1 The –depth option causes find to search for files in a depth-first search, thereby reducing the likelihood of permissions problems when you restore the files from the archive. See the discussion of this option on page 647. ––pass-through –p (copy-pass mode) Copies files from one place on the system to another. Instead of constructing an archive file containing the files named on standard input, cpio copies them to the destination-directory (the last argument on the cpio command line). The effect is the same as if you had created an archive with copy-out mode and then extracted the files with copy-in mode, except using pass-through mode avoids creating an archive. The following example copies the files in the working directory and all subdirectories into ~max/code: $ find . -depth -print | cpio -pdm ~max/code Other Options The following options alter the behavior of cpio. These options work with one or more of the preceding major options. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––reset-access-time –a Resets the access times of source files after copying them so they have the same access time after copying as they did before. –B (block) Sets the block size to 5,120 bytes instead of the default 512 bytes. Under Linux this option affects input and output block sizes; under OS X it affects only output block sizes. Sets the block size used for input and output to n 512-byte blocks. L ––block-size=n –c (compatible) Writes header information in ASCII so older (incompatible) cpio utilities on other systems can read the file. This option is rarely needed. ––make-directories –d Creates directories as needed when copying files. For example, you need this option when you are extracting files from an archive with a file list generated by find with the –depth option. This option can be used only with the –i (––extract) and –p (––pass-through) options. 646 cpio ––pattern-file=filename –E filename Reads pattern-list from filename, one pattern per line. Additionally, you can specify pattern-list on the command line. ––file=archive –F archive Uses archive as the name of the archive file. In extract mode, reads from archive instead of standard input. In create mode, writes to archive instead of standard output. You can use this option to access a device on another system on a network; see the –f (––file) option to tar (page 847) for more information. ––nonmatching –f (flip) Reverses the sense of the test performed on pattern-list when extracting files from an archive. Files are extracted from the archive only if they do not match any of the patterns in the pattern-list. ––help ––dereference ––link Displays a list of options. L –L For each file that is a symbolic link, copies the file the link points to (not the symbolic link itself). This option treats files that are not symbolic links normally. –l When possible, links files instead of copying them. ––preserve-modification-time –m Preserves the modification times of files that are extracted from an archive. Without this option the files show the time they were extracted. With this option the created files show the time they had when they were copied into the archive. ––no-absolute-filenames In extract mode, creates all filenames relative to the working directory—even files that were archived with absolute pathnames. L ––rename –r Allows you to rename files as cpio copies them. When cpio prompts you with the name of a file, you respond with the new name. The file is then copied with the new name. If you press RETURN instead, cpio does not copy the file. ––list –t (table of contents) Displays a table of contents of the archive. This option works only with the –i (––extract) option, although no files are actually extracted from the archive. With the –v (––verbose) option, it displays a detailed table of contents in a format similar to that used by ls –l. ––unconditional –u Overwrites existing files regardless of their modification times. Without this option cpio will not overwrite a more recently modified file with an older one; it displays a warning message. ––verbose –v Lists files as they are processed. With the –t (––list) option, it displays a detailed table of contents in a format similar to that used by ls –l. cpio Discussion 647 Without the –u (––unconditional) option, cpio will not overwrite a more recently modified file with an older file. You can use both ordinary and directory filenames as input when you create an archive. If the name of an ordinary file appears in the input list before the name of its parent directory, the ordinary file appears before its parent directory in the archive as well. This order can lead to an avoidable error: When you extract files from the archive, the child has nowhere to go in the file structure if its parent has not yet been extracted. Making sure that files appear after their parent directories in the archive is not always a solution. One problem occurs if the –m (––preserve-modification-time) option is used when extracting files. Because the modification time of a parent directory is updated whenever a file is created within it, the original modification time of the parent directory is lost when the first file is written to it. The solution to this potential problem is to ensure that all files appear before their parent directories when creating an archive and to create directories as needed when extracting files from an archive. When you use this technique, directories are extracted only after all files have been written to them and their modification times are preserved. With the –depth option, find generates a list of files, with all children appearing in the list before their parent directories. If you use this list to create an archive, the files are in the proper order. (Refer to the first example in the next section.) When extracting files from an archive, the –d (––make-directories) option causes cpio to create parent directories as needed and the –m (––preserve-modification-time) option does just what its name says. Using this combination of utilities and options preserves directory modification times through a create/extract sequence. This strategy also solves another potential problem. Sometimes a parent directory may not have permissions set so that you can extract files into it. When cpio automatically creates the directory with –d (––make-directories), you can be assured that you have write permission to the directory. When the directory is extracted from the archive (after all the files are written into the directory), it is extracted with its original permissions. Examples The first example creates an archive of the files in Sam’s home directory, writing the archive to a USB flash drive mounted at /dev/sde1. $ find /home/sam -depth -print | cpio -oB >/dev/sde1 The find utility produces the filenames that cpio uses to build the archive. The –depth option causes all entries in a directory to be listed before listing the directory name itself, making it possible for cpio to preserve the original modification times of directories (see the preceding “Discussion” section). Use the –d (––make-directories) and 648 cpio –m (––preserve-modification-time) options when you extract files from this archive (see the following examples). The –B option sets the block size to 5,120 bytes. Under Mac OS X, home directories are stored in /Users, not /home. To check the contents of the archive file and display a detailed listing of the files it contains, give the following command: $ cpio -itv < /dev/sde1 The following command restores the files that formerly were in the memo subdirectory of Sam’s home directory: $ cpio -idm /home/sam/memo/* < /dev/sde1 The –d (––make-directories) option ensures that any subdirectories that were in the memo directory are re-created as needed. The –m (––preserve-modification-time) option preserves the modification times of files and directories. The asterisk in the regular expression is escaped to keep the shell from expanding it. The next command is the same as the preceding command except it uses the Linux ––no-absolute-filenames option to re-create the memo directory in the working directory, which is named memocopy. The pattern does not start with the slash that represents the root directory, allowing cpio to create the files with relative pathnames. $ pwd /home/sam/memocopy $ cpio -idm --no-absolute-filenames home/sam/memo/* < /dev/sde1 The final example uses the –f option to restore all files in the archive except those that were formerly in the memo subdirectory: $ cpio -ivmdf /home/sam/memo/* < /dev/sde1 The –v option lists the extracted files as cpio processes the archive, verifying the expected files are extracted. crontab 649 Maintains crontab files crontab [–u user-name] filename crontab [–u user-name] option A crontab file associates periodic times (such as 14:00 on Wednesdays) with commands. The cron utility executes each command at the specified time. When you are working as yourself, the crontab utility installs, removes, lists, and allows you to edit your crontab file. A user working with root privileges can work with any user’s crontab file. Arguments The first format copies the contents of filename (which contains crontab commands) into the crontab file of the user who runs the command or that of username. When the user does not have a crontab file, this process creates a new one; when the user has a crontab file, this process overwrites the file. When you replace filename with a hyphen (–), crontab reads commands from standard input. The second format lists, removes, or edits the crontab file, depending on which option you specify. Options Choose only one of the –e, –l, or –r options. A user working with root privileges can use –u with one of these options. –e (edit) Runs the text editor specified by the VISUAL or EDITOR shell variable on the crontab file, enabling you to add, change, or delete entries. This option installs the modified crontab file when you exit from the editor. –l (list) Displays the contents of the crontab file. –r (remove) Deletes the crontab file. –u username (user) Works on username’s crontab file. Only a user working with root privileges can use this option. Notes This section covers the versions of cron, crontab, and crontab files that were written by Paul Vixie—hence this version of cron is called Vixie cron. These versions are POSIX compliant and differ from an earlier version of Vixie cron as well as from the classic SVR3 syntax. User crontab files are kept in the /var/spool/cron or /var/spool/cron/crontabs directory. Each file is named with the username of the user that it belongs to. The system utility named cron reads the crontab files and runs the commands. If a command line in a crontab file does not redirect its output, all output sent to standard crontab 650 crontab output and standard error is mailed to the user unless you set the MAILTO variable within the crontab file to a different username. To make the system administrator’s job easier, the directories named /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, and /etc/cron.monthly hold crontab files that, on most systems, are run by run-parts, which in turn are run by the /etc/crontab file. Each of these directories contains files that execute system tasks at the interval named by the directory. A user working with root privileges can add files to these directories instead of adding lines to root’s crontab file. A typical /etc/crontab file looks like this: $ cat /etc/crontab SHELL=/bin/bash PATH=/sbin:/bin:/usr/sbin:/usr/bin MAILTO=root HOME=/ # run-parts 01 * * * * root 02 4 * * * root 22 4 * * 0 root 42 4 1 * * root run-parts /etc/cron.hourly run-parts /etc/cron.daily run-parts /etc/cron.weekly run-parts /etc/cron.monthly Each entry in a crontab file begins with five fields that specify when the command is to run (minute, hour, day of the month, month, and day of the week). The cron utility interprets an asterisk appearing in place of a number as a wildcard representing all possible values. In the day-of-the-week field, you can use either 7 or 0 to represent Sunday. It is a good practice to start cron jobs a variable number of minutes before or after the hour, half-hour, or quarter-hour. When you start jobs at these times, it becomes less likely that many processes will start at the same time, thereby potentially overloading the system. When cron starts (usually when the system is booted), it reads all of the crontab files into memory. The cron utility mostly sleeps but wakes up once a minute, reviews all crontab entries it has stored in memory, and runs whichever jobs are due to be run at that time. cron.allow, cron.deny By creating, editing, and removing the cron.allow and cron.deny files, a user working with root privileges determines which users can run crontab. Under Linux these files are kept in the /etc directory; under Mac OS X they are kept in /var/at (which has a symbolic link at /usr/lib/cron). When you create a cron.deny file with no entries and no cron.allow file exists, everyone can use crontab. When the cron.allow file exists, only users listed in that file can use crontab, regardless of the presence and contents of cron.deny. Otherwise, you can list in the cron.allow file those users who are allowed to use crontab and in cron.deny those users who are not allowed to use it. (Listing a user in cron.deny is not strictly necessary because, if a cron.allow file exists and the user is not listed in it, the user will not be able to use crontab anyway.) crontab Examples 651 The following example uses crontab –l to list the contents of Sam’s crontab file (/var/spool/cron/sam). All the scripts that Sam runs are in his ~/bin directory. The first line sets the MAILTO variable to max so that Max gets the output from commands run from Sam’s crontab file that is not redirected. The sat.job script runs every Saturday (day 6) at 2:05 AM, twice.week runs at 12:02 AM on Sunday and Thursday (days 0 and 4), and twice.day runs twice a day, every day, at 10:05 AM and 4:05 PM. $ who am i sam $ crontab -l MAILTO=max 05 02 * * 6 00 02 * * 0,4 05 10,16 * * * $HOME/bin/sat.job $HOME/bin/twice.week $HOME/bin/twice.day To add an entry to your crontab file, run the crontab utility with the –e (edit) option. Some Linux systems use a version of crontab that does not support the –e option. If the local system runs such a version, you need to make a copy of your existing crontab file, edit it, and then resubmit it, as in the example that follows. The –l (list) option displays a copy of your crontab file. $ crontab -l > newcron $ vim newcron ... $ crontab newcron 652 cut cut Selects characters or fields from input lines cut cut [options] [file-list] The cut utility selects characters or fields from lines of input and writes them to standard output. Character and field numbering start with 1. Arguments The file-list is a list of ordinary files. If you do not specify an argument or if you specify a hyphen (–) in place of a filename, cut reads from standard input. Options Under Linux, cut accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––characters=clist –c clist Selects the characters given by the column numbers in clist. The value of clist is one or more comma-separated column numbers or column ranges. A range is specified by two column numbers separated by a hyphen. A range of –n means columns 1 through n; n– means columns n through the end of the line. ––delimiter=dchar –d dchar Specifies dchar as the input field delimiter. Also specifies dchar as the output field delimiter unless you use the ––output-delimiter option. The default delimiter is a TAB character. Quote characters as necessary to protect them from shell expansion. ––fields=flist –f flist Selects the fields specified in flist. The value of flist is one or more commaseparated field numbers or field ranges. A range is specified by two field numbers separated by a hyphen. A range of –n means fields 1 through n; n– means fields n through the last field. The field delimiter is a TAB character unless you use the ––delimiter option to change it. ––output-delimiter=ochar Specifies ochar as the output field delimiter. The default delimiter is the TAB character. You can specify a different delimiter by using the ––delimiter option. Quote characters as necessary to protect them from shell expansion. ––only-delimited –s Copies only lines containing delimiters. Without this option, cut copies—but does not modify—lines that do not contain delimiters. cut 653 Notes Although limited in functionality, cut is easy to learn and use and is a good choice when columns and fields can be selected without using pattern matching. Sometimes cut is used with paste (page 784). Examples For the next two examples, assume that an ls –l command produces the following output: $ ls -l total 2944 -rwxr-xr-x -rw-rw-r--rw-rw-r--rw-rw-r--rw-rw-r--rw-rw-r--rw-rw-r-- 1 1 1 1 1 1 1 zach zach zach zach zach zach zach pubs 259 Feb 1 00:12 countout pubs 9453 Feb 4 23:17 headers pubs 1474828 Jan 14 14:15 memo pubs 1474828 Jan 14 14:33 memos_save pubs 7134 Feb 4 23:18 tmp1 pubs 4770 Feb 4 23:26 tmp2 pubs 13580 Nov 7 08:01 typescript The following command outputs the permissions of the files in the working directory. The cut utility with the –c option selects characters 2 through 10 from each input line. The characters in this range are written to standard output. $ ls -l | cut -c2-10 otal 2944 rwxr-xr-x rw-rw-r-rw-rw-r-rw-rw-r-rw-rw-r-rw-rw-r-rw-rw-r-- The next command outputs the size and name of each file in the working directory. The –f option selects the fifth and ninth fields from the input lines. The –d option tells cut to use SPACEs, not TABs, as delimiters. The tr utility (page 864) with the –s option changes sequences of more than one SPACE character into a single SPACE; otherwise, cut counts the extra SPACE characters as separate fields. $ ls -l | tr -s ' ' ' ' | cut -f5,9 -d' ' 259 countout 9453 headers 1474828 memo 1474828 memos_save 7134 tmp1 4770 tmp2 13580 typescript The last example displays a list of full names as stored in the fifth field of the /etc/passwd file. The –d option specifies that the colon character be used as the field 654 cut delimiter. Although this example works under Mac OS X, /etc/passwd does not contain information about most users; see “Open Directory” on page 926 for more information. $ cat /etc/passwd root:x:0:0:Root:/:/bin/sh sam:x:401:50:Sam the Great:/home/sam:/bin/zsh max:x:402:50:Max Wild:/home/max:/bin/bash zach:x:504:500:Zach Brill:/home/zach:/bin/tcsh hls:x:505:500:Helen Simpson:/home/hls:/bin/bash $ cut -d: -f5 /etc/passwd Root Sam the Great Max Wild Zach Brill Helen Simpson date 655 date date Displays or sets the system time and date date [options] [+format] date [options] [newdate] The date utility displays the time and date known to the system. A user working with root privileges can use date to change the system clock. Arguments The +format argument specifies the format for the output of date. The format string, which consists of field descriptors and text, follows a plus sign (+). The field descriptors are preceded by percent signs, and date replaces each one by its value in the output. Table V-10 lists some of the field descriptors. Table V-10 Selected field descriptors Descriptor Meaning %a Abbreviated weekday—Sun to Sat %A Unabbreviated weekday—Sunday to Saturday %b Abbreviated month—Jan to Dec %B Unabbreviated month—January to December %c Date and time in default format used by date %d Day of the month—01 to 31 %D Date in mm/dd/yy format %H Hour—00 to 23 %I Hour—00 to 12 %j Julian date (day of the year—001 to 366) %m Month of the year—01 to 12 %M Minutes—00 to 59 %n NEWLINE character %P AM %r Time in AM/PM notation %s Number of seconds since the beginning of January 1, 1970 %S Seconds—00 to 60 (the 60 accommodates leap seconds) or PM 656 date Table V-10 Selected field descriptors (continued) Descriptor Meaning %t TAB character %T Time in HH:MM:SS format %w Day of the week—0 to 6 (0 = Sunday) %y Last two digits of the year—00 to 99 %Y Year in four-digit format (for example, 2009) %Z Time zone (for example, PDT) By default date zero fills numeric fields. Placing an underscore (_) immediately following the percent sign (%) for a field causes date to blank fill the field. Placing a hyphen (–) following the percent sign causes date not to fill the field—that is, to left justify the field. The date utility assumes that, in a format string, any character that is not a percent sign, an underscore or a hyphen following the percent sign, or a field descriptor is ordinary text and copies it to standard output. You can use ordinary text to add punctuation to the date and to add labels (for example, you can put the word DATE: in front of the date). Surround the format argument with single quotation marks if it contains SPACE s or other characters that have a special meaning to the shell. Setting the system clock When a user working with root privileges specifies newdate, the system changes the system clock to reflect the new date. The newdate argument has the format nnddhhmm[[cc]yy][.ss] where nn is the number of the month (01–12), dd is the day of the month (01–31), hh is the hour based on a 24-hour clock (00–23), and mm is the minutes (00–59). When you change the date, you must specify at least these fields. The optional cc specifies the first two digits of the year (the value of the century minus 1), and yy specifies the last two digits of the year. You can specify yy or ccyy following mm. When you do not specify a year, date assumes that the year has not changed. You can specify the number of seconds past the start of the minute with .ss. Options Under Linux, date accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––date=datestring –d datestring Displays the date specified by datestring, not the current date. This option does not change the system clock. L date 657 ––reference=file –r file Displays the modification date and time of file in place of the current date and time. L ––utc or ––universal –u Displays or sets the time and date using Universal Coordinated Time (UTC; page 986). UTC is also called Greenwich Mean Time (GMT). Notes If you set up a locale database, date uses that database to substitute terms appropriate to your locale (page 963). Examples The first example shows how to set the date for 2:07:30 changing the year: PM on August 19 without # date 08191407.30 Fri Aug 19 14:07:30 PDT 2009 The next example shows the format argument, which causes date to display the date in a commonly used format: $ date '+Today is %h %d, %Y' Today is Aug 19, 2009 658 dd dd Converts and copies a file dd dd [arguments] The dd (device-to-device copy) utility converts and copies a file. The primary use of dd is to copy files to and from hard disk files and removable media. It can operate on hard disk partitions and create block-for-block identical disk images. Often dd can handle the transfer of information to and from other operating systems when other methods fail. Its rich set of arguments gives you precise control over the characteristics of the transfer. Arguments Under Linux, dd accepts the common options described on page 603. By default dd copies standard input to standard output. bs=n (block size) Reads and writes n bytes at a time. This argument overrides the ibs and obs arguments. cbs=n (conversion block size) When performing data conversion during the copy, converts n bytes at a time. conv=type[,type...] By applying conversion types in the order given on the command line, converts the data being copied. The types must be separated by commas with no SPACEs. The types of conversions are shown in Table V-11. Restricts to numblocks the number of blocks of input that dd copies. The size of each block is the number of bytes specified by the bs or ibs argument. count=numblocks (input block size) Reads n bytes at a time. ibs=n (input file) Reads from filename instead of from standard input. You can use a device name for filename to read from that device. if=filename (output block size) Writes n bytes at a time. obs=n of=filename (output file) Writes to filename instead of to standard output. You can use a device name for filename to write to that device. seek=numblocks Skips numblocks blocks of output before writing any output. The size of each block is the number of bytes specified by the bs or obs argument. skip=numblocks Skips numblocks blocks of input before starting to copy. The size of each block is the number of bytes specified by the bs or ibs argument. Table V-11 Conversion types type Meaning ascii Converts EBCDIC-encoded characters to ASCII, allowing you to read tapes written on IBM mainframe and similar computers. dd 659 Table V-11 Conversion types (continued) type Meaning block Each time a line of input is read (that is, a sequence of characters terminated with a NEWLINE character), outputs a block of text without the NEWLINE. Each output block has the size given by the bs or obs argument and is created by adding trailing SPACE characters to the text until it is the proper size. ebcdic Converts ASCII-encoded characters to EBCDIC, allowing you to write tapes for use on IBM mainframe and similar computers. lcase Converts uppercase letters to lowercase while copying data. noerror If a read error occurs, dd normally terminates. This conversion allows dd to continue processing data and is useful when you are trying to recover data from bad media. notrunc Does not truncate the output file before writing to it. ucase Converts lowercase letters to uppercase while copying data. unblock Performs the opposite of the block conversion. Notes Under Linux, you can use the standard multiplicative suffixes to make it easier to specify large block sizes. See Table V-1 on page 603. Under Mac OS X, you can use some of the standard multiplicative suffixes; however, OS X uses lowercase letters in place of the uppercase letters shown in the table. In addition, under OS X, dd supports b (block; multiply by 512) and w (word; multiply by the number of bytes in an integer). Examples You can use dd to create a file filled with pseudo-random bytes. $ dd if=/dev/urandom of=randfile2 bs=1 count=100 The preceding command reads from the /dev/urandom file (an interface to the kernel’s random number generator) and writes to the file named randfile. The block size is 1 and the count is 100, so randfile is 100 bytes long. For bytes that are more random, you can read from /dev/random. See the urandom and random man pages for more information. Under OS X, urandom and random behave identically. You can also use dd to make an exact copy of a disk partition. Be careful however— the following command wipes out anything that was on the /dev/sdb1 partition. # dd if=/dev/sda1 of=/dev/sdb1 Wiping a file You can use a similar technique to wipe data from a file before deleting it, making it almost impossible to recover data from the deleted file. You might want to wipe a file for security reasons; wipe a file several times for added security. 660 dd In the following example, ls shows the size of the file named secret; dd, with a block size of 1 and a count corresponding to the number of bytes in secret, then wipes the file. The conv=notrunc argument ensures that dd writes over the data in the file and not another place on the disk. $ ls -l secret -rw-rw-r-- 1 max max 2494 Feb 6 00:56 secret $ dd if=/dev/urandom of=secret bs=1 count=2494 conv=notrunc 2494+0 records in 2494+0 records out $ rm secret Copying a diskette You can use dd to make an exact copy of a floppy diskette. First copy the contents of the diskette to a file on the hard drive and then copy the file from the hard disk to a formatted diskette. This technique works regardless of what is on the floppy diskette. The next example copies a DOS-formatted diskette. (The filename for the floppy on the local system may differ from that in the example.) The mount, ls, umount sequences at the beginning and end of the example verify that the original diskette and the copy hold the same files. You can use the floppy.copy file to make multiple copies of the diskette. # mount -t msdos /dev/fd0H1440 /mnt # ls /mnt abprint.dat bti.ini setup.ins supfiles.z wbt.z adbook.z setup.exe setup.pkg telephon.z # umount /mnt # dd if=/dev/fd0 ibs=512 > floppy.copy 2880+0 records in 2880+0 records out # ls -l floppy.copy -rw-rw-r-1 max speedy 1474560 Oct 11 05:43 floppy.copy # dd if=floppy.copy bs=512 of=/dev/fd0 2880+0 records in 2880+0 records out # mount -t msdos /dev/fd0H1440 /mnt # ls /mnt abprint.dat bti.ini setup.ins supfiles.z wbt.z adbook.z setup.exe setup.pkg telephon.z # umount /mnt df 661 df df [options] [filesystem-list] The df (disk free) utility reports on the total space and the free space on each mounted device. Arguments When you call df without an argument, it reports on the free space on each of the devices mounted on the local system. The filesystem-list is an optional list of one or more pathnames that specify the filesystems you want the report to cover. This argument works on Mac OS X and some Linux systems. You can refer to a mounted filesystem by its device pathname or by the pathname of the directory it is mounted on. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––all –a Reports on filesystems with a size of 0 blocks, such as /dev/proc. Normally df does not report on these filesystems. ––block-size=sz –B sz The sz specifies the units that the report uses (the default is 1-kilobyte blocks). The sz is a multiplicative suffix from Table V-1 on page 603. See also the –h (––human-readable) and –H (––si) options. L –g (gigabyte) Displays sizes in 1-gigabyte blocks. O ––si –H Displays sizes in K (kilobyte), M (megabyte), and G (gigabyte) blocks, as is appropriate. Uses powers of 1,000. ––human-readable –h Displays sizes in K (kilobyte), M (megabyte), and G (gigabyte) blocks, as is appropriate. Uses powers of 1,024. ––inodes –i Reports the number of inodes (page 959) that are used and free instead of reporting on blocks. –k (kilobyte) Displays sizes in 1-kilobyte blocks. ––local –l Displays local filesystems only. –m (megabyte) Displays sizes in 1-megabyte blocks. O ––type=fstype –t fstype Reports information only about the filesystems of type fstype, such as DOS or NFS. Repeat this option to report on several types of filesystems. L ––exclude-type=fstype –x fstype Reports information only about the filesystems not of type fstype. L df Displays disk space usage 662 df Notes Under Mac OS X, the df utility supports the BLOCKSIZE environment variable (page 603) and ignores block sizes smaller than 512 bytes or larger than 1 gigabyte. Under Mac OS X, the count of used and free inodes (–i option) is meaningless on HFS+ filesystems. On these filesystems, new files can be created as long as free space is available in the filesystem. Examples In the following example, df displays information about all mounted filesystems on the local system: $ df Filesystem /dev/hda12 /dev/hda1 /dev/hda8 /dev/hda9 /dev/hda10 /dev/hda5 /dev/hda7 /dev/hda6 zach:/c zach:/d 1k-blocks 1517920 15522 1011928 1011928 1130540 4032092 1011928 2522048 2096160 2096450 Used Available Use% Mounted on 53264 1387548 4% / 4846 9875 33% /boot 110268 850256 11% /free1 30624 929900 3% /free2 78992 994120 7% /free3 1988080 1839188 52% /home 60 960464 0% /tmp 824084 1569848 34% /usr 1811392 284768 86% /zach_c 1935097 161353 92% /zach_d Next df is called with the –l and –h options, generating a human-readable list of local filesystems. The sizes in this listing are given in terms of megabytes and gigabytes. $ df -lh Filesystem /dev/hda12 /dev/hda1 /dev/hda8 /dev/hda9 /dev/hda10 /dev/hda5 /dev/hda7 /dev/hda6 Size 1.4G 15M 988M 988M 1.1G 3.8G 988M 2.4G Used Avail Use% Mounted on 52M 1.3G 4% / 4.7M 9.6M 33% /boot 108M 830M 11% /free1 30M 908M 3% /free2 77M 971M 7% /free3 1.9G 1.8G 52% /home 60k 938M 0% /tmp 805M 1.5G 34% /usr The next example, which runs under Linux only, displays information about the /free2 partition in megabyte units: $ df -BM /free2 Filesystem /dev/hda9 1M-blocks 988 Used Available Use% Mounted on 30 908 3% /free2 The final example, which runs under Linux only, displays information about NFS filesystems in human-readable terms: $ df -ht nfs Filesystem zach:/c zach:/d Size 2.0G 2.0G Used Avail Use% Mounted on 1.7G 278M 86% /zach_c 1.8G 157M 92% /zach_d diff 663 diff diff [options] file1 file2 diff [options] file1 directory diff [options] directory file2 diff [options] directory1 directory2 The diff utility displays line-by-line differences between two text files. By default diff displays the differences as instructions, which you can then use to edit one of the files to make it the same as the other. Arguments The file1 and file2 are pathnames of ordinary text files that diff works on. When the directory argument is used in place of file2, diff looks for a file in directory with the same name as file1. It works similarly when directory replaces file1. When you specify two directory arguments, diff compares the files in directory1 with the files that have the same simple filenames in directory2. Options The diff utility accepts the common options described on page 603, with one exception: When one of the arguments is a directory and the other is an ordinary file, you cannot compare to standard input. The Mac OS X version of diff accepts long options tip Options for diff preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––ignore-blank-lines –B Ignores differences that involve only blank lines. ––ignore-space-change –b Ignores whitespace (SPACE s and TAB s) at the ends of lines and considers other strings of whitespace to be equal. ––context[=lines] –C [lines] Displays the sections of the two files that differ, including lines lines (the default is 3) around each line that differs to show the context. Each line in file1 that is missing from file2 is preceded by a hyphen (–); each extra line in file2 is preceded by a plus sign (+); and lines that have different versions in the two files are preceded by an exclamation point (!). When lines that differ are within lines lines of each other, they are grouped together in the output. ––ed –e Creates and sends to standard output a script for the ed editor, which will edit file1 to make it the same as file2. You must add w (Write) and q (Quit) instructions to the end of the script if you plan to redirect input to ed from the script. When you use ––ed, diff displays the changes in reverse order: Changes to the end of the file are listed before changes to the top, preventing early changes from affecting later changes when the script is used as input to ed. For example, if a line near the top were deleted, subsequent line numbers in the script would be wrong. diff Displays the differences between two text files 664 diff –i Ignores differences in case when comparing files. ––ignore-case ––new-file –N When comparing directories, when a file is present in one of the directories only, considers it to be present and empty in the other directory. ––show-c-function –p Shows which C function, bash control structure, Perl subroutine, and so forth each change affects. ––brief –q Does not display the differences between lines in the files. Instead, diff reports only that the files differ. ––recursive –r When using diff to compare the files in two directories, causes the comparisons to descend through the directory hierarchies. ––unified[=lines] –U lines Uses the easier-to-read unified output format. See the discussion of diff on page 54 for more detail and an example. The lines argument is the number of lines of context; the default is three. ––ignore-all-space –w (whitespace) Ignores whitespace when comparing lines. ––width=n –W n Sets the width of the columns that diff uses to display the output to n characters. This option is useful with the ––side-by-side option. The sdiff utility (see the “Notes” section) uses a lowercase w to perform the same function: –w n. ––side-by-side Notes –y Displays the output in a side-by-side format. This option generates the same output as sdiff. Use the ––width=columns option with this option. The sdiff utility is similar to diff but its output may be easier to read. The diff ––sideby-side option produces the same output as sdiff. See the “Examples” section and refer to the diff and sdiff man and info pages for more information. Use the diff3 utility to compare three files. Use cmp (page 634) to compare nontext (binary) files. Discussion When you use diff without any options, it produces a series of lines containing Add (a), Delete (d), and Change (c) instructions. Each of these lines is followed by the lines from the file you need to add to, delete from, or change, respectively, to make the files the same. A less than symbol () precedes lines from file2. The diff output appears in the format shown in Table V-12. A pair of line numbers separated by a comma represents a range of lines; a single line number represents a single line. The diff utility assumes you will convert file1 to file2. The line numbers to the left of each of the a, c, or d instructions always pertain to file1; the line numbers to the diff 665 right of the instructions apply to file2. To convert file2 to file1, run diff again, reversing the order of the arguments. Table V-12 Examples diff output Instruction Meaning (to change file1 to file2) line1 a line2,line3 > lines from file2 Append lines line2 through line3 from file2 after line1 in file1. line1,line2 d line3 < lines from file1 Delete line1 through line2 from file1. line1,line2 c line3,line4 < lines from file1 ––– > lines from file 2 Change line1 through line2 in file1 to line3 through line 4 from file2. The first example shows how diff displays the differences between two short, similar files: $ cat m aaaaa bbbbb ccccc $ cat n aaaaa ccccc $ diff m n 2d1 < bbbbb The difference between files m and n is that the second line of file m (bbbbb) is missing from file n. The first line that diff displays (2d1) indicates that you need to delete the second line from file1 (m) to make it the same as file2 (n). The next line diff displays starts with a less than symbol (rrrrr In the preceding example, diff issues the instruction 2a3 to indicate you must append a line to file m, after line 2, to make it the same as file p. The second line diff displays indicates the line is from file p (the line begins with >, indicating file2). In this example, you need the information on this line; the appended line must contain the text rrrrr. The next example uses file m again, this time with file r, to show how diff indicates a line that needs to be changed: $ cat r aaaaa -q ccccc $ diff m r 2c2 < bbbbb --> -q The difference between the two files appears in line 2: File m contains bbbbb, and file r contains –q. The diff utility displays 2c2 to indicate that you need to change line 2. After indicating a change is needed, diff shows you must change line 2 in file m (bbbbb) to line 2 in file r (–q) to make the files the same. The three hyphens indicate the end of the text in file m that needs to be changed and the beginning of the text in file r that is to replace it. Comparing the same files using the side-by-side and width options (–y and –W) yields an easier-to-read result. The pipe symbol (|) indicates the line on one side must replace the line on the other side to make the files the same: $ diff -y -W 30 m r aaaaa aaaaa bbbbb | -q ccccc ccccc The next examples compare the two files q and v: $ cat q Monday Tuesday Wednesday Thursday Saturday Sunday $ cat v Monday Wednesday Thursday Thursday Friday Saturday Sundae diff 667 Running in side-by-side mode diff shows Tuesday is missing from file v, there is only one Thursday in file q (there are two in file v), and Friday is missing from file q. The last line is Sunday in file q and Sundae in file v: diff indicates these lines are different. You can change file q to be the same as file v by removing Tuesday, adding one Thursday and Friday, and substituting Sundae from file v for Sunday from file q. Alternatively, you can change file v to be the same as file q by adding Tuesday, removing one Thursday and Friday, and substituting Sunday from file q for Sundae from file v. $ diff -y -W 30 q v Monday Monday Tuesday < Wednesday Wednesday Thursday Thursday > Thursday > Friday Saturday Saturday Sunday | Sundae Context diff With the ––context option (called a context diff), diff displays output that tells you how to turn the first file into the second file. The top two lines identify the files and show that q is represented by asterisks, whereas v is represented by hyphens. Following a row of asterisks that indicates the beginning of a hunk of text is a row of asterisks with the numbers 1,6 in the middle. This line indicates that the instructions in the first section tell you what to remove from or change in file q—namely, lines 1 through 6 (that is, all the lines of file q; in a longer file it would mark the first hunk). The hyphen on the second subsequent line indicates you need to remove the line with Tuesday. The line with an exclamation point indicates you need to replace the line with Sunday with the corresponding line from file v. The row of hyphens with the numbers 1,7 in the middle indicates that the next section tells you which lines from file v—lines 1 through 7—you need to add or change in file q. You need to add a second line with Thursday and a line with Friday, and you need to change Sunday in file q to Sundae (from file v). $ diff --context q v *** q Mon Aug 24 18:26:45 2009 --- v Mon Aug 24 18:27:55 2009 *************** *** 1,6 **** Monday - Tuesday Wednesday Thursday Saturday ! Sunday --- 1,7 ---Monday Wednesday Thursday + Thursday + Friday Saturday ! Sundae 668 diskutil O diskutil O diskutil O Checks, modifies, and repairs local volumes diskutil action [arguments] The diskutil utility mounts, unmounts, and displays information about disks and partitions (volumes). It can also format and repair filesystems and divide a disk into partitions. Arguments The action specifies what diskutil is to do. Table V-13 lists common actions along with the argument each takes. Table V-13 diskutil actions Action Argument Description eraseVolume type name device Reformats device using the format type and the label name. The name specifies the name of the volume; alphanumeric names are the easiest to work with. The filesystem type is typically HFS+, but can also be UFS or MS-DOS. You can specify additional options as part of the type. For example, a FAT32 filesystem (as used in Windows 98 and later) would have a type of MS-DOS FAT32. A journaled, case-sensitive, HFS+ filesystem would have a type of Case-sensitive Journaled HFS+. info device Displays information about device. Does not require ownership of device. list [device] Lists partitions on device. Without device lists partitions on all devices. Does not require ownership of device. mount device Mounts device. mountDisk device Mounts all devices on the disk containing device. reformat device Reformats device using its current name and format. repairVolume device Repairs the filesystem on device. unmount device Unmounts device. unmountDisk device Unmounts all devices on the disk containing device. verifyVolume device Verifies the filesystem on device. Does not require ownership of device. diskutil O Notes 669 The diskutil utility provides access to the Disk Management framework, the support code used by the Disk Utility application. It allows some choices that are not supported from the graphical interface. You must own device, or be working with root privileges, when you specify an action that modifies or changes the state of a volume. fsck disktool Examples The diskutil verifyVolume and repairVolume actions are analogous to the fsck utility on Linux systems. Under OS X, the fsck utility is deprecated except when the system is in single-user mode. Some of the functions performed by diskutil were handled by disktool in the past. The first example displays a list of disk devices and volumes available on the local system: $ diskutil list /dev/disk0 #: type 0: Apple_partition_scheme 1: Apple_partition_map 2: Apple_HFS 3: Apple_HFS /dev/disk1 #: type 0: Apple_partition_scheme 1: Apple_partition_map 2: Apple_Driver43 3: Apple_Driver43 4: Apple_Driver_ATA 5: Apple_Driver_ATA 6: Apple_FWDriver 7: Apple_Driver_IOKit 8: Apple_Patches 9: Apple_HFS 10: Apple_HFS name Eva01 Users name Spare House size *152.7 GB 31.5 KB 30.7 GB 121.7 GB identifier disk0 disk0s1 disk0s3 disk0s5 size *232.9 GB 31.5 KB 28.0 KB 28.0 KB 28.0 KB 256.0 KB 256.0 KB 256.0 KB 48.8 GB 184.1 GB identifier disk1 disk1s1 disk1s2 disk1s3 disk1s4 disk1s5 disk1s6 disk1s7 disk1s8 disk1s9 disk1s10 The next example displays information about one of the mounted volumes: $ diskutil info disk1s9 Device Node: /dev/disk1s9 Device Identifier: disk1s9 Mount Point: /Volumes/Spare Volume Name: Spare File System: Owners: Partition Type: Bootable: Media Type: HFS+ Enabled Apple_HFS Is bootable Generic 670 diskutil O Protocol: SMART Status: UUID: FireWire Not Supported C77BB3DC-EFBB-30B0-B191-DE7E01D8A563 Total Size: Free Space: 48.8 GB 48.8 GB Read Only: Ejectable: No Yes The next example formats the partition at /dev/disk1s8 as an HFS+ Extended (HFSX) filesystem and labels it Spare2. This command erases all data on the partition: # diskutil eraseVolume 'Case-sensitive HFS+' Spare2 disk1s8 Started erase on disk disk1s10 Erasing Mounting Disk Finished erase on disk disk1s10 The final example shows the output of a successful verifyVolume operation: $ diskutil verifyVolume disk1s9 Started verify/repair on volume disk1s9 Spare Checking HFS Plus volume. Checking Extents Overflow file. Checking Catalog file. Checking Catalog hierarchy. Checking volume bitmap. Checking volume information. The volume Spare appears to be OK. Mounting Disk Verify/repair finished on volume disk1s9 Spare 671 ditto O Copies files and creates and unpacks archives ditto [options] source-file destination-file ditto [options] source-file-list destination-directory ditto –c [options] source-directory destination-archive ditto –x [options] source-archive-list destination-directory The ditto utility copies files and their ownership, timestamps, and other attributes, including extended attributes (page 928). It can copy to and from cpio and zip archive files, as well as copy ordinary files and directories. Arguments The source-file is the pathname of the file that ditto is to make a copy of. The destination-file is the pathname that ditto assigns to the resulting copy of the file. The source-file-list specifies one or more pathnames of files and directories that ditto makes copies of. The destination-directory is the pathname of the directory that ditto copies the files and directories into. When you specify a destination-directory, ditto gives each of the copied files the same simple filename as its source-file. The source-directory is a single directory that ditto copies into the destination-archive. The resulting archive holds copies of the contents of source-directory, but not the directory itself. The source-archive-list specifies one or more pathnames of archives that ditto extracts into destination-directory. Using a hyphen (–) in place of a filename or a directory name causes ditto to read from standard input or write to standard output instead of reading from or writing to that file or directory. Options You cannot use the –c and –x options together. –c (create archive) Creates an archive file. ––help Displays a help message. –k (pkzip) Uses the zip format, instead of the default cpio (page 644) format, to create or extract archives. For more information on zip, see the tip on page 62. ––norsrc (no resource) Ignores extended attributes. This option causes ditto to copy only data forks (the default behavior under Mac OS X 10.3 and earlier). ––rsrc (resource) Copies extended attributes, including resource forks (the default behavior under Mac OS X 10.4 and later). Also –rsrc and –rsrcFork. –V (very verbose) Sends a line to standard error for each file, symbolic link, and device node ditto copies. –v (verbose) Sends a line to standard error for each directory ditto copies. ditto O ditto O 672 ditto O –X (exclude) Prevents ditto from searching directories in filesystems other than the filesystems that hold the files it was explicitly told to copy. –x (extract archive) Extracts files from an archive file. –z (compress) Uses gzip (page 724) or gunzip to compress or decompress cpio archives. Notes The ditto utility does not copy the locked attribute flag (page 931). The utility also does not copy ACLs. By default ditto creates and reads archives (page 941) in the cpio (page 644) format. The ditto utility cannot list the contents of archive files; it can only create or extract files from archives. Use pax or cpio to list the contents of cpio archives, and use unzip with the –l option to list the contents of zip files. Examples The following examples show three ways to back up a user’s home directory, including extended attributes (except as mentioned in “Notes”), preserving timestamps and permissions. The first example copies Zach’s home directory to the volume (filesystem) named Backups; the copy is a new directory named zach.0228: $ ditto /Users/zach /Volumes/Backups/zach.0228 The next example copies Zach’s home directory into a single cpio-format archive file on the volume named Backups: $ ditto -c /Users/zach /Volumes/Backups/zach.0228.cpio The next example copies Zach’s home directory into a zip archive: $ ditto -c -k /Users/zach /Volumes/Backups/zach.0228.zip Each of the next three examples restores the corresponding backup archive into Zach’s home directory, overwriting any files that are already there: $ ditto /Volumes/Backups/zach.0228 /Users/zach $ ditto -x /Volumes/Backups/zach.0228.cpio /Users/zach $ ditto -x -k /Volumes/Backups/zach.0228.zip /Users/zach The following example copies the Scripts directory to a directory named ScriptsBackups on the remote host bravo. It uses an argument of a hyphen in place of source-directory locally to write to standard output and in place of destinationdirectory on the remote system to read from standard input: $ ditto -c Scripts - | ssh bravo ditto -x - ScriptsBackups The final example copies the local startup disk (the root filesystem) to the volume named Backups.root. Because some of the files can be read only by root, the script must be run by a user with root privileges. The –X option keeps ditto from trying to copy other volumes (filesystems) that are mounted under /. # ditto -X / /Volumes/Backups.root dmesg 673 Displays kernel messages dmesg [options] The dmesg utility displays messages stored in the kernel ring buffer. Options –c Clears the kernel ring buffer after running dmesg. L –M core The core is the name of the (core dump) file to process (defaults to /dev/kmem). O –N kernel The kernel is the pathname of a kernel file (defaults to /mach). If you are displaying information about a core dump, kernel should be the kernel that was running at the time the core file was created. O Discussion When the system boots, the kernel fills its ring buffer with messages regarding hardware and module initialization. Messages in the kernel ring buffer are often useful for diagnosing system problems. Notes Under Mac OS X, you must run this utility while working with root privileges. As a ring buffer, the kernel message buffer keeps the most recent messages it receives, discarding the oldest messages once it fills up. To save a list of kernel boot messages, give the following command immediately after booting the system and logging in: $ dmesg > dmesg.boot This command saves the kernel messages in the dmesg.boot file. This list can be educational and quite useful when you are having a problem with the boot process. Under most Linux systems, after the system boots, the system records much of the same information as dmesg displays in /var/log/messages or a similar file. Examples The following command displays kernel messages in the ring buffer with the string serial in them, regardless of case: $ dmesg | grep -i serial Apple16X50PCI2: Identified 4 Serial channels at PCI SLOT-2 Bus=5 Dev=2 Func=0 dmesg dmesg 674 dscl O dscl O dscl O Displays and manages Directory Service information dscl [options] [datasource [command]] The dscl (Directory Service command line) utility enables you to work with Directory Service directory nodes. When you call dscl without arguments, it runs interactively. Arguments The datasource is a node name or a Mac OS X Server host specified by a hostname or IP address. A period (.) specifies the local domain. Options –p (prompt) Prompts for a password as needed. –q (quiet) Does not prompt. –u user Authenticates as user. Commands Refer to the “Notes” section for definitions of some of the terms used here. The hyphen (–) before a command is optional. –list path [key] (also –ls) Lists subdirectories in path, one per line. If you specify key, this command lists subdirectories that match key. –read [path [key]] (also –cat and .) Displays a directory, one property per line. –readall [path [key]] Displays properties with a given key. –search path key value Displays properties where key matches value. Notes When discussing Directory Service, the term directory refers to a collection of data (a database), not to a filesystem directory. Each directory holds one or more properties. Each property comprises a key/value pair, where there may be more than one value for a given key. In general, dscl displays a property with the key first, followed by a colon, and then the value. If there is more than one value, the values are separated by SPACEs. If a value contains SPACEs, dscl displays the value on the line following the key. Under Mac OS X and Mac OS X Server, Open Directory stores information for the local system in key/value-formatted XML files in the /var/db/dslocal directory hierarchy. dscl O 675 The dscl utility is the command-line equivalent of NetInfo Manager (available on versions of Mac OS X prior to 10.5) or of Workgroup Manager on Mac OS X Server. Examples The dscl –list command displays a list of top-level directories when you specify a path of /: $ dscl . -list / AFPServer AFPUserAliases Aliases AppleMetaRecord Augments Automount ... SharePoints SMBServer Users WebServer The period as the first argument to dscl specifies the local domain as the data source. The next command displays a list of Users directories: $ dscl . -list /Users _amavisd _appowner _appserver _ard ... _www _xgridagent _xgridcontroller daemon max nobody root You can use the dscl –read command to display information about a specific user: $ dscl . -read /Users/root AppleMetaNodeLocation: /Local/Default GeneratedUID: FFFFEEEE-DDDD-CCCC-BBBB-AAAA00000000 NFSHomeDirectory: /var/root Password: * PrimaryGroupID: 0 RealName: System Administrator RecordName: root RecordType: dsRecTypeStandard:Users UniqueID: 0 UserShell: /bin/sh 676 dscl O The following dscl –readall command lists all usernames and user IDs on the local system. The command looks for the RecordName and UniqueID keys in the /Users directory and displays the associated values. The dscl utility separates multiple values with SPACEs. See page 926 for an example of a shell script that calls dscl with the –readall command. $ dscl . -readall /Users RecordName UniqueID RecordName: _amavisd amavisd UniqueID: 83 RecordName: _appowner appowner UniqueID: 87 ... RecordName: daemon UniqueID: 1 RecordName: sam UniqueID: 501 RecordName: nobody UniqueID: -2 RecordName: root UniqueID: 0 The following example uses the dscl –search command to display all properties where the key RecordName equals sam: $ dscl . -search / RecordName sam Users/sam RecordName = (sam) du 677 du du [options] [path-list] The du (disk usage) utility reports how much disk space is occupied by a directory hierarchy or a file. By default du displays the number of 1,024-byte blocks occupied by the directory hierarchy or file. Arguments Without any arguments, du displays information about the working directory and its subdirectories. The path-list specifies the directories and files you want information on. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. Without any options, du displays the total storage used for each argument in pathlist. For directories, du displays this total after recursively listing the totals for each subdirectory. ––all –a Displays the space used by all ordinary files along with the total for each directory. ––block-size=sz –B sz The sz argument specifies the units the report uses. It is a multiplicative suffix from Table V-1 on page 603. See also the –H (––si) and –h (––human-readable) options. L ––total –c Displays a grand total at the end of the output. ––dereference-args –D (partial dereference) For each file that is a symbolic link, reports on the file the link points to, not the symbolic link itself. This option affects files specified on the command line; it does not affect files found while descending a directory hierarchy. This option treats files that are not symbolic links normally. L –d depth Displays information for subdirectories to a level of depth directories. O ––si –H (human readable) Displays sizes in K (kilobyte), M (megabyte), and G (gigabyte) blocks, as appropriate. Uses powers of 1,000. In the future, the –H option will change to be the equivalent of –D. L –H (partial dereference) For each file that is a symbolic link, reports on the file the link points to, not the symbolic link itself. This option affects files specified on the command line; it does not affect files found while descending a directory hierarchy. This option treats files that are not symbolic links normally. See page 623 for an example of the use of the –H versus –L options. O du Displays information on disk usage by directory hierarchy and/or file 678 du ––human-readable –h Displays sizes in K (kilobyte), M (megabyte), and G (gigabyte) blocks, as appropriate. Uses powers of 1,024. –k Displays sizes in 1-kilobyte blocks. ––dereference –L For each file that is a symbolic link, reports on the file the link points to, not the symbolic link itself. This option affects all files and treats files that are not symbolic links normally. The default is –P (––no-dereference). See page 623 for an example of the use of the –H versus –L options. –m Displays sizes in 1-megabyte blocks. ––no-dereference –P For each file that is a symbolic link, reports on the symbolic link, not the file the link points to. This option affects all files and treats files that are not symbolic links normally. This behavior is the default. See page 625 for an example of the use of the –P option. ––summarize –s Displays only the total size for each directory or file you specify on the command line; subdirectory totals are not displayed. ––one-file-system –x Reports only on files and directories on the same filesystem as that of the argument being processed. Examples In the first example, du displays size information about subdirectories in the working directory. The last line contains the grand total for the working directory and its subdirectories. $ du 26 4 47 4 12 105 ./Postscript ./RCS ./XIcon ./Printer/RCS ./Printer . The total (105) is the number of blocks occupied by all plain files and directories under the working directory. All files are counted, even though du displays only the sizes of directories. If you do not have read permission for a file or directory that du encounters, du sends a warning to standard error and skips that file or directory. Next, using the –s (summarize) option, du displays the total for each of the directories in /usr but does not display information for subdirectories: $ du -s 4 260292 10052 7772 1720468 105240 0 /usr/* /usr/X11R6 /usr/bin /usr/games /usr/include /usr/lib /usr/lib32 /usr/lib64 du 679 du: cannot read directory `/usr/local/lost+found': Permission denied ... 130696 /usr/src Add to the previous example the –c (total) option and du displays the same listing with a grand total at the end: $ du -sc /usr/* 4 /usr/X11R6 260292 /usr/bin ... 130696 /usr/src 3931436 total The following example uses the –s (summarize), –h (human-readable), and –c (total) options: $ du -shc /usr/* 4.0K /usr/X11R6 255M /usr/bin 9.9M /usr/games 7.6M /usr/include 1.7G /usr/lib 103M /usr/lib32 ... 128M /usr/src 3.8G total The final example displays, in human-readable format, the total size of all files the user can read in the /usr filesystem. Redirecting standard error to /dev/null discards all warnings about files and directories that are unreadable. $ du -hs /usr 2>/dev/null 3.8G /usr 680 echo echo echo Displays a message echo [options] message The echo utility copies its arguments, followed by a NEWLINE, to standard output. Both the Bourne Again and TC Shells have their own echo builtin that works similarly to the echo utility. Arguments The message consists of one or more arguments, which can include quoted strings, ambiguous file references, and shell variables. A SPACE separates each argument from the next. The shell recognizes unquoted special characters in the arguments. For example, the shell expands an asterisk into a list of filenames in the working directory. Options You can configure the tcsh echo builtin to treat backslash escape sequences and the –n option in different ways. Refer to echo_style in the tcsh man page. The typical tcsh configuration recognizes the –n option, enables backslash escape sequences, and ignores the –E and –e options. –E Suppresses the interpretation of backslash escape sequences such as . Available with the bash builtin version of echo only. –e Enables the interpretation of backslash escape sequences such as . Available with the bash builtin version of echo only. Gives a short summary of how to use echo. The summary includes a list of the backslash escape sequences interpreted by echo. This option works only with the echo utility; it does not work with the echo builtins. L ––help –n Suppresses the NEWLINE terminating the message. Notes Suppressing the interpretation of backslash escape sequences is the default behavior of the bash builtin version of echo and of the echo utility. You can use echo to send messages to the screen from a shell script. See page 138 for a discussion of how to use echo to display filenames using wildcard characters. The echo utility and builtins provide an escape notation to represent certain nonprinting characters in message (Table V-14). You must use the –e option for these backslash escape sequences to work with the echo utility and the bash echo builtin. Typically you do not need the –e option with the tcsh echo builtin. Table V-14 Backslash escape sequences Sequence Meaning \a Bell \c Suppress trailing NEWLINE echo 681 Table V-14 Examples Backslash escape sequences (continued) Sequence Meaning NEWLINE \t HORIZONTAL TAB \v VERTICAL TAB \\ BACKSLASH Following are some echo commands. These commands will work with the echo utility (/bin/echo) and the bash and tcsh echo builtins, except for the last, which may not need the –e option under tcsh. $ echo "This command displays a string." This command displays a string. $ echo -n "This displayed string is not followed by a NEWLINE." This displayed string is not followed by a NEWLINE.$ echo hi hi $ echo -e "This message contains\v a vertical tab." This message contains a vertical tab. $ The following examples contain messages with the backslash escape sequence \c. In the first example, the shell processes the arguments before calling echo. When the shell sees the \c, it replaces the \c with the character c. The next three examples show how to quote the \c so that the shell passes it to echo, which then does not append a NEWLINE to the end of the message. The first four examples are run under bash and require the –e option. The final example runs under tcsh, which may not need this option. $ echo -e There is a newline after this line.\c There is a newline after this line.c $ echo -e 'There is no newline after this line.\c' There is no newline after this line.$ $ echo -e "There is no newline after this line.\c" There is no newline after this line.$ $ echo -e There is no newline after this line.\\c There is no newline after this line.$ $ tcsh tcsh $ echo -e 'There is no newline after this line.\c' There is no newline after this line.$ You can use the –n option in place of –e and \c. 682 expr expr Evaluates an expression expr expr expression The expr utility evaluates an expression and sends the result to standard output. It evaluates character strings that represent either numeric or nonnumeric values. Operators are used with the strings to form expressions. Arguments The expression is composed of strings interspersed with operators. Each string and operator constitute a distinct argument that you must separate from other arguments with a SPACE. You must quote operators that have special meanings to the shell (for example, the multiplication operator, *). The following list of expr operators is given in order of decreasing precedence. Each operator within a group of operators has the same precedence. You can change the order of evaluation by using parentheses. : (comparison) Compares two strings, starting with the first character in each string and ending with the last character in the second string. The second string is a regular expression with an implied caret (^) as its first character. If expr finds a match, it displays the number of characters in the second string. If expr does not find a match, it displays a zero. * (multiplication) (division) (remainder) Work only on strings that contain the numerals 0 through 9 and optionally a leading minus sign. Convert strings to integer numbers, perform the specified arithmetic operation on numbers, and convert the result back to a string before sending it to standard output. + – (addition) (subtraction) Function in the same manner as the preceding group of operators. / % < = > (less than) (less than or equal to) (equal to) (not equal to) (greater than or equal to) (greater than) Relational operators work on both numeric and nonnumeric arguments. If one or both of the arguments are nonnumeric, the comparison is nonnumeric, using the machine collating sequence (typically ASCII). If both arguments are expr 683 numeric, the comparison is numeric. The expr utility displays a 1 (one) if the comparison is true and a 0 (zero) if it is false. Notes & (AND) Evaluates both of its arguments. If neither is 0 or a null string, expr displays the value of the first argument. Otherwise, it displays a 0 (zero). You must quote this operator. | (OR) Evaluates the first argument. If it is neither 0 nor a null string, expr displays the value of the first argument. Otherwise, it displays the value of the second argument. You must quote this operator. The expr utility returns an exit status of 0 (zero) if the expression evaluates to anything other than a null string or the number 0, a status of 1 if the expression is null or 0, and a status of 2 if the expression is invalid. Although expr and this discussion distinguish between numeric and nonnumeric arguments, all arguments to expr are nonnumeric (character strings). When applicable, expr attempts to convert an argument to a number (for example, when using the + operator). If a string contains characters other than 0 through 9 and optionally a leading minus sign, expr cannot convert it. Specifically, if a string contains a plus sign or a decimal point, expr considers it to be nonnumeric. If both arguments are numeric, the comparison is numeric. If one is nonnumeric, the comparison is lexicographic. Examples In the following examples, expr evaluates constants. You can also use expr to evaluate variables in a shell script. The fourth command displays an error message because of the illegal decimal point in 5.3: $ expr 17 + 40 57 $ expr 10 - 24 -14 $ expr -17 + 20 3 $ expr 5.3 + 4 expr: non-numeric argument The multiplication (*), division (/), and remainder (%) operators provide additional arithmetic power. You must quote the multiplication operator (precede it with a backslash) so that the shell will not treat it as a special character (an ambiguous file reference). You cannot put quotation marks around the entire expression because each string and operator must be a separate argument. $ expr 5 * 4 20 $ expr 21 / 7 3 $ expr 23 % 7 2 684 expr The next two examples show how parentheses change the order of evaluation. You must quote each parenthesis and surround the backslash/parenthesis combination with SPACEs: $ expr 2 * 3 + 4 10 $ expr 2 * \(3 + 4 \) 14 You can use relational operators to determine the relationship between numeric or nonnumeric arguments. The following commands compare two strings to see if they are equal; expr displays a 0 when the relationship is false and a 1 when it is true. $ expr fred == sam 0 $ expr sam == sam 1 In the following examples, the relational operators, which must be quoted, establish order between numeric or nonnumeric arguments. Again, if a relationship is true, expr displays a 1. $ expr fred \> sam 0 $ expr fred \< sam 1 $ expr 5 \< 7 1 The next command compares 5 with m. When one of the arguments expr is comparing with a relational operator is nonnumeric, expr considers the other to be nonnumeric. In this case, because m is nonnumeric, expr treats 5 as a nonnumeric argument. The comparison is between the ASCII (on many systems) values of m and 5. The ASCII value of m is 109 and that of 5 is 53, so expr evaluates the relationship as true. $ expr 5 \< m 1 In the next example, the matching operator determines that the four characters in the second string match the first four characters in the first string. The expr utility displays the number of matching characters (4). $ expr abcdefghijkl : abcd 4 The & operator displays a 0 if one or both of its arguments are 0 or a null string; otherwise, it displays the first argument: $ expr '' \& book 0 $ expr magazine \& book magazine expr 685 $ expr 5 \& 0 0 $ expr 5 \& 6 5 The | operator displays the first argument if it is not 0 or a null string; otherwise, it displays the second argument: $ expr '' \| book book $ expr magazine \| book magazine $ expr 5 \| 0 5 $ expr 0 \| 5 5 $ expr 5 \| 6 5 686 file file Displays the classification of a file file file [option] file-list The file utility classifies files according to their contents. Arguments The file-list is a list of the pathnames of one or more files that file classifies. You can specify any kind of file, including ordinary, directory, and special files, in the file-list. Options The Mac OS X version of file accepts long options tip Options for file preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––files-from=file –f file Takes the names of files to be examined from file rather than from file-list on the command line. The names of the files must be listed one per line in file. ––no-dereference –h For each file that is a symbolic link, reports on the symbolic link, not the file the link points to. This option treats files that are not symbolic links normally. This behavior is the default on systems where the environment variable POSIXLY_CORRECT is not defined (typical). ––help Displays a help message. ––mime –I Displays MIME (page 966) type strings. O ––mime –i Displays MIME (page 966) type strings. L ––dereference –L For each file that is a symbolic link, reports on the file the link points to, not the symbolic link itself. This option treats files that are not symbolic links normally. This behavior is the default on systems where the environment variable POSIXLY_CORRECT is defined. ––uncompress –z (zip) Attempts to classify files within a compressed file. Notes The file utility can classify more than 5,000 file types. Some of the more common file types found on Linux systems, as displayed by file, follow: archive ascii text c program text commands text core file cpio archive data file 687 directory ELF 32-bit LSB executable empty English text executable The file utility uses a maximum of three tests in its attempt to classify a file: filesystem, magic number, and language tests. When file identifies the type of a file, it ceases testing. The filesystem test examines the return from a stat() system call to see whether the file is empty or a special file. The magic number (page 964) test looks for data in particular fixed formats near the beginning of the file. The language test, if needed, determines whether the file is a text file, which encoding it uses, and which language it is written in. Refer to the file man page for a more detailed description of how file works. The results of file are not always correct. Examples Some examples of file identification follow: /etc/Muttrc: ASCII English text /etc/Muttrc.d: directory /etc/adjtime: ASCII text /etc/aliases.db: Berkeley DB (Hash, version 9, native byte-order) /etc/at.deny: writable, regular file, no read permission

/etc/bash_completion: ASCII Pascal program text /etc/blkid.tab.old: Non-ISO extended-ASCII text, with CR, LF line terminators /etc/brltty.conf: UTF-8 Unicode C++ program text /etc/chatscripts: setgid directory /etc/magic: magic text file for file(1) cmd /etc/motd: symbolic link to `/var/run/motd' /etc/qemu-ifup: POSIX shell script text executable /usr/bin/4xml: a python script text executable /usr/bin/Xorg: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), for GNU/Linux 2.6.8, dynamically linked (uses shared libs), stripped /usr/bin/debconf: perl script text executable /usr/bin/locate: symbolic link to `/etc/alternatives/locate' /usr/share/man/man7/hier.7.gz: gzip compressed data, was "hier.7", from Unix, last modified: Thu Jan 31 03:06:00 2008, max compression 688 find find Finds files based on criteria find [directory-list] [option] [expression] The find utility selects files that are located in specified directory hierarchies and that meet specified criteria. Arguments The directory-list specifies the directory hierarchies that find is to search. When you do not specify a directory-list, find searches the working directory hierarchy. The option controls whether find dereferences symbolic links as it descends directory hierarchies. By default find does not dereference symbolic links (it works with the symbolic link, not the file the link points to). Under Mac OS X, you can use the –x option to prevent find from searching directories in filesystems other than those specified in directory-list. Under Linux, the –xdev criterion performs the same function. The expression contains criteria, as described in the "Criteria" section. The find utility tests each of the files in each of the directories in the directory-list to see whether it meets the criteria described by the expression. When you do not specify an expression, the expression defaults to –print. A SPACE separating two criteria is a Boolean AND operator: The file must meet both criteria to be selected. A –or or –o separating the criteria is a Boolean OR operator: The file must meet one or the other (or both) of the criteria to be selected. You can negate any criterion by preceding it with an exclamation point. The find utility evaluates criteria from left to right unless you group them using parentheses. Within the expression you must quote special characters so the shell does not interpret them but rather passes them to find. Special characters that are frequently used with find include parentheses, brackets, question marks, and asterisks. Each element within the expression is a separate argument. You must separate arguments from each other with SPACEs. A SPACE must appear on both sides of each parenthesis, exclamation point, criterion, or other element. Options –H (partial dereference) For each file that is a symbolic link, works with the file the link points to, not the symbolic link itself. This option affects files specified on the command line; it does not affect files found while descending a directory hierarchy. This option treats files that are not symbolic links normally. See page 623 for an example of the use of the –H versus –L options. –L (dereference) For each file that is a symbolic link, works with the file the link points to, not the symbolic link itself. This option affects all files and treats find 689 files that are not symbolic links normally. See page 623 for an example of the use of the –H versus –L options. –P (no dereference) For each file that is a symbolic link, works with the symbolic link, not the file the link points to. This option affects all files and treats files that are not symbolic links normally. This behavior is the default. See page 625 for an example of the use of the –P option. –x Causes find not to search directories in filesystems other than the one(s) specified by directory-list. Under Linux, use the –xdev criterion. O Criteria You can use the following criteria within the expression. As used in this list, ±n is a decimal integer that can be expressed as +n (more than n), –n (fewer than n), or n (exactly n). –anewer filename (accessed newer) The file being evaluated meets this criterion if it was accessed more recently than filename. –atime ±n (access time) The file being evaluated meets this criterion if it was last accessed ±n days ago. When you use this option, find changes the access times of directories it searches. –depth The file being evaluated always meets this action criterion. It causes find to take action on entries in a directory before it acts on the directory itself. When you use find to send files to the cpio utility, the –depth criterion enables cpio to preserve the modification times of directories when you restore files (assuming you use the ––preserve–modification–time option to cpio). See the "Discussion" and "Examples" sections under cpio on page 647. –exec command \; The file being evaluated meets this action criterion if the command returns a 0 (zero [true]) exit status. You must terminate the command with a quoted semicolon. A pair of braces ({}) within the command represents the name of the file being evaluated. You can use the –exec action criterion at the end of a group of other criteria to execute the command if the preceding criteria are met. Refer to the following "Discussion" section for more information. See the section on xargs on page 881 for a more efficient way of doing what this option does. –group name The file being evaluated meets this criterion if it is associated with the group named name. You can use a numeric group ID in place of name. –inum n The file being evaluated meets this criterion if its inode number is n. 690 find –links ±n The file being evaluated meets this criterion if it has ±n links. –mtime ±n (modify time) The file being evaluated meets this criterion if it was last modified ±n days ago. –name filename The file being evaluated meets this criterion if the pattern filename matches its name. The filename can include wildcard characters (*, ?, and []) but these characters must be quoted. –newer filename The file being evaluated meets this criterion if it was modified more recently than filename. –nogroup The file being evaluated meets this criterion if it does not belong to a group known on the local system. –nouser The file being evaluated meets this criterion if it does not belong to a user known on the local system. –ok command \; This action criterion is the same as –exec except it displays each command to be executed enclosed in angle brackets as a prompt and executes the command only if it receives a response that starts with a y or Y from standard input. –perm [±]mode The file being evaluated meets this criterion if it has the access permissions given by mode. If mode is preceded by a minus sign (–), the file access permissions must include all the bits in mode. For example, if mode is 644, a file with 755 permissions will meet this criterion. If mode is preceded by a plus sign (+), the file access permissions must include at least one of the bits in mode. If no plus or minus sign precedes mode, the mode of the file must exactly match mode. You may use either a symbolic or octal representation for mode (see chmod on page 626). –print The file being evaluated always meets this action criterion. When evaluation of the expression reaches this criterion, find displays the pathname of the file it is evaluating. If –print is the only criterion in the expression, find displays the names of all files in the directory-list. If this criterion appears with other criteria, find displays the name only if the preceding criteria are met. If no action criteria appear in the expression, –print is assumed. (Refer to the following "Discussion" and "Notes" sections.) find 691 –size ±n[c|k|M|G] The file being evaluated meets this criterion if it is the size specified by ±n, measured in 512-byte blocks. Follow n with the letter c to measure files in characters, k to measure files in kilobytes, M to measure files in megabytes, or G to measure files in gigabytes. –type filetype The file being evaluated meets this criterion if its file type is specified by filetype. Select a filetype from the following list: b c d f l p s Block special file Character special file Directory file Ordinary file Symbolic link FIFO (named pipe) Socket –user name The file being evaluated meets this criterion if it belongs to the user with the username name. You can use a numeric user ID in place of name. –xdev The file being evaluated always meets this action criterion. It prevents find from searching directories in filesystems other than the one specified by directory-list. Also –mount. Under Mac OS X, use the –x option. L Discussion Assume x and y are criteria. The following command line never tests whether the file meets criterion y if it does not meet criterion x. Because the criteria are separated by a SPACE (the Boolean AND operator), once find determines that criterion x is not met, the file cannot meet the criteria so find does not continue testing. You can read the expression as "(test to see whether) the file meets criterion x and [SPACE means and] criterion y." $ find dir x y The next command line tests the file against criterion y if criterion x is not met. The file can still meet the criteria so find continues the evaluation. You can read the expression as "(test to see whether) the file meets criterion x or criterion y." If the file meets criterion x, find does not evaluate criterion y as there is no need to do so. $ find dir x -or y Action criteria Certain "criteria" do not select files but rather cause find to take action. The action is triggered when find evaluates one of these action criteria. Therefore, the position of an action criterion on the command line—not the result of its evaluation—determines whether find takes the action. 692 find The –print action criterion causes find to display the pathname of the file it is testing. The following command line displays the names of all files in the dir directory (and all its subdirectories), regardless of whether they meet the criterion x: $ find dir -print x The following command line displays only the names of the files in the dir directory that meet criterion x: $ find dir x -print This use of –print after the testing criteria is the default action criterion. The following command line generates the same output as the previous one: $ find dir x Notes You can use the –a operator between criteria to improve clarity. This operator is a Boolean AND operator, just as the SPACE is. You may want to consider using pax (page 786) in place of cpio. Examples The simplest find command has no arguments and lists the files in the working directory and all subdirectories: $ find . ... The following command finds the files in the working directory and subdirectories that have filenames beginning with a. The command uses a period to designate the working directory. To prevent the shell from interpreting the a* as an ambiguous file reference, it is enclosed within single quotation marks. $ find . -name 'a*' The –print criterion is implicit in the preceding command. If you omit the directory-list argument, find searches the working directory. The next command performs the same function as the preceding one without explicitly specifying the working directory: $ find -name 'a*' The next command sends a list of selected filenames to the cpio utility, which writes them to the device mounted on /dev/sde1. The first part of the command line ends with a pipe symbol, so the shell expects another command to follow and displays a secondary prompt (>) before accepting the rest of the command line. You can read this find command as "find, in the root directory and all subdirectories (/), ordinary files (-type f) that have been modified within the past day (–mtime –1), with the exception of files whose names are suffixed with .o (! –name '*.o')." (An object file carries a .o suffix and usually does not need to be preserved because it can be re-created from the corresponding source file.) find 693 $ find / -type f -mtime -1 ! -name '*.o' -print | > cpio -oB > /dev/sde1 The following command finds, displays the filenames of, and deletes the files named core or junk in the working directory and its subdirectories: $ find . \(-name core -o -name junk \) -print -exec rm {} \; ... The parentheses and the semicolon following –exec are quoted so the shell does not treat them as special characters. SPACE s separate the quoted parentheses from other elements on the command line. Read this find command as "find, in the working directory and subdirectories (.), files named core (–name core) or (–o) junk (–name junk) [if a file meets these criteria, continue] and (SPACE) print the name of the file (–print) and (SPACE) delete the file (–exec rm {})." The following shell script uses find in conjunction with grep to identify files that contain a particular string. This script enables you to look for a file when you remember its contents but cannot remember its filename. The finder script locates files in the working directory and subdirectories that contain the string specified on the command line. The –type f criterion causes find to pass to grep only the names of ordinary files, not directory files. $ cat finder find . -type f -exec grep -l "$1" {} \; $ finder "Executive Meeting" ./january/memo.0102 ./april/memo.0415 When called with the string Executive Meeting, finder locates two files containing that string: ./january/memo.0102 and ./april/memo.0415. The period (.) in the pathnames represents the working directory; january and april are subdirectories of the working directory. The grep utility with the ––recursive option performs the same function as the finder script. The next command looks in two user directories for files that are larger than 100 blocks (–size +100) and have been accessed only more than five days ago—that is, files that have not been accessed within the past five days (–atime +5). This find command then asks whether you want to delete the file (–ok rm {}). You must respond to each query with y (for yes) or n (for no). The rm command works only if you have write and execute access permissions to the directory. $ find /home/max /home/hls -size +100 -atime +5 -ok rm {} \; < rm ... /home/max/notes >? y < rm ... /home/max/letter >? n ... In the next example, /home/max/memos is a symbolic link to Sam's directory named /home/sam/memos. When you use the –follow option with find, the symbolic link is followed, and that directory is searched. 694 find $ ls -l /home/max lrwxrwxrwx 1 max -rw-r--r-- 1 max pubs pubs 17 Aug 19 17:07 memos -> /home/sam/memos 5119 Aug 19 17:08 report $ find /home/max -print /home/max /home/max/memos /home/max/report /home/max/.profile $ find /home/max -follow -print /home/max /home/max/memos /home/max/memos/memo.817 /home/max/memos/memo.710 /home/max/report /home/max/.profile finger 695 Displays information about users finger [options] [user-list] The finger utility displays the usernames of users, together with their full names, terminal device numbers, times they logged in, and other information. The options control how much information finger displays, and the user-list specifies which users finger displays information about. The finger utility can retrieve information from both local and remote systems. Arguments Without any arguments, finger provides a short (–s) report on users who are logged in on the local system. When you specify a user-list, finger provides a long (–l) report on each user in the user-list. Names in the user-list are not case sensitive. If the name includes an at sign (@), the finger utility interprets the name following the @ as the name of a remote host to contact over the network. If a username appears in front of the @ sign, finger provides information about that user on the remote system. Options –l (long) Displays detailed information (the default display when user-list is specified). –m (match) If a user-list is specified, displays entries only for those users whose username matches one of the names in user-list. Without this option the user-list names match usernames and full names. –p (no plan, project, or pgpkey) Does not display the contents of .plan, .project, and .pgpkey files for users. Because these files may contain backslash escape sequences that can change the behavior of the screen, you may not wish to view them. Normally the long listing displays the contents of these files if they exist in the user's home directory. –s (short) Provides a short report for each user (the default display when user-list is not specified). Discussion The long report provided by the finger utility includes the user's username, full name, home directory location, and login shell, plus information about when the user last logged in and how long it has been since the user last typed on the keyboard and read her email. After extracting this information from system files, finger displays the contents of the ~/.plan, ~/.project, and ~/.pgpkey files in the user's home directory. It is up to each user to create and maintain these files, which usually provide more information about the user (such as telephone number, postal mail address, schedule, interests, and PGP key). The short report generated by finger is similar to that provided by the w utility; it includes the user's username, his full name, the device number of the user's terminal, the amount of time that has elapsed since the user last typed on the terminal finger finger 696 finger keyboard, the time the user logged in, and the location of the user's terminal. If the user logged in over the network, finger displays the name of the remote system. Notes Not all Linux distributions install finger by default. When you specify a network address, the finger utility queries a standard network service that runs on the remote system. Although this service is supplied with most Linux systems, some administrators choose not to run it (so as to minimize the load on their systems, eliminate possible security risks, or simply maintain privacy). If you try to use finger to get information on someone at such a site, the result may be an error message or nothing at all. The remote system determines how much information to share with the local system and in which format. As a result the report displayed for any given system may differ from the examples shown here. See also "finger: Lists Users on the System" on page 68. A file named ~/.nofinger causes finger to deny the existence of the person whose home directory it appears in. For this subterfuge to work, the finger query must originate from a system other than the local host and the fingerd daemon must be able to see the .nofinger file (generally the home directory must have its execute bit for other users set). Examples The first example displays information on the users logged in on the local system: $ finger Login max hls sam Name Max Wild Helen Simpson Sam the Great Tty Idle tty1 13:29 pts/1 13:29 pts/2 * Login Jun 25 Jun 25 Jun 26 Time Office Office Phone 21:03 21:02 (:0) 07:47 (bravo.example.com) The asterisk (*) in front of the name of Helen's terminal (TTY) line indicates she has blocked others from sending messages directly to her terminal (see mesg on page 71). A long report displays the string messages off for users who have disabled messages. The next two examples cause finger to contact the remote system named kudos over the network for information: $ finger [kudos] Login max roy @kudos Name Max Wild Roy Wong Tty tty1 pts/2 Idle Login Time Office 23:15 Jun 25 11:22 Jun 25 11:22 $ finger watson@kudos [kudos] Login: max Name: Max Wild Directory: /home/max Shell: /bin/zsh On since Sat Jun 25 11:22 (PDT) on tty1, idle 23:22 Last login Sun Jun 26 06:20 (PDT) on ttyp2 from speedy Mail last read Thu Jun 23 08:10 2005 (PDT) Plan: For appointments contact Sam the Great, x1963. Office Phone fmt 697 fmt fmt Formats text very simply fmt [option] [file-list] The fmt utility does simple text formatting by attempting to make all nonblank lines nearly the same length. Arguments The fmt utility reads the files in file-list and sends a formatted version of their contents to standard output. If you do not specify a filename or if you specify a hyphen (–) in place of a filename, fmt reads from standard input. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––split-only –s Splits long lines but does not fill short lines. L –s Replaces multiple adjacent SPACE characters with a single SPACE. O ––tagged-paragraph –t Indents all but the first line of each paragraph. L –t n Specifies n as the number of SPACEs per TAB stop. The default is eight. O ––uniform-spacing –u Changes the formatted output so that one two SPACEs appear between sentences. L SPACE appears between words and ––width=n –w n Changes the output line length to n characters. Without this option, fmt keeps output lines close to 75 characters wide. You can also specify this option as –n. Notes The fmt utility works by moving NEWLINE characters. The indention of lines, as well as the spacing between words, is left intact. You can use this utility to format text while you are using an editor, such as vim. For example, you can format a paragraph with the vim editor in command mode by positioning the cursor at the top of the paragraph and then entering !}fmt –60. This command replaces the paragraph with the output generated by feeding it through fmt, specifying a width of 60 characters. Type u immediately if you want to undo the formatting. 698 fmt Examples The following example shows how fmt attempts to make all lines the same length. The –w 50 option gives a target line length of 50 characters. $ cat memo One factor that is important to remember while administering the dietary intake of Charcharodon carcharias is that there is, at least from the point of view of the subject, very little differentiating the prepared morsels being proffered from your digits. In other words, don't feed the sharks! $ fmt -w 50 memo One factor that is important to remember while administering the dietary intake of Charcharodon carcharias is that there is, at least from the point of view of the subject, very little differentiating the prepared morsels being proffered from your digits. In other words, don't feed the sharks! The next example demonstrates the ––split-only option. Long lines are broken so that none is longer than 50 characters; this option prevents fmt from filling short lines. $ fmt -w 50 --split-only memo One factor that is important to remember while administering the dietary intake of Charcharodon carcharias is that there is, at least from the point of view of the subject, very little differentiating the prepared morsels being proffered from your digits. In other words, don't feed the sharks! fsck 699 fsck fsck [options] [filesystem-list] The fsck utility verifies the integrity of a filesystem and reports on and optionally repairs problems it finds. It is a front end for filesystem checkers, each of which is specific to a certain filesystem type. The fsck utility is available under Linux only; under OS X, use diskutil. L Arguments Without the –A option and with no filesystem-list, fsck checks the filesystems listed in the /etc/fstab file one at a time (serially). With the –A option and with no filesystem-list, fsck checks all the filesystems listed in the /etc/fstab file in parallel if possible. See the –s option for a discussion of checking filesystems in parallel. The filesystem-list specifies the filesystems to be checked. It can either specify the name of the device that holds the filesystem (for example, /dev/hda2) or, if the filesystem appears in /etc/fstab, specify the mount point (for example, /usr) for the filesystem. The filesystem-list can also specify the label for the filesystem from /etc/fstab (for example, LABEL=home). Options When you run fsck, you can specify both global options and options specific to the filesystem type that fsck is checking (for example, ext2, ext3, msdos, reiserfs). Global options must precede type-specific options. Global Options –A (all) Processes all filesystems listed in the /etc/fstab file, in parallel if possible. See the –s option for a discussion of checking filesystems in parallel. Do not specify a filesystem-list when you use this option; you can specify filesystem types to be checked with the –t option. Use this option with the –a, –p, or –n option so fsck does not attempt to process filesystems in parallel interactively (in which case you would have no way of responding to its multiple prompts). –N (no) Assumes a no response to any questions that arise while processing a filesystem. This option generates the messages you would normally see but causes fsck to take no action. –R (root-skip) With the –A option, does not check the root filesystem. This option is useful when the system boots, because the root filesystem may be mounted with read-write access. –s (serial) Causes fsck to process filesystems one at a time. Without this option, fsck processes multiple filesystems that reside on separate physical disk drives in parallel. Parallel processing enables fsck to process multiple filesystems more quickly. This option is required if you want to process filesystems interactively. See the –a, –p, or –N (or –n, on some filesystems) option to turn off interactive processing. fsck Checks and repairs a filesystem 700 fsck –T (title) Causes fsck not to display its title. –t fstype (filesystem type) A comma-separated list that specifies the filesystem type(s) to process. With the –A option, fsck processes all the filesystems in /etc/fstab that are of type fstype. Common filesystem types are ext2, ext3, ext4, msdos, and reiserfs. You do not typically check remote NFS filesystems. –V (verbose) Displays more output, including filesystem type-specific commands. Filesystem Type-Specific Options The following command lists the filesystem checking utilities available on the local system. Files with the same inode numbers are linked (page 107). $ ls -i /sbin/*fsck* 63801 /sbin/dosfsck 63856 /sbin/fsck.cramfs 63763 /sbin/e2fsck 63763 /sbin/fsck.ext2 63780 /sbin/fsck 63763 /sbin/fsck.ext3 63801 /sbin/fsck.msdos 63801 /sbin/fsck.vfat Review the man page or give the pathname of the filesystem checking utility to determine which options the utility accepts: $ /sbin/fsck.ext3 Usage: /sbin/fsck.ext3 [-panyrcdfvstDFSV] [-b superblock] [-B blocksize] [-I inode_buffer_blocks] [-P process_inode_size] [-l|-L bad_blocks_file] [-C fd] [-j ext-journal] [-E extended-options] device Emergency help: -p -n -y -c -f ... Automatic repair (no questions) Make no changes to the filesystem Assume "yes" to all questions Check for bad blocks and add them to the badblock list Force checking even if filesystem is marked clean The following options apply to many filesystem types, including ext2 and ext3: –a (automatic) Same as the –p option; kept for backward compatibility. –f (force) Forces fsck to check filesystems even if they are clean. A clean filesystem is one that was just successfully checked with fsck or was successfully unmounted and has not been mounted since then. Clean filesystems are skipped by fsck, which greatly speeds up system booting under normal conditions. For information on setting up periodic, automatic filesystem checking on ext2, ext3, and ext4 filesystems, see tune2fs page 868. –n (no) Same as the –N global option. Does not work with all filesystems. –p (preen) Attempts to repair all minor inconsistencies it finds when processing a filesystem. If any problems are not repaired, fsck terminates with a nonzero exit status. This option runs fsck in batch mode; as a consequence, it does not fsck 701 ask whether to correct each problem it finds. The –p option is commonly used with the –A option when checking filesystems while booting Linux. –r (interactive) Asks whether to correct or ignore each problem that is found. For many filesystem types, this behavior is the default. This option is not available on all filesystems. –y (yes) Assumes a yes response to any questions that fsck asks while processing a filesystem. Use this option with caution, as it gives fsck free reign to do what it thinks is best to clean up a filesystem. Notes The fsck and fsck_hfs utilities are deprecated under Mac OS X version 10.5 and later. Apple suggests using diskutil (page 668) instead. You can run fsck from a live or rescue CD. When a filesystem is consistent, fsck displays a report such as the following: # fsck -f /dev/sdb1 fsck 1.40.8 (13-Mar-2008) e2fsck 1.40.8 (13-Mar-2008) Pass 1: Checking inodes, blocks, and sizes Pass 2: Checking directory structure Pass 3: Checking directory connectivity Pass 4: Checking reference counts Pass 5: Checking group summary information /dev/sdb1: 710/4153408 files (10.1% non-contiguous), 455813/8303589 blocks Interactive mode You can run fsck either interactively or in batch mode. For many filesystems, unless you use one of the –a, –p, –y, or –n options, fsck runs in interactive mode. In interactive mode, if fsck finds a problem with a filesystem, it reports the problem and allows you to choose whether to repair or ignore it. If you repair a problem you may lose some data; however, that is often the most reasonable alternative. Although it is technically feasible to repair files that are damaged and that fsck says you should remove, this action is rarely practical. The best insurance against significant loss of data is to make frequent backups. Order of checking The fsck utility looks at the sixth column in the /etc/fstab file to determine if, and in which order, it should check filesystems. A 0 (zero) in this position indicates the filesystem should not be checked. A 1 (one) indicates that it should be checked first; this status is usually reserved for the root filesystem. A 2 (two) indicates that the filesystem should be checked after those marked with a 1. fsck is a front end Similar to mkfs (page 764), fsck is a front end that calls other utilities to handle various types of filesystems. For example, fsck calls e2fsck to check the widely used ext2 and ext3 filesystems. Refer to the e2fsck man page for more information. Other utilities that fsck calls are typically named fsck.type, where type is the filesystem type. By splitting fsck in this manner, filesystem developers can provide programs to check their filesystems without affecting the development of other filesystems or changing how system administrators use fsck. 702 fsck Boot time Run fsck on filesystems that are unmounted or are mounted readonly. When Linux is booting, the root filesystem is first mounted readonly to allow it to be processed by fsck. If fsck finds no problems with the root filesystem, it is then remounted (using the remount option to the mount utility) read-write and fsck is typically run with the –A, –R, and –p options. lost+found When it encounters a file that has lost its link to its filename, fsck asks whether to reconnect it. If you choose to reconnect it, the file is put in a directory named lost+found in the root directory of the filesystem where the file was found. The reconnected file is given its inode number as a name. For fsck to restore files in this way, a lost+found directory must be present in the root directory of each filesystem. For example, if a system uses the /, /usr, and /home filesystems, you should have these three lost+found directories: /lost+found, /usr/lost+found, and /home/lost+found. Each lost+found directory must have unused entries in which fsck can store the inode numbers for files that have lost their links. When you create an ext2/ext3/ext4 filesystem, mkfs (page 764) creates a lost+found directory with the required unused entries. Alternatively, you can use the mklost+found utility to create this directory in ext2/ext3/ext4 filesystems if needed. On other types of filesystems, you can create the unused entries by adding many files to the directory and then removing them. Try using touch (page 862) to create 500 entries in the lost+found directory and then using rm to delete them. Messages Table V-15 lists fsck's common messages. In general fsck suggests the most logical way of dealing with a problem in the file structure. Unless you have information that suggests another response, respond to the prompts with yes. Use the system backup tapes or disks to restore data that is lost as a result of this process. Table V-15 Common fsck messages Phase (message) What fsck checks Phase 1 - Checking inodes, blocks, and sizes Checks inode information. Phase 2 - Checking directory structure Looks for directories that point to bad inodes that fsck found in Phase 1. Phase 3 - Checking directory connectivity Looks for unreferenced directories and a nonexistent or full lost+found directory. Phase 4 - Checking reference counts Checks for unreferenced files, a nonexistent or full lost+found directory, bad link counts, bad blocks, duplicated blocks, and incorrect inode counts. Phase 5 - Checking group summary information Checks whether the free list and other filesystem structures are OK. If any problems are found with the free list, Phase 6 is run. Phase 6 - Salvage free list If Phase 5 found any problems with the free list, Phase 6 fixes them. fsck 703 Cleanup Once it has repaired the filesystem, fsck informs you about the status of the filesystem. The fsck utility displays the following message after it repairs a filesystem: *****File System Was Modified***** On ext2/ext3/ext4 filesystems, fsck displays the following message when it has finished checking a filesystem: filesys: used/maximum files (percent non-contiguous), used/maximum blocks This message tells you how many files and disk blocks are in use as well as how many files and disk blocks the filesystem can hold. The percent non-contiguous tells you how fragmented the disk is. 704 ftp ftp Transfers files over a network ftp [options] [remote-system] The ftp utility is a user interface to the standard File Transfer Protocol (FTP), which transfers files between systems that can communicate over a network. To establish an FTP connection, you must have access to an account (personal, guest, or anonymous) on the remote system. Use FTP only to download public information security FTP is not a secure protocol. The ftp utility sends your password over the network as cleartext, which is not a secure practice. You can use sftp as a secure replacement for ftp if the server is running OpenSSH. You can also use scp (page 810) for many FTP functions other than allowing anonymous users to download information. Because scp uses an encrypted connection, user passwords and data cannot be sniffed. Arguments Options The remote-system is the name or network address of the server, running an FTP daemon (e.g., ftpd, vsftpd, or sshd), you want to exchange files with. –i (interactive) Turns off prompts during file transfers with mget and mput. See also prompt. –n (no automatic login) Disables automatic logins. –p (passive mode) Starts ftp in passive mode (page 706). –v (verbose) Tells you more about how ftp is working. Displays responses from the remote-system and reports transfer times and speeds. Discussion The ftp utility is interactive. After you start it, ftp prompts you to enter commands to set parameters and transfer files. You can use a number of commands in response to the ftp> prompt; following are some of the more common ones. ![command] Escapes to (spawns) a shell on the local system; use CONTROL-D or exit to return to ftp when you are finished using the local shell. Follow the exclamation point with a command to execute that command only; ftp returns to the ftp> prompt when the command completes executing. Because the shell that ftp spawns with this command is a child of the shell that is running ftp, no changes you make in this shell are preserved when you return to ftp. Specifically, when you want to copy files to a local directory other than the directory that you started ftp from, you need to use the ftp lcd command to change the local working directory: Issuing a cd command in the spawned shell will not make the change you desire. See page 709 for an example. ascii Sets the file transfer type to ASCII. This command allows you to transfer text files from systems that end lines with a RETURN/LINEFEED combination and automatically strip ftp 705 off the RETURN. Such a transfer is useful when the remote computer is a DOS or MS Windows machine. The cr command must be ON for ascii to work. binary Sets the file transfer type to binary. This command allows you to transfer files that contain non-ASCII (unprintable) characters correctly. It also works for ASCII files that do not require changes to the ends of lines. bye Closes the connection to a remote system and terminates ftp. Same as quit. cd remote-directory Changes to the working directory named remote-directory on the remote system. close Closes the connection with the remote system without exiting from ftp. cr (carriage return) Toggles RETURN stripping when you retrieve files in ASCII mode. See ascii. dir [directory [file]] Displays a listing of the directory named directory from the remote system. When you do not specify directory, the working directory is displayed. When you specify a file, the listing is saved on the local system in a file named file. get remote-file [local-file] Copies remote-file to the local system under the name local-file. Without local-file, ftp uses remote-file as the filename on the local system. The remote-file and local-file names can be pathnames. glob Toggles filename expansion for the mget and mput commands and displays the current state (Globbing on or Globbing off). help Displays a list of commands recognized by the ftp utility on the local system. lcd [local_directory] (local change directory) Changes the working directory on the local system to local_directory. Without an argument, this command changes the working directory on the local system to your home directory (just as cd does without an argument). ls [directory [file]] Similar to dir but produces a more concise listing on some remote systems. mget remote-file-list (multiple get) Unlike the get command, allows you to retrieve multiple files from the remote system. You can name the remote files literally or use wildcards (see glob). See also prompt. mput local-file-list (multiple put) The mput command allows you to copy multiple files from the local system to the remote system. You can name the local files literally or use wildcards (see glob). See also prompt. open Interactively specifies the name of the remote system. This command is useful if you did not specify a remote system on the command line or if the attempt to connect to the system failed. 706 ftp passive Toggles between the active (PORT—the default) and passive (PASV) transfer modes and displays the transfer mode. See "Passive versus active connections" under the "Notes" section. prompt When using mget or mput to receive or send multiple files, ftp asks for verification (by default) before transferring each file. This command toggles that behavior and displays the current state (Interactive mode off or Interactive mode on). put local-file [remote-file] Copies local-file to the remote system under the name remote-file. Without remotefile, ftp uses local-file as the filename on the remote system. The remote-file and local-file names can be pathnames. pwd Causes ftp to display the pathname of the remote working directory. Use !pwd to display the name of the local working directory. quit Closes the connection to a remote system and terminates ftp. Same as bye. reget remote-file Attempts to resume an aborted transfer. This command is similar to get, but instead of overwriting an existing local file, ftp appends new data to it. Not all servers sup- port reget. user [username] If the ftp utility did not log you in automatically, you can specify your account name as username. If you omit username, ftp prompts you for a username. Notes A Linux or Mac OS X system running ftp can exchange files with any of the many operating systems that support the FTP protocol. Many sites offer archives of free information on an FTP server, although many of these FTP sites are merely alternatives to an easier-to-access Web site (for example, ftp://ftp.ibiblio.org/pub/Linux and . Most browsers can connect to and download files from FTP servers. The ftp utility makes no assumptions about filesystem naming or structure because you can use ftp to exchange files with non-UNIX/Linux systems (whose filename conventions may be different). Anonymous FTP Many systems—most notably those from which you can download free software—allow you to log in as anonymous. Most systems that support anonymous logins accept the name ftp as an easier-to-spell and quicker-to-enter synonym for anonymous. An anonymous user is usually restricted to a portion of a filesystem set aside to hold files that are to be shared with remote users. When you log in as an anonymous user, the server prompts you to enter a password. Although any password may be accepted, by convention you are expected to supply your email address. Many systems that permit anonymous access store interesting files in the pub directory. Passive versus active connections A client can ask an FTP server to establish either a PASV (passive—the default) or PORT (active) connection for data transfer. Some servers are limited to one type of connection. The difference between passive and active FTP connections lies in whether the client or server initiates the data connection. In passive mode, the client ftp 707 initiates the data connection to the server (on port 20 by default); in active mode, the server initiates the data connection (there is no default port). Neither type of connection is inherently more secure. Passive connections are more common because a client behind a NAT (page 967) firewall can connect to a passive server and because it is simpler to program a scalable passive server. Automatic login You can store server-specific FTP username and password information so that you do not have to enter it each time you visit an FTP site. Each line of the ~/.netrc file identifies a server. When you connect to an FTP server, ftp reads ~/.netrc to determine whether you have an automatic login set up for that server. The format of a line in ~/.netrc is machine server login username password passwd where server is the name of the server, username is your username, and passwd is your password on server. Replace machine with default on the last line of the file to specify a username and password for systems not listed in ~/.netrc. The default line is useful for logging in on anonymous servers. A sample ~/.netrc file follows: $ cat ~/.netrc machine bravo login max password mypassword default login anonymous password max@example.com To protect the account information in .netrc, make it readable by only the user whose home directory it appears in. Refer to the netrc man page for more information. Examples Connect and log in Following are two ftp sessions wherein Max transfers files from and to an FTP server named bravo. When Max gives the command ftp bravo, the local ftp client connects to the server, which asks for a username and a password. Because he is logged in on his local system as max, ftp suggests that he log in on bravo as max. To log in as max, Max could just press RETURN. His username on bravo is watson, however, so he types watson in response to the Name (bravo:max): prompt. Max responds to the Password: prompt with his normal (remote) system password, and the FTP server greets him and informs him that it is Using binary mode to transfer files. With ftp in binary mode, Max can transfer ASCII and binary files. $ ftp bravo Connected to bravo. 220 (vsFTPd 2.0.7) 530 Please login with USER and PASS. 530 Please login with USER and PASS. KERBEROS_V4 rejected as an authentication type Name (bravo:max): watson 331 Please specify the password. Password: 230 Login successful. Remote system type is UNIX. Using binary mode to transfer files. ftp> 708 ftp After logging in, Max uses the ftp ls command to display the contents of his remote working directory, which is his home directory on bravo. Then he cds to the memos directory and displays the files there. ls and cd ftp> ls 227 Entering Passive Mode (192,168,0,6,79,105) 150 Here comes the directory listing. drwxr-xr-x 2 500 500 4096 Oct 10 23:52 expenses drwxr-xr-x 2 500 500 4096 Oct 10 23:59 memos drwxrwxr-x 22 500 500 4096 Oct 10 23:32 tech 226 Directory send OK. ftp> cd memos 250 Directory successfully changed. ftp> ls 227 Entering Passive Mode (192,168,0,6,114,210) 150 Here comes the directory listing. -rw-r--r-1 500 500 4770 Oct 10 -rw-r--r-1 500 500 7134 Oct 10 -rw-r--r-1 500 500 9453 Oct 10 -rw-r--r-1 500 500 3466 Oct 10 -rw-r--r-1 500 500 1945 Oct 10 226 Directory send OK. 23:58 23:58 23:58 23:59 23:59 memo.0514 memo.0628 memo.0905 memo.0921 memo.1102 Next Max uses the ftp get command to copy memo.1102 from the server to the local system. His use of binary mode ensures that he will get a good copy of the file regardless of whether it is in binary or ASCII format. The server confirms that the file was copied successfully and notes the size of the file and the time it took to copy. Max then copies the local file memo.1114 to the remote system. The file is copied into his remote working directory, memos. get and put ftp> get memo.1102 local: memo.1102 remote: memo.1102 227 Entering Passive Mode (192,168,0,6,194,214) 150 Opening BINARY mode data connection for memo.1102 (1945 bytes). 226 File send OK. 1945 bytes received in 7.1e-05 secs (2.7e+04 Kbytes/sec) ftp> put memo.1114 local: memo.1114 remote: memo.1114 227 Entering Passive Mode (192,168,0,6,174,97) 150 Ok to send data. 226 File receive OK. 1945 bytes sent in 2.8e-05 secs (6.8e+04 Kbytes/sec) After a while Max decides he wants to copy all the files in the memos directory on bravo to a new directory on the local system. He gives an ls command to make sure he is going to copy the right files, but ftp has timed out. Instead of exiting from ftp and giving another ftp command from the shell, Max gives ftp an open bravo command to reconnect to the server. After logging in, he uses the ftp cd command to change directories to memos on the server. ftp Timeout and open lcd (local cd) 709 ftp> ls 421 Timeout. Passive mode refused. ftp> open bravo Connected to bravo (192.168.0.6). 220 (vsFTPd 1.1.3) ... ftp> cd memos 250 Directory successfully changed. At this point, Max realizes he has not created the new directory to hold the files he wants to download. Giving an ftp mkdir command would create a new directory on the server, but Max wants a new directory on the local system. He uses an exclamation point (!) followed by a mkdir memos.hold command to invoke a shell and run mkdir on the local system, thereby creating a directory named memos.hold in his working directory on the local system. (You can display the name of your working directory on the local system with !pwd.) Next, because he wants to copy files from the server to the memos.hold directory on his local system, Max has to change his working directory on the local system. Giving the command !cd memos.hold will not accomplish what Max wants to do because the exclamation point spawns a new shell on the local system and the cd command would be effective only in the new shell, which is not the shell that ftp is running under. For this situation, ftp provides the lcd (local cd) command, which changes the working directory for ftp and reports on the new local working directory. ftp> !mkdir memos.hold ftp> lcd memos.hold Local directory now /home/max/memos.hold Max uses the ftp mget (multiple get) command followed by the asterisk (*) wildcard to copy all the files from the remote memos directory to the memos.hold directory on the local system. When ftp prompts him for the first file, he realizes that he forgot to turn off the prompts, so he responds with n and presses CONTROL-C to stop copying files in response to the second prompt. The server checks whether he wants to continue with his mget command. Next Max gives the ftp prompt command, which toggles the prompt action (turns it off if it is on and turns it on if it is off). Now when he gives an mget * command, ftp copies all the files without prompting him. After getting the files he wants, Max gives a quit command to close the connection with the server, exit from ftp, and return to the local shell prompt. mget and prompt ftp> mget * mget memo.0514? n mget memo.0628? CONTROL-C Continue with mget? n ftp> prompt Interactive mode off. 710 ftp ftp> mget * local: memo.0514 remote: memo.0514 227 Entering Passive Mode (192,168,0,6,53,55) 150 Opening BINARY mode data connection for memo.0514 (4770 bytes). 226 File send OK. 4770 bytes received in 8.8e-05 secs (5.3e+04 Kbytes/sec) local: memo.0628 remote: memo.0628 227 Entering Passive Mode (192,168,0,6,65,102) 150 Opening BINARY mode data connection for memo.0628 (7134 bytes). 226 File send OK. ... 150 Opening BINARY mode data connection for memo.1114 (1945 bytes). 226 File send OK. 1945 bytes received in 3.9e-05 secs (4.9e+04 Kbytes/sec) ftp> quit 221 Goodbye. gawk 711 Searches for and processes patterns in a file gawk [options] [program] [file-list] gawk [options] –f program-file [file-list] AWK is a pattern-scanning and processing language that searches one or more files for records (usually lines) that match specified patterns. It processes lines by performing actions, such as writing the record to standard output or incrementing a counter, each time it finds a match. As opposed to procedural languages, AWK is data driven: You describe the data you want to work with and tell AWK what to do with the data once it finds it. See Chapter 12 for information on gawk tip See Chapter 12 starting on page 531 for information on the awk, gawk, and mawk implementations of the AWK language. gawk gawk 712 gcc gcc Compiles C and C++ programs gcc gcc [options] file-list [–larg] g++ [options] file-list [–larg] The Linux and Mac OS X operating systems use the GNU C compiler, gcc, to preprocess, compile, assemble, and link C language source files. The same compiler with a different front end, g++, processes C++ source code. The gcc and g++ compilers can also assemble and link assembly language source files, link object files only, or build object files for use in shared libraries. These compilers take input from files you specify on the command line. Unless you use the –o option, they store the executable program in a.out. The gcc and g++ compilers are part of GCC, the GNU Compiler Collection, which includes front ends for C, C++, Objective C, Fortran, Java, and Ada as well as libraries for these languages. Go to gcc.gnu.org for more information. gcc and g++ tip Although this section specifies the gcc compiler, most of it applies to g++ as well. Arguments The file-list is a list of files gcc is to process. Options Without any options gcc accepts C language source files, assembly language files, object files, and other files described in Table V-16 on page 715. The gcc utility preprocesses, compiles, assembles, and links these files as appropriate, producing an executable file named a.out. If gcc is used to create object files without linking them to produce an executable file, each object file is named by adding the extension .o to the basename of the corresponding source file. If gcc is used to create an executable file, it deletes the object files after linking. Some of the most commonly used options are listed here. When certain filename extensions are associated with an option, you can assume gcc adds the extension to the basename of the source file. –c (compile) Suppresses the linking step of compilation. Compiles and/or assembles source code files and leaves the object code in files with the extension .o. –Dname[=value] Usually #define preprocessor directives are given in header, or include, files. You can use this option to define symbolic names on the command line instead. For example, –DLinux is equivalent to placing the line #define Linux in an include file, and –DMACH=i586 is the same as #define MACH i586. gcc 713 –E (everything) For source code files, suppresses all steps of compilation except preprocessing and writes the result to standard output. By convention the extension .i is used for preprocessed C source and .ii for preprocessed C++ source. –fpic Causes gcc to produce position-independent code, which is suitable for installing into a shared library. –fwritable-strings By default the GNU C compiler places string constants into protected memory, where they cannot be changed. Some (usually older) programs assume you can modify string constants. This option changes the behavior of gcc so that string constants can be modified. –g (gdb) Embeds diagnostic information in the object files. This information is used by symbolic debuggers, such as gdb. Although this option is necessary only if you later use a debugger, it is a good practice to include it as a matter of course. –Idirectory Looks for include files in directory before looking in the standard locations. Give this option multiple times to look in more than one directory. –larg (lowercase "l") Searches the directories /lib and /usr/lib for a library file named libarg.a. If the file is found, gcc then searches this library for any required functions. Replace arg with the name of the library you want to search. For example, the –lm option normally links the standard math library libm.a. The position of this option is significant: It generally needs to appear at the end of the command line but can be repeated multiple times to search different libraries. Libraries are searched in the order in which they appear on the command line. The linker uses the library only to resolve undefined symbols from modules that precede the library option on the command line. You can add other library paths to search for libarg.a using the –L option. –Ldirectory Adds directory to the list of directories to search for libraries given with the –l option. Directories that are added to the list with –L are searched before gcc looks in the standard locations for libraries. –o file (output) Names the executable program that results from linking file instead of a.out. –On (optimize) Attempts to improve (optimize) the object code produced by the compiler. The value of n may be 0, 1, 2, or 3 (or 06 if you are compiling code 714 gcc for the Linux kernel). The default value of n is 1. Larger values of n result in better optimization but may increase both the size of the object file and the time it takes gcc to run. Specify –O0 to turn off optimization. Many related options control precisely the types of optimizations attempted by gcc when you use –O. Refer to the gcc info page for details. –pedantic The C language accepted by the GNU C compiler includes features that are not part of the ANSI standard for the C language. Using this option forces gcc to reject these language extensions and accept only standard C programming language features. –Q Displays the names of functions as gcc compiles them. This option also displays statistics about each pass. –S (suppress) Suppresses the assembling and linking steps of compilation on source code files. The resulting assembly language files have .s filename extensions. –traditional Causes gcc to accept only C programming language features that existed in the traditional Kernighan and Ritchie C programming language. With this option, older programs written using the traditional C language (which existed before the ANSI standard C language was defined) can be compiled correctly. –Wall Causes gcc to warn you about questionable code in the source code files. Many related options control warning messages more precisely. Notes The preceding list of options represents only a small fraction of the full set of options available with the GNU C compiler. See the gcc info page for a complete list. Although the –o option is generally used to specify a filename in which to store object code, this option also allows you to name files resulting from other compilation steps. In the following example, the –o option causes the assembly language produced by the gcc command to be stored in the file acode instead of pgm.s, the default: $ gcc -S -o acode pgm.c The lint utility found in many UNIX systems is not available on Linux or Mac OS X. However, the –Wall option performs many of the same checks and can be used in place of lint. Table V-16 summarizes the conventions used by the C compiler for assigning filename extensions. gcc Table V-16 Examples 715 Filename extensions Extension Type of file .a Library of object modules .c C language source file .C, .cc, or .cxx C++ language source file .i Preprocessed C language source file .ii Preprocessed C++ language source file .o Object file .s Assembly language source file .S Assembly language source file that needs preprocessing The first example compiles, assembles, and links a single C program, compute.c. The executable output is stored in a.out. The gcc utility deletes the object file. $ gcc compute.c The next example compiles the same program using the C optimizer (–O option). It assembles and links the optimized code. The –o option causes gcc to store the executable output in compute. $ gcc -O -o compute compute.c Next a C source file, an assembly language file, and an object file are compiled, assembled, and linked. The executable output is stored in progo. $ gcc -o progo procom.c profast.s proout.o In the next example, gcc searches the standard math library found in /lib/libm.a when it is linking the himath program and stores the executable output in a.out: $ gcc himath.c -lm In the following example, the C compiler compiles topo.c with options that check the code for questionable source code practices (–Wall option) and violations of the ANSI C standard (–pedantic option). The –g option embeds debugging support in the executable file, which is saved in topo with the –o topo option. Full optimization is enabled with the –O3 option. The warnings produced by the C compiler are sent to standard output. In this example the first and last warnings result from the –pedantic option; the other warnings result from the –Wall option. 716 gcc $ gcc -Wall -g -O3 -pedantic -o topo topo.c In file included from topo.c:2: /usr/include/ctype.h:65: warning: comma at end of enumerator list topo.c:13: warning: return-type defaults to 'int' topo.c: In function 'main': topo.c:14: warning: unused variable 'c' topo.c: In function 'getline': topo.c:44: warning: 'c' might be used uninitialized in this function When compiling programs that rely on the X11 include files and libraries, you may need to use the –I and –L options to tell gcc where to locate those include files and libraries. The next example uses those options and instructs gcc to link the program with the basic X11 library: $ gcc -I/usr/X11R6/include plot.c -L/usr/X11R6/lib -lX11 717 GetFileInfo O Displays file attributes GetFileInfo [option] file The GetFileInfo utility displays file attributes (page 931), including the file's type and creator code, creation and last modification times, and attribute flags such as the invisible and locked flags. Arguments The file specifies a single file or a directory that GetFileInfo displays information about. Options The options for GetFileInfo correspond to the options for SetFile (page 813). Without an option, GetFileInfo reports on the metadata of file, indicating the flags that are set, the file's type and creator codes, and its creation and modification dates. Missing data is omitted. When you specify an option, GetFileInfo displays the information specified by that option only. This utility accepts a single option; it silently ignores additional options. –aflag (attribute) Reports the status of a single attribute flag named flag. This option displays 1 if flag is set and 0 if flag is not set. The flag must follow the –a immediately, without any intervening SPACEs. See Table D-2 on page 931 for a list of attribute flags. –c (creator) Displays the creator code of file. If file is a directory and has no creator code, this option displays an error message. –d (date) Displays the creation date of file as mm/dd/yyyy hh:mm:ss, using a 24-hour clock. –m (modification) Displays the modification date of file as mm/dd/yyyy hh:mm:ss, using a 24-hour clock. –P (no dereference) For each file that is a symbolic link, displays information about the symbolic link, not the file the link points to. This option affects all files and treats files that are not symbolic links normally. See page 625 for an example of the –P option. –t (type) Displays the type code of file. If file is a directory and has no type code, this option displays an error message. Discussion Without an option, GetFileInfo displays flags as the string avbstclinmedz, with uppercase letters denoting which flags are set. See page 931 for a discussion of attribute flags. GetFileInfo GetFileInfo O 718 GetFileInfo O Notes You can use the SetFile utility (page 813) to set file attributes. You can set Mac OS X permissions and ownership (page 93) using chmod (page 626) or chown (page 631), and you can display this information using ls (page 745) or stat (page 835). Directories do not have type or creator codes, and they may not have all flags. The GetFileInfo utility cannot read special files such as device files. Examples The first example shows the output from GetFileInfo when you call it without an option. $ GetFileInfo picture.jpg file: "/private/tmp/picture.jpg" type: "JPEG" creator: "GKON" attributes: avbstClinmedz created: 07/18/2005 15:15:26 modified: 07/18/2005 15:15:26 The only uppercase letter on the attributes line is C, indicating that this flag is set. The c flag tells the Finder to look for a custom icon for this file. See Table D-2 on page 931 for a list of flags. The next example uses the –a flag to display the attribute flags for a file: $ GetFileInfo -a /Applications/Games/Alchemy/Alchemy avBstclInmedz The output shows that the b and i flags are set. The GetFileInfo utility can process only one file each time you call it. The following multiline bash command uses a for loop (page 411) to display the creator codes of multiple files. The echo command displays the name of the file being examined because GetFileInfo does not always display the name of the file: $ for i in * > do echo -n "$i: "; GetFileInfo -c "$i" > done Desktop: Desktop is a directory and has no creator Documents: Documents is a directory and has no creator ... aa: "" ab: "" ... grep 719 grep grep [options] pattern [file-list] The grep utility searches one or more text files, line by line, for a pattern, which can be a simple string or another form of a regular expression. The grep utility takes various actions, specified by options, each time it finds a line that contains a match for the pattern. This utility takes its input either from files you specify on the command line or from standard input. Arguments The pattern is a regular expression, as defined in Appendix A. You must quote regular expressions that contain special characters, SPACE s, or TAB s. An easy way to quote these characters is to enclose the entire expression within single quotation marks. The file-list is a list of the pathnames of ordinary text files that grep searches. With the –r option, file-list may contain directories whose contents are searched. Options Without any options grep sends lines that contain a match for pattern to standard output. When you specify more than one file on the command line, grep precedes each line it displays with the name of the file it came from, followed by a colon. Major Options You can use only one of the following three options at a time. Normally you do not need to use any, because grep defaults to –G, which is regular grep. –E (extended) Interprets pattern as an extended regular expression (page 895). The command grep –E is the same as egrep. See the "Notes" section. –F (fixed) Interprets pattern as a fixed string of characters. The command grep –F is the same as fgrep. –G (grep) Interprets pattern as a basic regular expression. This is the default major option if you do not specify a major option. Other Options The grep utility accepts the common options described on page 603. The Mac OS X version of grep accepts long options tip Options for grep preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––count –c Displays only the number of lines that contain a match in each file. ––context=n –C n Displays n lines of context around each matching line. grep Searches for a pattern in files 720 grep ––file=file –f file Reads file, which contains one pattern per line, and finds lines in the input that match each of the patterns. ––no-filename –h Does not display the filename at the beginning of each line when searching through multiple files. ––ignore-case –i Causes lowercase letters in the pattern to match uppercase letters in the file, and vice versa. Use this option when you are searching for a word that may appear at the beginning of a sentence (that is, the word may or may not start with an uppercase letter). ––files-with-matches –l (lowercase "l"; list) Displays only the name of each file that contains one or more matches. A filename is displayed only once, even if the file contains more than one match. ––max-count=n –m n Stops reading each file, or standard input, after displaying n lines containing matches. ––line-number ––quiet or ––silent –n Precedes each line by its line number in the file. The file does not need to contain line numbers. –q Does not write anything to standard output; only sets the exit code. ––recursive –r or –R Recursively descends directories in the file-list and processes files within these directories. ––no-messages –s (silent) Does not display an error message if a file in the file-list does not exist or is not readable. ––invert-match –v Causes lines not containing a match to satisfy the search. When you use this option by itself, grep displays all lines that do not contain a match for the pattern. ––word-regexp –w With this option, the pattern must match a whole word. This option is helpful if you are searching for a specific word that may also appear as a substring of another word in the file. ––line-regexp Notes egrep and fgrep –x The pattern matches whole lines only. The grep utility returns an exit status of 0 if it finds a match, 1 if it does not find a match, and 2 if the file is not accessible or the grep command contains a syntax error. Two utilities perform functions similar to that of grep. The egrep utility (same as grep –E) allows you to use extended regular expressions (page 895), which include a different set of special characters than basic regular expressions (page 893). The grep 721 fgrep utility (same as grep –F) is fast and compact but processes only simple strings, not regular expressions. GNU grep, which runs under Linux and Mac OS X, uses extended regular expressions in place of regular expressions. Thus egrep is virtually the same as grep. Refer to the grep info page for a minimal distinction. Examples The following examples assume that the working directory contains three files: testa, testb, and testc. File testa File testb File testc aaabb bbbcc ff-ff cccdd dddaa aaaaa bbbbb ccccc ddddd AAAAA BBBBB CCCCC DDDDD The grep utility can search for a pattern that is a simple string of characters. The following command line searches testa and displays each line containing the string bb: $ grep bb testa aaabb bbbcc The –v option reverses the sense of the test. The following example displays the lines in testa without bb: $ grep -v bb testa ff-ff cccdd dddaa The –n option displays the line number of each displayed line: $ grep -n bb testa 1:aaabb 2:bbbcc The grep utility can search through more than one file. Here it searches through each file in the working directory. The name of the file containing the string precedes each line of output. $ grep bb * testa:aaabb testa:bbbcc testb:bbbbb When the search for the string bb is done with the –w option, grep produces no output because none of the files contains the string bb as a separate word: $ grep -w bb $ * 722 grep The search grep performs is case sensitive. Because the previous examples specified lowercase bb, grep did not find the uppercase string BBBBB in testc. The –i option causes both uppercase and lowercase letters to match either case of letter in the pattern: $ grep -i bb testa:aaabb testa:bbbcc testb:bbbbb testc:BBBBB $ grep -i BB testa:aaabb testa:bbbcc testb:bbbbb testc:BBBBB * * The –c option displays the number of lines in each file that contain a match: $ grep -c bb testa:2 testb:1 testc:0 * The –f option finds matches for each pattern in a file of patterns. In the next example, gfile holds two patterns, one per line, and grep searches for matches to the patterns in gfile: $ cat gfile aaa bbb $ grep -f gfile test* testa:aaabb testa:bbbcc testb:aaaaa testb:bbbbb The following command line searches text2 and displays lines that contain a string of characters starting with st, followed by zero or more characters (.* represents zero or more characters in a regular expression—see Appendix A), and ending in ing: $ grep 'st.*ing' text2 ... The ^ regular expression, which matches the beginning of a line, can be used alone to match every line in a file. Together with the –n option, ^ can be used to display the lines in a file, preceded by their line numbers: $ grep -n '^' testa 1:aaabb 2:bbbcc 3:ff-ff 4:cccdd 5:dddaa grep 723 The next command line counts the number of times #include statements appear in C source files in the working directory. The –h option causes grep to suppress the filenames from its output. The input to sort consists of all lines from *.c that match #include. The output from sort is an ordered list of lines that contains many duplicates. When uniq with the –c option processes this sorted list, it outputs repeated lines only once, along with a count of the number of repetitions in its input. $ 9 2 1 6 2 2 2 3 grep -h '#include' *.c | sort | uniq -c #include "buff.h" #include "poly.h" #include "screen.h" #include "window.h" #include "x2.h" #include "x3.h" #include #include The final command calls the vim editor with a list of files in the working directory that contain the string Sampson. The $(...) command substitution construct (page 340) causes the shell to execute grep in place and supply vim with a list of filenames that you want to edit: $ vim $(grep -l 'Sampson' ... *) The single quotation marks are not necessary in this example, but they are required if the regular expression you are searching for contains special characters or SPACEs. It is generally a good habit to quote the pattern so the shell does not interpret any special characters the pattern may contain. 724 gzip gzip Compresses or decompresses files gzip [options] [file-list] gunzip [options] [file-list] zcat [file-list] The gzip utility compresses files, the gunzip utility restores files compressed with gzip, and the zcat utility displays files compressed with gzip. Arguments The file-list is a list of the names of one or more files that are to be compressed or decompressed. If a directory appears in file-list with no ––recursive option, gzip/gunzip issues an error message and ignores the directory. With the ––recursive option, gzip/gunzip recursively compresses/decompresses files within the directory hierarchy. If file-list is empty or if the special option – (hyphen) is present, gzip reads from standard input. The ––stdout option causes gzip and gunzip to write to standard output. The information in this section also applies to gunzip, a link to gzip. Options The gzip, gunzip, and zcat utilities accept the common options described on page 603. The Mac OS X versions of gzip, gunzip, and zcat accept long options tip Options for gzip, gunzip, and zcat preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––stdout –c Writes the results of compression or decompression to standard output instead of to filename.gz. ––decompress or ––uncompress –d Decompresses a file compressed with gzip. This option with gzip is equivalent to the gunzip command. ––force –f Overwrites an existing output file on compression/decompression. ––list –l For each compressed file in file-list, displays the file's compressed and decompressed sizes, the compression ratio, and the name of the file before compression. Use this option with ––verbose to display additional information. ––fast or ––best –n Controls the tradeoff between the speed of compression and the amount of compression. The n argument is a digit from 1 to 9; level 1 is the fastest (least) compression and level 9 is the best (slowest and most) compression. The default level is 6. The options ––fast and ––best are synonyms for –1 and –9, respectively. ––quiet ––recursive –q Suppresses warning messages. –r Recursively descends directories in file-list and compresses/decompresses files within these directories. gzip 725 ––test –t Verifies the integrity of a compressed file. This option displays nothing if the file is OK. ––verbose –v Displays the name of the file, the name of the compressed file, and the amount of compression as each file is processed. Discussion Compressing files reduces disk space requirements and shortens the time needed to transmit files between systems. When gzip compresses a file, it adds the extension .gz to the filename. For example, compressing the file fname creates the file fname.gz and, unless you use the ––stdout (–c) option, deletes the original file. To restore fname, use the command gunzip with the argument fname.gz. Almost all files become much smaller when compressed with gzip. On rare occasions a file will become larger, but only by a slight amount. The type of a file and its contents (as well as the –n option) determine how much smaller a file becomes; text files are often reduced by 60 to 70 percent. The attributes of a file, such as its owner, permissions, and modification and access times, are left intact when gzip compresses and gunzip decompresses a file. If the compressed version of a file already exists, gzip reports that fact and asks for your confirmation before overwriting the existing file. If a file has multiple links to it, gzip issues an error message and exits. The ––force option overrides the default behavior in both of these situations. Notes Without the ––stdout (–c) option, gzip removes the files in file-list. The bzip2 utility (page 615) compresses files more efficiently than does gzip. In addition to the gzip format, gunzip recognizes several other compression formats, enabling gunzip to decompress files compressed with compress. To see an example of a file that becomes larger when compressed with gzip, compare the size of a file that has been compressed once with the same file compressed with gzip again. Because gzip complains when you give it an argument with the extension .gz, you need to rename the file before compressing it a second time. The tar utility with the –z modifier (page 848) calls gzip. The following

related utilities display and manipulate compressed files. None of these utilities changes the files it works on. zcat file-list Works like cat except that it uses gunzip to decompress file-list as it copies files to standard output. zdiff [options] file1 [file2] Works like diff (page 663) except file1 and file2 are decompressed with gunzip as needed. The zdiff utility accepts the same options as diff. If you omit file2, zdiff compares file1 with the compressed version of file1. zless file-list Works like less except that it uses gunzip to decompress file-list as it displays files. 726 gzip Examples In the first example, gzip compresses two files. Next gunzip decompresses one of the files. When a file is compressed and decompressed, its size changes but its modification time remains the same: $ ls -l total 175 -rw-rw-r-- 1 max group 33557 Jul 20 17:32 patch-2.0.7 -rw-rw-r-- 1 max group 143258 Jul 20 17:32 patch-2.0.8 $ gzip * $ ls -l total 51 -rw-rw-r-- 1 max group 9693 Jul 20 17:32 patch-2.0.7.gz -rw-rw-r-- 1 max group 40426 Jul 20 17:32 patch-2.0.8.gz $ gunzip patch-2.0.7.gz $ ls -l total 75 -rw-rw-r-- 1 max group 33557 Jul 20 17:32 patch-2.0.7 -rw-rw-r-- 1 max group 40426 Jul 20 17:32 patch-2.0.8.gz In the next example, the files in Sam's home directory are archived using cpio (page 644). The archive is compressed with gzip before it is written to the device mounted on /dev/sde1. $ find ~sam -depth -print | cpio -oBm | gzip >/dev/sde1 head 727 Displays the beginning of a file head [options] [file-list] The head utility displays the beginning (head) of a file. This utility takes its input either from one or more files you specify on the command line or from standard input. Arguments The file-list is a list of the pathnames of the files that head displays. When you specify more than one file, head displays the filename before displaying the first few lines of each file. When you do not specify a file, head takes its input from standard input. Options Under Linux, head accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––bytes=n[u] –c n[u] Displays the first n bytes (characters) of a file. Under Linux only, the u argument is an optional multiplicative suffix as described on page 602, except that head uses a lowercase k for kilobyte (1,024-byte blocks) and accepts b for 512-byte blocks. If you include a multiplicative suffix, head counts by this unit instead of by bytes. ––lines=n –n n Displays the first n lines of a file. You can use –n to specify n lines without using the lines keyword or the –n option. If you specify a negative value for n, head displays all but the last n lines of the file. ––quiet –q Suppresses header information when you specify more than one filename on the command line. L Notes The head utility displays the first ten lines of a file by default. Examples The examples in this section are based on the following file: $ cat eleven line one line two line three line four line five line six line seven line eight line nine line ten line eleven head head 728 head Without any arguments head displays the first ten lines of a file: $ head eleven line one line two line three line four line five line six line seven line eight line nine line ten The next example displays the first three lines (–n 3) of the file: $ head -n 3 eleven line one line two line three The following example is equivalent to the preceding one: $ head -3 eleven line one line two line three The next example displays the first six characters (–c 6) in the file: $ head -c 6 eleven line o$ The final example displays all but the last seven lines of the file: $ head -n -7 eleven line one line two line three line four kill 729 kill kill [option] PID-list kill –l [signal-name | signal-number] The kill utility sends a signal to one or more processes. Typically this signal terminates the processes. For kill to work, the processes must belong to the user executing kill, with one exception: A user working with root privileges can terminate any process. The –l (lowercase "l") option lists information about signals. Arguments Options The PID-list is a list of process identification (PID) numbers of processes that kill is to terminate. –l (list) Without an argument, displays a list of signals. With an argument of a signal-name, displays the corresponding signal-number. With an argument of a signal-number, displays the corresponding signal-name. –signal-name | –signal-number Sends the signal specified by signal-name or signal-number to PID-list. You can specify a signal-name preceded by SIG or not (e.g., SIGKILL or KILL). Without this option, kill sends a software termination signal (SIGTERM; signal number 15). Notes See also killall on page 731. See Table 10-5 on page 453 for a list of signals. The command kill –l displays a complete list of signal numbers and names. In addition to the kill utility, a kill builtin is available in the Bourne Again and TC Shells. The builtins work similarly to the utility described here. Give the command /bin/kill to use the kill utility and the command kill to use the builtin. It does not usually matter which version you call. The shell displays the PID number of a background process when you initiate the process. You can also use the ps utility to determine PID numbers. If the software termination signal does not terminate a process, try sending a KILL signal (signal number 9). A process can choose to ignore any signal except KILL. The kill utility/builtin accepts job identifiers in place of the PID-list. Job identifiers consist of a percent sign (%) followed by either a job number or a string that uniquely identifies the job. kill Terminates a process by PID 730 kill To terminate all processes that the current login process initiated and have the operating system log you out, give the command kill –9 0. root: do not run kill with arguments of –9 0 or KILL 0 caution If you run the command kill –9 0 while you are working with root privileges, you will bring the system down. Examples The first example shows a command line executing the file compute as a background process and the kill utility terminating it: $ compute & [2] 259 $ kill 259 $ RETURN [2]+ Terminated compute The next example shows the ps utility determining the PID number of the background process running a program named xprog and the kill utility terminating xprog with the TERM signal: $ ps PID TTY 7525 pts/1 14668 pts/1 14699 pts/1 TIME CMD 00:00:00 bash 00:00:00 xprog 00:00:00 ps $ kill -TERM 14668 $ killall 731 Terminates a process by name killall [option] name-list The killall utility sends a signal to one or more processes executing specified commands. Typically this signal terminates the processes. For killall to work, the processes must belong to the user executing killall, with one exception: A user working with root privileges can terminate any process. Arguments The name-list is a SPACE-separated list of names of programs that are to receive signals. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––interactive ––list ––quiet –i Prompts for confirmation before killing a process. L –l Displays a list of signals (but kill –l displays a better list). With this option killall does not accept a name-list. –q Does not display a message if killall fails to terminate a process. L –signal-name | –signal-number Sends the signal specified by signal-name or signal-number to name-list. You can specify a signal-name preceded by SIG or not (e.g., SIGKILL or KILL). Without this option, kill sends a software termination signal (SIGTERM; signal number 15). Notes See also kill on page 729. See Table 10-5 on page 453 for a list of signals. The command kill –l displays a complete list of signal numbers and names. If the software termination signal does not terminate the process, try sending a KILL signal (signal number 9). A process can choose to ignore any signal except KILL. You can use ps (page 796) to determine the name of the program you want to terminate. Examples You can give the following commands to experiment with killall: $ sleep 60 & [1] 23274 $ sleep 50 & [2] 23275 $ sleep 40 & [3] 23276 killall killall 732 killall $ sleep 120 & [4] 23277 $ killall sleep $ RETURN [1] Terminated [2] Terminated [3]- Terminated [4]+ Terminated sleep sleep sleep sleep 60 50 40 120 The next command, run by a user with root privileges, terminates all instances of the Firefox browser: # killall firefox 733 launchctl O Controls the launchd daemon launchctl [command [options] [arguments]] The launchctl utility controls the launchd daemon. Arguments The command is the command that launchctl sends to launchd. Table V-17 lists some of the commands and the options and arguments each command accepts. Without a command, launchctl reads commands, options, and arguments from standard input, one set per line. Without a command, when standard input comes from the keyboard, launchctl runs interactively. Table V-17 Option launchctl commands Command Argument Description help None Displays a help message list None Lists jobs loaded into launchd load [–w] Job configuration file Loads the job named by the argument shutdown None Prepares for shutdown by removing all jobs start Job name Starts the job named by the argument stop Job name Stops the job named by the argument unload [–w] Job configuration file Unloads the job named by the argument Only the load and unload commands take an option. –w (write) When loading a file, removes the Disabled key and saves the modified configuration file. When unloading a file, adds the Disabled key and saves the modified configuration file. Discussion The launchctl utility is the user interface to launchd, which manages system daemons and background tasks (called jobs). Each job is described by a job configuration file, which is a property list file in the format defined by the launchd.plist man page. For security reasons, users not working with root privileges cannot communicate with the system's primary launchd process, PID 1. When such a user loads jobs, OS X creates a new instance of launchd for that user. When all its jobs are unloaded, that instance of launchd quits. Notes The launchctl utility and launchd daemon were introduced in Mac OS X version 10.4. Under version 10.3 and earlier, system jobs were managed by init, xinetd, and cron. launchctl O launchctl O 734 launchctl O Examples The first example, which is run by a user with root privileges, uses the list command to list launchd jobs running on the local system: # launchctl list PID Status Label 51479 0x109490.launchctl 50515 0x10a780.bash 50514 0x10a680.sshd 50511 0x108d20.sshd 22 0x108bc0.securityd 0 com.apple.launchctl.StandardIO 37057 [0x0-0x4e84e8].com.apple.ScreenSaver.Engine 27860 0x10a4e0.DiskManagementTo 27859 [0x0-0x3a23a2].com.apple.SoftwareUpdate ... The next example enables the ntalk service. Looking at the ntalk.plist file before and after the launchctl command is executed shows that launchctl has modified the file by removing the Disabled key. # cat /System/Library/LaunchDaemons/ntalk.plist ... Disabled Label com.apple.ntalkd ... # launchctl load -w /System/Library/LaunchDaemons/ntalk.plist # cat /System/Library/LaunchDaemons/ntalk.plist ... Label com.apple.ntalkd ... Without any arguments, launchctl prompts for commands on standard input. Give a quit command or press CONTROL-D to exit from launchctl. In the last example, a user running with root privileges causes launchctl to display a list of jobs and then to stop the job that would launch airportd: # launchctl launchd% list PID Status Label 8659 0x10ba10.cron 1 0x10c760.launchd ... 0 com.apple.airport.updateprefs 0 com.apple.airportd 0 com.apple.AirPort.wps 0 0x100670.dashboardadvisoryd 0 com.apple.launchctl.System launchd% stop com.apple.airportd launchd% quit less 735 less less [options] [file-list] The less utility displays text files, one screen at a time. Arguments The file-list is the list of files you want to view. If there is no file-list, less reads from standard input. Options The less utility accepts the common options described on page 603. The Mac OS X version of less accepts long options tip Options for less preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––clear-screen –c Repaints the screen from the top line down instead of scrolling. ––QUIT-AT-EOF –E (exit) Normally less requires you to enter q to terminate. This option exits automatically the first time less reads the end of file. ––quit-at-eof –e (exit) Similar to –E, except that less exits automatically the second time it reads the end of file. ––quit-if-one-screen –F Displays the file and quits if the file can be displayed on a single screen. –i Causes a search for a string of lowercase letters to match both uppercase and lowercase letters. This option is ignored if you specify a pattern that includes uppercase letters. ––ignore-case ––IGNORE-CASE –I Causes a search for a string of letters of any case to match both uppercase and lowercase letters, regardless of the case of the search pattern. ––long-prompt –m Each prompt reports the percentage of the file you have viewed. This option causes less to display a prompt similar to the prompt used by more. It reports byte numbers when less reads from standard input because less has no way of determining the size of the input file. ––LINE-NUMBERS –N Displays a line number at the beginning of each line. ––prompt=prompt –Pprompt Changes the short prompt string (the prompt that appears at the bottom of each screen of output) to prompt. Enclose prompt in quotation marks if it contains SPACEs. You can include special symbols in prompt, which less will replace less Displays text files, one screen at a time 736 less with other values when it displays the prompt. For example, less displays the current filename in place of %f in prompt. See the less man page for a list of these special symbols and descriptions of other prompts. Custom prompts are useful if you are running less from within another program and want to give instructions or information to the person using the program. The default prompt is the name of the file displayed in reverse video. ––squeeze-blank-lines –s Displays multiple, adjacent blank lines as a single blank line. When you use less to display text that has been formatted for printing with blank space at the top and bottom of each page, this option shortens these headers and footers to a single line. ––tabs=n –xn Sets tab stops n characters apart. The default is eight characters. ––window=n –[z]n Sets the scrolling size to n lines. The default is the height of the display in lines. Each time you move forward or backward a page, you move n lines. The z part of the option maintains compatibility with more and can be omitted. +command Any command you can give less while it is running can also be given as an option by preceding it with a plus sign (+) on the command line. See the "Commands" section. A command preceded by a plus sign on the command line is executed as soon as less starts and applies to the first file only. ++command Similar to +command except that command is applied to every file in file-list, not just the first file. Notes The phrase "less is more" explains the origin of the name of this utility. The more utility is the original Berkeley UNIX pager (also available under Linux). The less utility is similar to more but includes many enhancements. (Under OS X, less and more are copies of the same file.) After displaying a screen of text, less displays a prompt and waits for you to enter a command. You can skip forward and backward in the file, invoke an editor, search for a pattern, or perform a number of other tasks. See the v command in the next section for information on how you can edit the file you are viewing with less. You can set the options to less either from the command line when you call less or by setting the LESS environment variable. For example, you can use the following command from bash to use less with the –x4 and –s options: $ export LESS="-x4 -s" less 737 Normally you would set LESS in ~/.bash_profile if you are using bash or in ~/.login if you are using tcsh. Once you have set the LESS variable, less is invoked with the specified options each time you call it. Any options you give on the command line override the settings in the LESS variable. The LESS variable is used both when you call less from the command line and when less is invoked by another program, such as man. To specify less as the pager to use with man and other programs, set the environment variable PAGER to less. For example, with bash you can add the following line to ~/.bash_profile: export PAGER=less Commands Whenever less pauses, you can enter any of a large number of commands. This section describes some commonly used commands. Refer to the less man page for the full list of commands. The optional numeric argument n defaults to 1, with the exceptions noted. You do not need to follow these commands with a RETURN. nb or nCONTROL-B (backward) Scrolls backward n lines. The default value of n is the height of the screen in lines. nd or nCONTROL-D (down) Scrolls forward n lines. The default value of n is one-half the height of the screen in lines. When you specify n, it becomes the new default value for this command. F (forward) Scrolls forward. If the end of the input is reached, this command waits for more input and then continues scrolling. This command allows you to use less in a manner similar to tail –f (page 843), except that less paginates the output as it appears. ng (go) Goes to line number n. This command may not work if the file is read from standard input and you have moved too far into the file. The default value of n is 1. h or H (help) Displays a summary of all available commands. The summary is displayed using less, as the list of commands is quite long. nRETURN or nj (jump) Scrolls forward n lines. The default value of n is 1. q or :q Terminates less. nu or nCONTROL-U Scrolls backward n lines. The default value of n is one-half the height of the screen in lines. When you specify n, it becomes the default value for this command. v Brings the current file into an editor with the cursor on the current line. The less utility uses the editor specified in the EDITOR environment variable. If EDITOR is not set, less uses vi (which is typically linked to vim). nw Scrolls backward like nb, except that the value of n becomes the new default value for this command. ny or nk Scrolls backward n lines. The default value of n is 1. nz Displays the next n lines like nSPACE except that the value of n, if present, becomes the new default value for the z and SPACE commands. 738 less nSPACE Displays the next n lines. Pressing the SPACE bar by itself displays the next screen of text. /regular-expression Skips forward in the file, looking for lines that contain a match for regular-expression. If you begin regular-expression with an exclamation point (!), this command looks for lines that do not contain a match for regular-expression. If regular-expression begins with an asterisk (*), this command continues the search through file-list. (A normal search stops at the end of the current file.) If regular-expression begins with an at sign (@), this command begins the search at the beginning of file-list and continues to the end of file-list. ?regular-expression This command is similar to the previous one but searches backward through the file (and file-list). An asterisk (*) as the first character in regular-expression causes the search to continue backward through file-list to the beginning of the first file. An at sign (@) causes the search to start with the last line of the last file in file-list and progress toward the first line of the first file. { or (or [If one of these characters appears in the top line of the display, this command scrolls forward to the matching right brace, parenthesis, or bracket. For example, typing { causes less to move the cursor forward to the matching }. } or) or] Similar to the preceding commands, these commands move the cursor backward to the matching left brace, parenthesis, or bracket. CONTROL-L Redraws the screen. This command is useful if the text on the screen has become garbled. [n]:n Skips to the next file in file-list. If n is given, skips to the nth next file in file-list. ![command line] Executes command line under the shell specified by the SHELL environment variable or under sh (usually linked to or a copy of bash) by default. A percent sign (%) in command line is replaced by the name of the current file. If you omit command line, less starts an interactive shell. Examples The following example displays the file memo.txt. To see more of the file, the user presses the SPACE bar in response to the less prompt at the lower-left corner of the screen: $ less memo.txt ... memo.txt SPACE ... In the next example, the user changes the prompt to a more meaningful message and uses the –N option to display line numbers. Finally the user instructs less to skip forward to the first line containing the string procedure. less 739 $ less -Ps"Press SPACE to continue, q to quit" -N +/procedure ncut.icn 28 procedure main(args) 29 local filelist, arg, fields, delim 30 31 filelist:=[] ... 45 # Check for real field list 46 # 47 if /fields then stop("-fFIELD_LIST is required.") 48 49 # Process the files and output the fields Press SPACE to continue, q to quit 740 ln ln Makes a link to a file ln ln [options] existing-file [new-link] ln [options] existing-file-list directory The ln utility creates hard or symbolic links to one or more files. You can create a symbolic link, but not a hard link, to a directory. Arguments In the first format the existing-file is the pathname of the file you want to create a link to. The new-link is the pathname of the new link. When you are creating a symbolic link, the existing-file can be a directory. If you omit new-link, ln creates a link to existing-file in the working directory, and uses the same simple filename as existing-file. In the second format the existing-file-list is a list of the pathnames of the ordinary files you want to create links to. The ln utility establishes the new links in the directory. The simple filenames of the entries in the directory are the same as the simple filenames of the files in the existing-file-list. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––backup –b If the ln utility will remove a file, this option makes a backup by appending a tilde (~) to the filename. This option works only with ––force. L ––force –f Normally ln does not create the link if new-link already exists. This option removes new-link before creating the link. When you use ––force and ––backup together (Linux only), ln makes a copy of new-link before removing it. ––interactive –i If new-link already exists, this option prompts you before removing new-link. If you enter y or yes, ln removes new-link before creating the link. If you answer n or no, ln does not remove new-link and does not make a new link. ––symbolic –s Creates a symbolic link. When you use this option, the existing-file and the new-link may be directories and may reside on different filesystems. Refer to "Symbolic Links" on page 108. Notes For more information refer to "Links" on page 104. The ls utility with the –l option displays the number of hard links to a file (Figure 4-12, page 94). Hard links By default ln creates hard links. A hard link to a file is indistinguishable from the original file. All hard links to a file must be in the same filesystem. For more information refer to "Hard Links" on page 106. Symbolic links You can also use ln to create symbolic links. Unlike a hard link, a symbolic link can exist in a different filesystem from the linked-to file. Also, a symbolic link can point to a directory. For more information refer to "Symbolic Links" on page 108. ln 741 If new-link is the name of an existing file, ln does not create the link unless you use the ––force option (Linux only) or answer yes when using the –i (––interactive) option. Examples The following command creates a link between memo2 in the literature subdirectory of Zach's home directory and the working directory. The file appears as memo2 (the simple filename of the existing file) in the working directory: $ ln ~zach/literature/memo2 . You can omit the period that represents the working directory from the preceding command. When you give a single argument to ln, it creates a link in the working directory. The next command creates a link to the same file. This time the file appears as new_memo in the working directory: $ ln ~zach/literature/memo2 new_memo The following command creates a link that causes the file to appear in Sam's home directory: $ ln ~zach/literature/memo2 ~sam/new_memo You must have write and execute access permissions to the other user's directory for this command to work. If you own the file, you can use chmod to give the other user write access permission to the file. The next command creates a symbolic link to a directory. The ls –ld command shows the link: $ ln -s /usr/local/bin bin $ ls -ld bin lrwxrwxrwx 1 zach zach 14 Feb 10 13:26 bin -> /usr/local/bin The final example attempts to create a symbolic link named memo1 to the file memo2. Because the file memo1 exists, ln refuses to make the link. When you use the –i (––interactive) option, ln asks whether you want to replace the existing memo1 file with the symbolic link. If you enter y or yes, ln creates the link and the old memo1 disappears. $ ls -l memo? -rw-rw-r-1 zach group -rw-rw-r-1 zach group $ ln -s memo2 memo1 ln: memo1: File exists $ ln -si memo2 memo1 ln: replace 'memo1'? y $ ls -l memo? lrwxrwxrwx 1 zach group -rw-rw-r-1 zach group 224 Jul 31 14:48 memo1 753 Jul 31 14:49 memo2 5 Jul 31 14:49 memo1 -> memo2 753 Jul 31 14:49 memo2 Under Linux you can also use the ––force option to cause ln to overwrite a file. 742 lpr lpr Sends files to printers lpr lpr [options] [file-list] lpq [options] [job-identifiers] lprm [options] [job-identifiers] The lpr utility places one or more files into a print queue, providing orderly access to printers for several users or processes. This utility can work with printers attached to remote systems. You can use the lprm utility to remove files from the print queues and the lpq utility to check the status of files in the queues. Refer to "Notes" later in this section. Arguments The file-list is a list of one or more filenames for lpr to print. Often these files are text files, but many systems are configured so lpr can accept and properly print a variety of file types including PostScript and PDF files. Without a file-list, lpr accepts input from standard input. The job-identifiers is a list of job numbers or usernames. If you do not know the job number, use lpq to display a list of print jobs. Options Some of the following options depend on which type of file is being printed as well as on how the system is configured for printing. –h (no header) Suppresses printing of the header (burst) page. This page is useful for identifying the owner of the output in a multiuser setup, but printing it is a waste of paper when this identification is not needed. –l (lowercase "l") Specifies that lpr should not preprocess (filter) the file being printed. Use this option when the file is already formatted for the printer. –P printer Routes the print jobs to the queue for the printer named printer. If you do not use this option, print jobs are routed to the default printer for the local system. The acceptable values for printer are found in the Linux file /etc/printcap and can be displayed by an lpstat –t command. These values vary from system to system. –r (remove) Deletes the files in file-list after calling lpr. –# n Prints n copies of each file. Depending on which shell you are using, you may need to escape the # by preceding it with a backslash to keep the shell from interpreting it as a special character. lpr 743 Discussion The lpr utility takes input either from files you specify on the command line or from standard input; it adds these files to the print queue as print jobs. The utility assigns a unique identification number to each print job. The lpq utility displays the job numbers of the print jobs that lpr has set up; you can use the lprm utility to remove a job from the print queue. lpq The lpq utility displays information about jobs in a print queue. When called without any arguments, lpq lists all print jobs queued for the default printer. Use lpr's –P printer option with lpq to look at other print queues—even those for printers connected to remote systems. With the –l option, lpq displays more information about each job. If you give a username as an argument, lpq displays only the printer jobs belonging to that user. lprm One item displayed by lpq is the job number for each print job in the queue. To remove a job from the print queue, give the job number as an argument to lprm. Unless you are working with root privileges, you can remove only your own jobs. Even a user working with root privileges may not be able to remove a job from a queue for a remote printer. If you do not give any arguments to lprm, it removes the active printer job (that is, the job that is now printing) from the queue, if you own that job. Notes If you normally use a printer other than the system default printer, you can set up lpr to use another printer as your personal default by assigning the name of this printer to the environment variable PRINTER. For example, if you use bash, you can add the following line to ~/.bash_profile to set your default printer to the printer named ps: export PRINTER=ps LPD and LPR Traditionally, UNIX had two printing systems: the BSD Line Printer Daemon (LPD) and the System V Line Printer system (LPR). Linux adopted those systems at first, and both UNIX and Linux have seen modifications to and replacements for these systems. Today CUPS is the default printing system under Linux and OS X. CUPS CUPS is a cross-platform print server built around the Internet Printing Protocol (IPP), which is based on HTTP. CUPS provides a number of printer drivers and can print different types of files, including PostScript files. CUPS provides System V and BSD command-line interfaces and, in addition to IPP, supports LPD/LPR, HTTP, SMB, and JetDirect (socket) protocols, among others. This section describes the LPD command-line interface that runs under CUPS and also in native mode on older systems. Examples The first command sends the file named memo2 to the default printer: $ lpr memo2 744 lpr Next a pipe sends the output of ls to the printer named deskjet: $ ls | lpr -Pdeskjet The next example paginates and sends the file memo to the printer: $ pr -h "Today's memo" memo | lpr The next example shows a number of print jobs queued for the default printer. Max owns all the jobs, and the first one is being printed (it is active). Jobs 635 and 639 were created by sending input to lpr's standard input; job 638 was created by giving ncut.icn as an argument to the lpr command. The last column gives the size of each print job. $ lpq deskjet is ready Rank Owner active max 1st max 2nd max and printing Job Files 635 (stdin) 638 ncut.icn 639 (stdin) Total Size 38128 bytes 3587 bytes 3960 bytes The next command removes job 638 from the default print queue: $ lprm 638 ls 745 ls ls Displays information about one or more files ls [options] [file-list] The ls utility displays information about one or more files. It lists the information alphabetically by filename unless you use an option that changes the order. Arguments When you do not provide an argument, ls displays the names of the visible files (those with filenames that do not begin with a period) in the working directory. The file-list is a list of one or more pathnames of any ordinary, directory, or device files. It can include ambiguous file references. When the file-list includes a directory, ls displays the contents of the directory. It displays the name of the directory only when needed to avoid ambiguity, such as when the listing includes more than one directory. When you specify an ordinary file, ls displays information about that one file. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. The options determine the type of information ls displays, the manner in which it displays the information, and the order in which the information is displayed. When you do not use an option, ls displays a short list that contains just the names of files, in alphabetical order. ––almost-all –A The same as –a but does not list the . and .. directory entries. ––all –a Includes hidden filenames (those filenames that begin with a period; page 82) in the listing. Without this option ls does not list information about hidden files unless you include the name of a hidden file in the file-list. The * ambiguous file reference does not match a leading period in a filename (page 138), so you must use this option or explicitly specify a filename (ambiguous or not) that begins with a period to display hidden files. ––escape –b Displays nonprinting characters in a filename, using backslash escape sequences similar to those used in C language strings (Table V-18). Other nonprinting characters are displayed as a backslash followed by an octal number. Table V-18 Backslash escape sequences Sequence Meaning \b BACKSPACE NEWLINE 746 ls Table V-18 Backslash escape sequences (continued) Sequence Meaning \r RETURN \t HORIZONTAL TAB \v VERTICAL TAB \\ BACKSLASH ––color[=when] The ls utility can display various types of files in different colors but normally does not use colors (the same result as when you specify when as none). If you do not specify when or if you specify when as always, ls uses colors. When you specify when as auto, ls uses colors only when the output goes to a screen. See the "Notes" section for more information. L ––directory –d Displays directories without displaying their contents. This option does not dereference symbolic links; that is, for each file that is a symbolic link, this option lists the symbolic link, not the file the link points to. –e Displays ACLs (page 934). O ––format=word By default ls displays files sorted vertically. This option sorts files based on word: across or horizontal (also –x), separated by commas (also –m), long (also –l), or single-column (also –1). L ––classify –F Displays a slash (/) after each directory, an asterisk (*) after each executable file, and an at sign (@) after a symbolic link. ––dereference-command-line –H (partial dereference) For each file that is a symbolic link, lists the file the link points to, not the symbolic link itself. This option affects files specified on the command line; it does not affect files found while descending a directory hierarchy. This option treats files that are not symbolic links normally. See page 623 for an example of the use of the –H versus –L options. ––human-readable –h With the –l option, displays sizes in K (kilobyte), M (megabyte), and G (gigabyte) blocks, as appropriate. This option works with the –l option only. It displays powers of 1,024. Under Mac OS X, it displays B (bytes) in addition to the preceding suffixes. See also ––si. ––inode –i Displays the inode number of each file. With the –l option, this option displays the inode number in column 1 and shifts other items one column to the right. ––dereference –L (dereference) For each file that is a symbolic link, lists the file the link points to, not the symbolic link itself. This option affects all files and treats files that are not symbolic links normally. See page 623 for an example of the use of the –H versus –L options. ls 747 –l (lowercase "l") Lists more information about each file. This option does not dereference symbolic links; that is, for each file that is a symbolic link, this option lists the symbolic link, not the file the link points to. If standard output for a directory listing is sent to the screen, this option displays the number of blocks used by all files in the listing on a line before the listing. Use this option with –h to make file sizes more readable. See the "Discussion" section for more information. ––format=long ––format=commas –m Displays a comma-separated list of files that fills the width of the screen. –P (no dereference) For each file that is a symbolic link, lists the symbolic link, not the file the link points to. This option affects all files and treats files that are not symbolic links normally. See page 625 for an example of the use of the –P option. O ––hide-control-chars –q Displays nonprinting characters in a filename as question marks. When standard output is sent to the screen, this behavior is the default. Without this option, when standard output is sent to a filter or a file, nonprinting characters are displayed as themselves. ––recursive ––reverse ––size –R Recursively lists directory hierarchies. –r Displays the list of filenames in reverse sorted order. –s Displays the number of 1,024-byte (Linux) or 512-byte (Mac OS X) blocks allocated to the file. The size precedes the filename. With the –l option this option displays the size in column 1 and shifts other items one column to the right. If standard output for a directory listing is sent to the screen, this option displays the number of blocks used by all files in the listing on a line before the listing. You can include the –h option to make the file sizes easier to read. Under Mac OS X, you can use the BLOCKSIZE environment variable (page 603) to change the size of the blocks this option reports on. ––si ––sort=time With the –l option, displays sizes in K (kilobyte), M (megabyte), and G (gigabyte) blocks, as appropriate. This option works with the –l option only. This option displays powers of 1,000. See also ––human-readable. L –t Displays files sorted by the time they were last modified. ––sort=word By default ls displays files in ASCII order. This option sorts the files based on word: filename extension (–X; Linux only), none (–U; Linux only), file size (–S), access time (–u), or modification time (–t). See ––time for an exception. L ––time=word By default ls with the –l option displays the modification time of a file. Set word to atime (–u) to display the access time or to ctime (–t) to display the modification time. The list is sorted by word when you also give the ––sort=time option. L 748 ls –u Displays files sorted by the time they were last accessed. ––sort=access ––format=extension –X Displays files sorted by filename extension. Files with no filename extension are listed first. L –x Displays files sorted by lines (the default display is sorted by columns). ––format=across ––format=single-column –1 (one) Displays one file per line. This type of display is the default when you redirect the output from ls. Discussion The ls long listing (–l or ––format=long options) displays the columns shown in Figure 4-12 on page 94. The first column, which contains 10 or 11 characters, is divided as described in the following paragraphs. The character in the first position describes the type of file, as shown in Table V-19. Table V-19 First character in a long ls display Character Meaning – Ordinary b Block device c Character device d Directory p FIFO (named pipe) l Symbolic link The next nine characters of the first column describe the access permissions associated with the file. These characters are divided into three sets of three characters each. The first three characters represent the owner's access permissions. If the owner has read access permission to the file, r appears in the first character position. If the owner is not permitted to read the file, a hyphen appears in this position. The next two positions represent the owner's write and execute access permissions. If w appears in the second position, the owner is permitted to write to the file; if x appears in the third position, the owner is permitted to execute the file. An s in the third position indicates the file has both setuid and execute permissions. An S in the third position indicates setuid permission without execute permission. A hyphen indicates that the owner does not have the access permission associated with the character position. In a similar manner the second set of three characters represents the access permissions for the group the file is associated with. An s in the third position indicates the ls 749 file has setgid permission with execute permission, and an S indicates setgid permission with no execute permission. The third set of three characters represents the access permissions for other users. A t in the third position indicates that the file has the sticky bit (page 980) set. Refer to chmod on page 626 for information on changing access permissions. If ACLs (page 99) are enabled and a listed file has an ACL, ls –l displays a plus sign (+) following the third set of three characters. Still referring to Figure 4-12 on page 94, the second column indicates the number of hard links to the file. Refer to page 104 for more information on links. The third and fourth columns display the name of the owner of the file and the name of the group the file is associated with, respectively. The fifth column indicates the size of the file in bytes or, if information about a device file is being displayed, the major and minor device numbers. In the case of a directory, this number is the size of the directory file, not the size of the files that are entries within the directory. (Use du [page 677] to display the sum of the sizes of all files in a directory.) Use the –h option to display the size of files in kilobytes, megabytes, or gigabytes. The last two columns display the date and time the file was last modified and the filename, respectively. Notes By default, ls dereferences symbolic links: For each file that is a symbolic link, ls lists the file the link points to, not the symbolic link itself. Use the –l or –d option to reference symbolic links (display information on symbolic links, not the files the links point to). For other than long listings (displayed by the –l option), when standard output goes to the screen, ls displays output in columns based on the width of the screen. When you redirect standard output to a filter or file, ls displays a single column. Refer to page 136 for examples of using ls with ambiguous file references. With the ––color option, ls displays the filenames of various types of files in different colors. By default executable files are green, directory files are blue, symbolic links are cyan, archives and compressed files are red, and ordinary text files are black. The manner in which ls colors the various file types is specified in the /etc/DIR_COLORS file. If this file does not exist on the local system, ls will not color filenames. You can modify /etc/DIR_COLORS to alter the default color/filetype mappings on a systemwide basis. For your personal use, you can copy /etc/DIR_COLORS to the ~/.dir_colors file in your home directory and modify it. For your login, ~/.dir_colors overrides the systemwide colors established in /etc/DIR_COLORS. Refer to the dir_colors and dircolors man pages for more information. 750 ls Examples The first example shows ls, without any options or arguments, listing the names of the files in the working directory in alphabetical order. The listing is sorted in columns (vertically): $ ls bin calendar c execute letters shell The next example shows the ls utility with the –x option, which sorts the files horizontally: $ ls -x bin execute c letters calendar shell The –F option appends a slash (/) to files that are directories, an asterisk to files that are executable, and an at sign (@) to files that are symbolic links: $ ls -Fx bin/ execute* c/ letters/ calendar shell@ Next the –l (long) option displays a long list. The files are still in alphabetical order: $ ls -l total 20 drwxr-xr-x drwxr-xr-x -rw-r--r--rwxr-xr-x drwxr-xr-x lrwxrwxrwx 2 2 1 1 2 1 sam sam sam sam sam pubs 4096 May 20 pubs 4096 Mar 26 pubs 104 Jan 9 pubs 85 May 6 pubs 4096 Apr 4 sam 9 May 21 09:17 11:59 14:44 08:27 18:56 11:35 bin c calendar execute letters shell -> /bin/bash The –a (all) option lists all files, including those with hidden names: $ ls -a . .. .profile bin c calendar execute letters shell Combining the –a and –l options displays a long listing of all files, including invisible files, in the working directory. This list is still in alphabetical order: $ ls -al total 32 drwxr-xr-x drwxrwxrwx -rw-r--r--drwxr-xr-x drwxr-xr-x -rw-r--r--rwxr-xr-x drwxr-xr-x lrwxrwxrwx 5 3 1 2 2 1 1 2 1 sam sam sam sam sam sam sam sam sam sam pubs pubs pubs pubs pubs sam 4096 4096 160 4096 4096 104 85 4096 9 May May May May Mar Jan May Apr May 21 21 21 20 26 9 6 4 21 11:50 11:50 11:45 09:17 11:59 14:44 08:27 18:56 11:35 . .. .profile bin c calendar execute letters shell -> /bin/bash ls 751 When you add the –r (reverse) option to the command line, ls produces a list in reverse alphabetical order: $ ls -ral total 32 lrwxrwxrwx drwxr-xr-x -rwxr-xr-x -rw-r--r-drwxr-xr-x drwxr-xr-x -rw-r--r-- drwxrwxrwx drwxr-xr-x 1 2 1 1 2 2 1 3 5 sam sam sam sam sam sam sam sam sam sam pubs pubs pubs pubs pubs sam sam sam 9 4096 85 104 4096 4096 160 4096 4096 May Apr May Jan Mar May May May May 21 4 6 9 26 20 21 21 21 11:35 18:56 08:27 14:44 11:59 09:17 11:45 11:50 11:50 shell -> /bin/bash letters execute calendar c bin .profile .. . Use the –t and –l options to list files so the most recently modified file appears at the top of the list: $ ls -tl total 20 lrwxrwxrwx drwxr-xr-x -rwxr-xr-x drwxr-xr-x -rw-r--r-- 1 2 1 2 2 1 sam sam sam sam sam sam sam 9 May 21 11:35 pubs 4096 May 20 09:17 pubs 85 May 6 08:27 pubs 4096 Apr 4 18:56 pubs 4096 Mar 26 11:59 pubs 104 Jan 9 14:44 shell -> /bin/bash bin execute letters c calendar Together the –r and –t options cause the file you modified least recently to appear at the top of the list: $ ls -trl total 20 -rw-r--r--drwxr-xr-x drwxr-xr-x -rwxr-xr-x drwxr-xr-x lrwxrwxrwx 1 2 2 1 2 1 sam sam sam sam sam pubs 104 Jan 9 14:44 pubs 4096 Mar 26 11:59 pubs 4096 Apr 4 18:56 pubs 85 May 6 08:27 pubs 4096 May 20 09:17 sam 9 May 21 11:35 calendar c letters execute bin shell -> /bin/bash The next example shows ls with a directory filename as an argument. The ls utility lists the contents of the directory in alphabetical order: $ ls bin c e lsdir To display information about the directory file itself, use the –d (directory) option. This option lists information about the directory only: $ ls -dl bin drwxr-xr-x 2 sam pubs 4096 May 20 09:17 bin You can use the following command to display a list of all hidden filenames (those starting with a period) in your home directory. It is a convenient way to list the startup (initialization) files in your home directory. 752 ls $ ls -d ~/.* /home/sam/. /home/sam/.. /home/sam/.AbiSuite /home/sam/.Azureus /home/sam/.BitTornado ... A plus sign (+) to the right of the permissions in a long listing denotes the presence of an ACL for a file: $ ls -l memo -rw-r--r--+ 1 sam pubs 19 Jul 19 21:59 memo Under Mac OS X you can use the –le option to display an ACL: $ ls -le memo -rw-r--r-- + 1 sam pubs 19 Jul 19 21:59 memo 0: user:jenny allow read See page 934 for more examples of using ls under Mac OS X to display ACLs. make 753 Keeps a set of programs current make [options] [target-files] [arguments] The GNU make utility keeps a set of executable programs current, based on differences in the modification times of the programs and the source files that each program is dependent on. Arguments The target-files refer to targets on dependency lines in the makefile. When you do not specify a target-file, make updates the target on the first dependency line in the makefile. Command-line arguments of the form name=value set the variable name to value inside the makefile. See the "Discussion" section for more information. Options If you do not use the –f option, make takes its input from a file named GNUmakefile, makefile, or Makefile (in that order) in the working directory. In this section, this input file is referred to as makefile. Many users prefer to use the name Makefile because it shows up earlier in directory listings. The Mac OS X version of make accepts long options tip Options for make preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––directory=dir –C dir Changes directories to dir before starting. ––debug –d Displays information about how make decides what to do. ––file=file –f file (input file) Uses file as input instead of makefile. ––jobs[=n] –j [n] (jobs) Runs up to n commands at the same time instead of the default of one command. Running multiple commands simultaneously is especially effective if you are working on a multiprocessor system. If you omit n, make does not limit the number of simultaneous jobs. ––keep-going –k Continues with the next file from the list of target-files instead of quitting when a construction command fails. ––just-print or ––dry-run –n (no execution) Displays, but does not execute, the commands that make would execute to bring the target-files up-to-date. ––silent or ––quiet ––touch –s Does not display the names of the commands being executed. –t Updates the modification times of target files but does not execute any construction commands. Refer to touch on page 862. make make 754 make Discussion The make utility bases its actions on the modification times of the programs and the source files that each program depends on. Each of the executable programs, or target-files, depends on one or more prerequisite files. The relationships between target-files and prerequisites are specified on dependency lines in a makefile. Construction commands follow the dependency line, specifying how make can update the target-files. See page 756 for examples of makefiles. Documentation Refer to www.gnu.org/software/make/manual/make.html and to the make info page for more information about make and makefiles. Although the most common use of make is to build programs from source code, this general-purpose build utility is suitable for a wide range of applications. Anywhere you can define a set of dependencies to get from one state to another represents a candidate for using make. Much of make's power derives from the features you can set up in a makefile. For example, you can define variables using the same syntax found in the Bourne Again Shell. Always define the variable SHELL in a makefile; set it to the pathname of the shell you want to use when running construction commands. To define the variable and assign it a value, place the following line near the top of a makefile: SHELL=/bin/sh Assigning the value /bin/sh to SHELL allows you to use a makefile on other computer systems. On Linux systems, /bin/sh is generally linked to /bin/bash or /bin/dash. Under Mac OS X, /bin/sh is a copy of bash that attempts to emulate the original Bourne Shell. The make utility uses the value of the environment variable SHELL if you do not set SHELL in a makefile. If SHELL does not hold the path of the shell you intended to use and if you do not set SHELL in a makefile, the construction commands may fail. Following is a list of additional features associated with make: • You can run specific construction commands silently by preceding them with an at sign (@). For example, the following lines will display a short help message when you run the command make help: help: @echo @echo @echo @echo @echo "You can make the following:" " " "libbuf.a -- the buffer library" "Bufdisplay -- display any-format buffer" "Buf2ppm -- convert buffer to pixmap" Without the @ signs in the preceding example, make would display each of the echo commands before executing it. This way of displaying a message works because no file is named help in the working directory. As a result make runs the construction commands in an attempt to build this file. Because the construction commands display messages but do not build the file help, you can run make help repeatedly with the same result. make 755 • You can cause make to ignore the exit status of a command by preceding the command with a hyphen (–). For example, the following line allows make to continue regardless of whether the call to /bin/rm is successful (the call to /bin/rm fails if libbuf.a does not exist): -/bin/rm libbuf.a • You can use special variables to refer to information that might change from one use of make to the next. Such information might include files that need updating, files that are newer than the target, and files that match a pattern. For example, you can use the variable $? in a construction command to identify all prerequisite files that are newer than the target file. This variable allows you to print any files that have changed since the last time you printed those files out: list: .list .list: Makefile buf.h xtbuff_ad.h buff.c buf_print.c xtbuff.c pr $? | lpr date >.list The target list depends on the source files that might be printed. The construction command pr $? | lpr prints only those source files that are newer than the file .list. The line date > .list modifies the .list file so it is newer than any of the source files. The next time you run the command make list, only the files that have been changed are printed. • You can include other makefiles as if they were part of the current makefile. The following line causes make to read Make.config and treat the contents of that file as though it were part of the current makefile, allowing you to put information common to more than one makefile in a single place: include Make.config Note Under Mac OS X, the make utility is part of the Developer Tools optional install. Examples The first example causes make to bring the target-file named analysis up-to-date by issuing three cc commands. It uses a makefile named GNUmakefile, makefile, or Makefile in the working directory. $ make analysis cc -c analy.c cc -c stats.c cc -o analysis analy.o stats.o The following example also updates analysis but uses a makefile named analysis.mk in the working directory: $ make -f analysis.mk analysis 'analysis' is up to date. 756 make The next example lists the commands make would execute to bring the target-file named credit up-to-date. Because of the –n (no-execution) option, make does not execute the commands. $ make -n credit cc -c -O credit.c cc -c -O accounts.c cc -c -O terms.c cc -o credit credit.c accounts.c terms.c The next example uses the –t option to update the modification time of the target-file named credit. After you use this option, make thinks that credit is up-to-date. $ make -t credit $ make credit 'credit' is up to date. Example makefiles Following is a very simple makefile named Makefile. This makefile compiles a program named morning (the target file). The first line is a dependency line that shows morning depends on morning.c. The next line is the construction line: It shows how to create morning using the gcc C compiler. The construction line must be indented using a TAB, not SPACEs. $ cat Makefile morning: morning.c TAB gcc -o morning morning.c When you give the command make, make compiles morning.c if it has been modified more recently than morning. The next example is a simple makefile for building a utility named ff. Because the cc command needed to build ff is complex, using a makefile allows you to rebuild ff easily, without having to remember and retype the cc command. $ cat Makefile # Build the ff command from the fastfind.c source SHELL=/bin/sh ff: gcc -traditional -O2 -g -DBIG=5120 -o ff fastfind.c myClib.a $ make ff gcc -traditional -O2 -g -DBIG=5120 -o ff fastfind.c myClib.a In the next example, a makefile keeps the file named compute up-to-date. The make utility ignores comment lines (lines that begin with a pound sign [#]); the first three lines of the following makefile are comment lines. The first dependency line shows that compute depends on two object files: compute.o and calc.o. The corresponding construction line gives the command make needs to produce compute. The second dependency line shows that compute.o depends not only on its C source file but also on the compute.h header file. The construction line for compute.o uses the C compiler optimizer (–O3 option). The third set of dependency and construction lines is not required. In their absence, make infers that calc.o depends on calc.c and produces the command line needed for the compilation. make 757 $ cat Makefile # # Makefile for compute # compute: compute.o calc.o gcc -o compute compute.o calc.o compute.o: compute.c compute.h gcc -c -O3 compute.c calc.o: calc.c gcc -c calc.c clean: rm *.o *core* *~ There are no prerequisites for clean, the last target. This target is often used to remove extraneous files that may be out-of-date or no longer needed, such as .o files. The next example shows a much more sophisticated makefile that uses features not discussed in this section. Refer to the sources cited under "Documentation" on page 754 for information about these and other advanced features. $ cat Makefile ## ## build and maintain the buffer library ## SHELL=/bin/sh ## ## Flags and libraries for compiling. The XLDLIBS are needed # whenever you build a program using the library. The CCFLAGS # give maximum optimization. CC=gcc CCFLAGS=-O2 $(CFLAGS) XLDLIBS= -lXaw3d -lXt -lXmu -lXext -lX11 -lm BUFLIB=libbuf.a ## ## Miscellaneous INCLUDES=buf.h XINCLUDES=xtbuff_ad.h OBJS=buff.o buf_print.o xtbuff.o ## ## Just a 'make' generates a help message help: Help @echo "You can make the following:" @echo " " @echo " libbuf.a -- the buffer library" @echo " bufdisplay -- display any-format buffer" @echo " buf2ppm -- convert buffer to pixmap" ## 758 make ## The main target is the library libbuf.a: $(OBJS) -/bin/rm libbuf.a ar rv libbuf.a $(OBJS) ranlib libbuf.a ## ## Secondary targets -- utilities built from the library bufdisplay: bufdisplay.c libbuf.a $(CC) $(CCFLAGS) bufdisplay.c -o bufdisplay $(BUFLIB) $(XLDLIBS) buf2ppm: buf2ppm.c libbuf.a $(CC) $(CCFLAGS) buf2ppm.c -o buf2ppm $(BUFLIB) ## ## Build the individual object units buff.o: $(INCLUDES) buff.c $(CC) -c $(CCFLAGS) buff.c buf_print.o:$(INCLUDES) buf_print.c $(CC) -c $(CCFLAGS) buf_print.c xtbuff.o: $(INCLUDES) $(XINCLUDES) xtbuff.c $(CC) -c $(CCFLAGS) xtbuff.c The make utility can be used for tasks other than compiling code. As a final example, assume you have a database that lists IP addresses and the corresponding hostnames in two columns and that the database dumps these values to a file named hosts.tab. You need to extract only the hostnames from this file and generate a Web page named hosts.html containing these names. The following makefile is a simple report writer: $ cat makefile # SHELL=/bin/bash # hosts.html: hosts.tab @echo "" > hosts.html @awk '{print $$2, ""}' hosts.tab >> hosts.html @echo "" >> hosts.html man 759 man man [options] [section] command man –k keyword The man (manual) utility provides online documentation for Linux and Mac OS X commands. In addition to user commands, documentation is available for many other commands and details that relate to Linux and OS X. Because many Linux and OS X commands come from GNU, the GNU info utility (page 36) frequently provides more complete information about them. A one-line header is associated with each manual page. This header consists of a command name, the section of the manual in which the command is found, and a brief description of what the command does. These headers are stored in a database; thus you can perform quick searches on keywords associated with each man page. Arguments The section argument tells man to limit its search to the specified section of the manual (see page 34 for a listing of manual sections). Without this argument man searches the sections in numerical order and displays the first man page it finds. In the second form of the man command, the –k option searches for the keyword in the database of man page headers; man displays a list of headers that contain the keyword. A man –k command performs the same function as apropos (page 35). Options Options preceded by a double hyphen (––) work under Linux only. Not all options preceded by a double hyphen work under all Linux distributions. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––all –a Displays man pages for all sections of the manual. Without this option man displays only the first page it finds. Use this option when you are not sure which section contains the desired information. –K keyword Searches for keyword in all man pages. This option can take a long time to run. It is not available under some Linux distributions. ––apropos –k keyword Displays manual page headers that contain the string keyword. You can scan this list for commands of interest. This option is equivalent to the apropos command (page 35). ––manpath=path –M path Searches the directories in path for man pages, where path is a colon-separated list of directories. See "Discussion." man Displays documentation for commands 760 man ––troff –t Formats the page for printing on a PostScript printer. The output goes to standard output. Discussion The manual pages are organized in sections, each pertaining to a separate aspect of the Linux system. Section 1 contains user-callable commands and is the section most likely to be accessed by users who are not system administrators or programmers. Other sections of the manual describe system calls, library functions, and commands used by system administrators. See page 34 for a listing of the manual sections. Pager The man utility uses less to display manual pages that fill more than one screen. To use another pager, set the environment variable PAGER to the pathname of that pager. For example, adding the following line to the ~/.bash_profile file allows a bash user to use more instead of less: export PAGER=/usr/bin/less Under OS X, less and more are copies of the same file. Because of the way each is called, they work slightly differently. MANPATH You can tell man where to look for man pages by setting the environment variable MANPATH to a colon-separated list of directories. For example, bash users running Linux can add the following line to ~/.bash_profile to cause man to search the /usr/man, /usr/local/man, and /usr/local/share/man directories: export MANPATH=/usr/man:/usr/local/man:/usr/local/share/man Working as a privileged user, you can edit /etc/manpath.config or /etc/man.config (Linux) or /etc/man.conf (OS X) to further configure man. Refer to the man man page for more information. Notes See page 33 for another discussion of man. The argument to man does not have to be a command name. For example, the command man ascii lists the ASCII characters and their various representations; the command man –k postscript lists man pages that pertain to PostScript. The man pages are commonly stored in an unformatted, compressed form. When you request a man page, it has to be decompressed and formatted before being displayed. To speed up subsequent requests for that man page, man attempts to save the formatted version of the page. Some utilities described in the manual pages have the same name as shell builtin commands. The behavior of the shell builtin may differ slightly from the behavior of the utility as described in the manual page. For information about shell builtins, see the man page for builtin or the man page for a specific shell. References to man pages frequently use section numbers in parentheses. For example, write(2) refers to the man page for write in section 2 of the manual (page 34). man 761 The first of the following commands uses the col utility to generate a simple text man page that does not include bold or underlined text. The second command generates a PostScript version of the man page. $ man ls | col -b > ls.txt $ man -t ls > ls.ps Under Linux you can use ps2pdf to convert the PostScript file to a PDF file. Examples The following example uses man to display the documentation for the command write, which sends messages to another user's terminal: $ man write WRITE(1) BSD General Commands Manual WRITE(1) NAME write - send a message to another user SYNOPSIS write user [ttyname] DESCRIPTION The write utility allows you to communicate with other users, by copying lines from your terminal to theirs. When you run the write command, the user you are writing to gets a message of the form: Message from yourname@yourhost on yourtty at hh:mm ... ... The next example displays the man page for another command—the man command itself, which is a good starting place for someone learning about the system: $ man man MAN(1) Manual pager utils MAN(1) NAME man - an interface to the on-line reference manuals SYNOPSIS man [-c|-w|-tZ] [-H[browser]] [-T[device]] [-X[dpi]] [-adhu7V] [-i|-I] [-m system[,...]] [-L locale] [-p string] [-C file] [-M path] [-P pager] [-r prompt] [-S list] [-e extension] [--warnings [warnings]] [[section] page ...] ... ... DESCRIPTION man is the system's manual pager. Each page argument given to man is normally the name of a program, utility or function. The manual page associated with each of these arguments is then found and displayed. A ... 762 man You can also use the man utility to find the man pages that pertain to a certain topic. In the next example, man –k displays man page headers containing the string latex. The apropos utility functions similarly to man –k. $ man -k latex elatex (1) [latex] latex (1) mkindex (1) pdflatex (1) pod2latex (1) Pod::LaTeX (3perl) ... - structured text formatting and typesetting structured text formatting and typesetting script to process LaTeX index and glossary files PDF output from TeX convert pod documentation to latex format Convert Pod data to formatted Latex The search for the keyword entered with the –k option is not case sensitive. Although the keyword entered on the command line is all lowercase, it matches the last header, which contains the string LaTeX (uppercase and lowercase). The 3perl entry on the last line indicates the man page is from Section 3 (Subroutines) of the Linux System Manual and comes from the Perl Programmers Reference Guide (it is a Perl subroutine; see Chapter 11 for more information on the Perl programming language). mkdir 763 Creates a directory mkdir [option] directory-list The mkdir utility creates one or more directories. Arguments The directory-list is a list of pathnames of directories that mkdir creates. Options Under Linux, mkdir accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––mode=mode –m mode Sets the permission to mode. You can represent the mode absolutely using an octal number (Table V-7 on page 627) or symbolically (Table V-4 on page 626). ––parents –p Creates directories that do not exist in the path to the directory you wish to create. ––verbose –v Displays the name of each directory created. This option is helpful when used with the –p option. Notes You must have permission to write to and search (execute permission) the parent directory of the directory you are creating. The mkdir utility creates directories that contain the standard hidden entries (. and ..). Examples The following command creates the accounts directory as a subdirectory of the working directory and the prospective directory as a subdirectory of accounts: $ mkdir -p accounts/prospective Without changing working directories, the same user now creates another subdirectory within the accounts directory: $ mkdir accounts/existing Next the user changes the working directory to the accounts directory and creates one more subdirectory: $ cd accounts $ mkdir closed The last example shows the user creating another subdirectory. This time the ––mode option removes all access permissions for the group and others: $ mkdir -m go= accounts/past_due mkdir mkdir 764 mkfs mkfs mkfs Creates a filesystem on a device mkfs [options] device The mkfs utility creates a filesystem on a device such as a floppy diskette or a partition of a hard disk. It acts as a front end for programs that create filesystems, each specific to a filesystem type. The mkfs utility is available under Linux only. L mkfs destroys all data on a device caution Be careful when using mkfs: It destroys all data on a device. Arguments The device is the name of the device that you want to create the filesystem on. If the device name is in /etc/fstab, you can use the mount point of the device instead of the device name (e.g., /home in place of /dev/sda2). Options When you run mkfs, you can specify both global options and options specific to the filesystem type that mkfs is creating (e.g., ext2, ext3, ext4, msdos, reiserfs). Global options must precede type-specific options. Global Options –t fstype (type) The fstype is the type of filesystem you want to create—for example, ext2, ext3, msdos, or reiserfs. The default filesystem varies. –v (verbose) Displays more output. Use –V for filesystem-specific information. Filesystem Type-Specific Options The options described in this section apply to many common filesystem types, including ext2 and ext3. The following command lists the filesystem creation utilities available on the local system: $ ls /sbin/mkfs.* /sbin/mkfs.bfs /sbin/mkfs.cramfs /sbin/mkfs.ext2 /sbin/mkfs.ext3 /sbin/mkfs.minix /sbin/mkfs.msdos /sbin/mkfs.reiserfs /sbin/mkfs.vfat There is frequently a link to /sbin/mkfs.ext2 at /sbin/mke2fs. Review the man page or give the pathname of the filesystem creation utility to determine which options the utility accepts. $ /sbin/mkfs.ext3 Usage: mkfs.ext3 [-c|-l filename] [-b block-size] [-f fragment-size] [-i bytes-per-inode] [-I inode-size] [-J journal-options] [-N number-of-inodes] [-m reserved-blocks-percentage] [-o creator-os] [-g blocks-per-group] [-L volume-label] [-M last-mounted-directory] [-O feature[,...]] [-r fs-revision] [-E extended-option[,...]] [-T fs-type] [-jnqvFSV] device [blocks-count] mkfs 765 –b size (block) Specifies the size of blocks in bytes. On ext2, ext3, and ext4 filesystems, valid block sizes are 1,024, 2,048, and 4,096 bytes. –c (check) Checks for bad blocks on the device before creating a filesystem. Specify this option twice to perform a slow, destructive, read-write test. Discussion Before you can write to and read from a hard disk or floppy diskette in the usual fashion, there must be a filesystem on it. Typically a hard disk is divided into partitions (page 970), each with a separate filesystem. A floppy diskette normally holds a single filesystem. Refer to Chapter 4 for more information on filesystems. Notes Under Mac OS X, use diskutil (page 668) to create a filesystem. You can use tune2fs (page 868) with the –j option to change an existing ext2 filesystem into a journaling filesystem (page 961) of type ext3. (See the "Examples" section.) You can also use tune2fs to change how often fsck (page 699) checks a filesystem. mkfs is a front end Examples Much like fsck, mkfs is a front end that calls other utilities to handle various types of filesystems. For example, mkfs calls mke2fs (which is typically linked to mkfs.ext2 and mkfs.ext3) to create the widely used ext2 and ext3 filesystems. Refer to the mke2fs man page for more information. Other utilities that mkfs calls are typically named mkfs.type, where type is the filesystem type. By splitting mkfs in this manner, filesystem developers can provide programs to create their filesystems without affecting the development of other filesystems or changing how system administrators use mkfs. In the following example, mkfs creates a filesystem on the device at /dev/hda8. In this case the default filesystem type is ext2. # /sbin/mkfs /dev/sda5 mke2fs 1.41.4 (27-Jan-2009) Filesystem label= OS type: Linux Block size=1024 (log=0) Fragment size=1024 (log=0) 122880 inodes, 489948 blocks 24497 blocks (5.00%) reserved for the super user First data block=1 Maximum filesystem blocks=67633152 60 block groups 8192 blocks per group, 8192 fragments per group 2048 inodes per group Superblock backups stored on blocks: 8193, 24577, 40961, 57345, 73729, 204801, 221185, 401409 Writing inode tables: done Writing superblocks and filesystem accounting information: done This filesystem will be automatically checked every 37 mounts or 180 days, whichever comes first. Use tune2fs -c or -i to override. 766 mkfs Next the administrator uses tune2fs to convert the ext2 filesystem to an ext3 journaling filesystem: # /sbin/tune2fs -j /dev/sda5 tune2fs 1.41.4 (27-Jan-2009) Creating journal inode: done This filesystem will be automatically checked every 37 mounts or 180 days, whichever comes first. Use tune2fs -c or -i to override. Mtools 767 Uses DOS-style commands on files and directories mcd [directory] mcopy [options] file-list target mdel file-list mdir [–w] directory mformat [options] device mtype [options] file-list These utilities mimic DOS commands and manipulate Linux, Mac OS X, or DOS files. The mcopy utility provides an easy way to move files between a Linux/OS X filesystem and a DOS disk. The default drive for all commands is /dev/fd0 or A:. Utilities Table V-20 lists some of the utilities in the Mtools collection. Table V-20 The Mtools collection Utility Function mcd Changes the working directory on the DOS disk mcopy Copies DOS files from one directory to another mdel Deletes DOS files mdir Lists the contents of DOS directories mformat Adds DOS formatting information to a disk mtype Displays the contents of DOS files Arguments The directory, which is used with mcd and mdir, must be the name of a directory on a DOS disk. The file-list, which is used with mcopy and mtype, is a SPACE-separated list of filenames. The target, which is used with mcopy, is the name of a regular file or a directory. If you give mcopy a file-list with more than one filename, target must be the name of a directory. The device, which is used

with mformat, is the DOS drive letter containing the disk to be formatted (for example, A:). Options mcopy –n Automatically replaces existing files without asking. Normally mcopy asks for verification before overwriting a file. –p (preserve) Preserves the attributes of files when they are copied. –s (recursive) Copies directory hierarchies. Mtools Mtools 768 Mtools –t (text) Converts DOS text files for use on a Linux/OS X system, and vice versa. Lines in DOS text files are terminated with the character pair RETURN–NEWLINE; lines in Linux/OS X text files end in NEWLINE. This option removes the RETURN character when copying from a DOS file and adds it when copying from a Linux file. mdir –w (wide) Displays only filenames and fits as many as possible on each line. By default mdir lists information about each file on a separate line, showing the filename, size, and creation time. mformat –f 1440 Specifies a 1,440K 3.5-inch HD floppy diskette. –f 2880 Specifies a 2,880K 3.5-inch ED floppy diskette. –v vol (label) Puts vol as the volume label on the newly formatted DOS disk. mtype –t (text) Similar to the –t option for mcopy, this option replaces each RETURN–NEWLINE character pair in the DOS file with a single NEWLINE character before displaying the file. Discussion Although these utilities mimic their DOS counterparts, they do not attempt to match those tools exactly. In most cases the restrictions imposed by DOS are removed. For example, the ambiguous file reference represented by the asterisk (*) matches all filenames (as it does under Linux/OS X), including those filenames that DOS would require *.* to match. Notes In this discussion, the term DOS disk refers to either a DOS partition on a hard disk or a DOS floppy diskette. If Mtools is not available in a Linux distribution repository, you can download Mtools from the Mtools home page (www.gnu.org/software/mtools). Under Mac OS X, follow these steps to download and compile Mtools: 1. Install MacPorts. 2. Exit from any running terminals. 3. Start a terminal and run sudo port install Mtools. Mtools 769 If the local kernel is configured to support DOS filesystems, you can mount DOS disks on a Linux/OS X filesystem and manipulate the files using Linux/OS X utilities. Although this feature is handy and reduces the need for Mtools, it may not be practical or efficient to mount and unmount DOS filesystems each time you need to access a DOS file. These tasks can be time-consuming, and some systems are set up so regular users cannot mount and unmount filesystems. Use caution when using Mtools. Its utilities may not warn you if you are about to overwrite a file. Using explicit pathnames—not ambiguous file references—reduces the chance of overwriting a file. The Mtools utilities are most commonly used to examine files on DOS floppy diskettes (mdir) and to copy files between a DOS floppy diskette and the Linux filesystem (mcopy). You can identify DOS disks by using the usual DOS drive letters: A: for the first floppy drive, C: for the first hard disk, and so on. To separate filenames in paths use either the Linux forward slash (/) or the DOS backslash (\). You need to escape backslashes to prevent the shell from interpreting them before passing the pathname to the utility you are using. Each of the Mtools utilities returns an exit code of 0 on success, 1 on complete failure, and 2 on partial failure. Examples In the first example, mdir displays the contents of a DOS floppy diskette in /dev/fd0: $ mdir Volume in drive A is DOS UTY Directory for A:/ ACAD CADVANCE CHIPTST DISK GENERIC INSTALL INSTALL KDINSTAL LOTUS PCAD READID README UTILITY WORD WP 15 LIF LIF EXE ID LIF COM DAT EXE LIF LIF EXE TXT LIF LIF LIF File(s) 419370 5-10-09 1:29p 40560 2-08-08 10:36a 2209 4-26-09 4:22p 31 12-27-09 4:49p 20983 2-08-08 10:37a 896 7-05-09 10:23a 45277 12-27-09 4:49p 110529 8-13-09 10:50a 44099 1-18-09 3:36p 17846 5-01-09 3:46p 17261 5-07-09 8:26a 9851 4-30-09 10:32a 51069 5-05-09 9:13a 16817 7-01-09 9:58a 57992 8-29-09 4:22p 599040 bytes free The next example uses mcopy to copy the *.TXT files from the DOS floppy diskette to the working directory on the local filesystem. Because only one file has the extension .TXT, only one file is copied. Because .TXT files are usually text files under 770 Mtools DOS, the –t option strips the unnecessary RETURN characters at the end of each line. The ambiguous file reference * is escaped on the command line to prevent the shell from attempting to expand it before passing the argument to mcopy. The mcopy utility locates the file README.TXT when given the pattern *.txt because DOS does not differentiate between uppercase and lowercase letters in filenames. $ mcopy -t a:*.txt . Copying README.TXT Finally, the DOS floppy diskette is reformatted using mformat, which wipes all data from the diskette. If the diskette has not been low-level formatted, you need to use fdformat before giving the following command: $ mformat a: A check with mdir shows the floppy diskette is empty after formatting: $ mdir a: Can't open /dev/fd0: No such device or address Cannot initialize 'A:' mv 771 mv mv [options] existing-file new-filename mv [options] existing-file-list directory mv [options] existing-directory new-directory The mv utility, which renames or moves one or more files, has three formats. The first renames a single file with a new filename that you supply. The second renames one or more files so that they appear in a specified directory. The third renames a directory. The mv utility physically moves the file if it is not possible to rename it (that is, if you move the file from one filesystem to another). Arguments In the first form, the existing-file is a pathname that specifies the ordinary file you want to rename. The new-filename is the new pathname of the file. In the second form, the existing-file-list is a list of the pathnames of the files you want to rename and the directory specifies the new parent directory for the files. The files you rename will have the same simple filenames as each of the files in the existing-file-list but new absolute pathnames. The third form renames the existing-directory with the new-directory name. This form works only when the new-directory does not already exist. Options Under Linux, mv accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––backup –b Makes a backup copy (by appending ~ to the filename) of any file that would be overwritten. L ––force –f Causes mv not to prompt you if a move would overwrite an existing file that you do not have write permission for. You must have write permission for the directory holding the existing file. ––interactive –i Prompts for confirmation if mv would overwrite a file. If your response begins with a y or Y, mv overwrites the file; otherwise, mv does not move the file. ––update –u If a move would overwrite an existing file—not a directory—this option causes mv to compare the modification times of the source and target files. If the target file has a more recent modification time (the target is newer than the source), mv does not replace it. L ––verbose –v Lists files as they are moved. mv Renames or moves a file 772 mv Notes When GNU mv copies a file from one filesystem to another, mv is implemented as cp (with the –a option) and rm: It first copies the existing-file to the new-file and then deletes the existing-file. If the new-file already exists, mv may delete it before copying. As with rm, you must have write and execute access permissions to the parent directory of the existing-file, but you do not need read or write access permission to the file itself. If the move would overwrite a file that you do not have write permission for, mv displays the file's access permissions and waits for a response. If you enter y or Y, mv overwrites the file; otherwise, it does not move the file. If you use the –f option, mv does not prompt you for a response but simply overwrites the file. Although earlier versions of mv could move only ordinary files between filesystems, mv can now move any type of file, including directories and device files. Examples The first command renames letter, a file in the working directory, as letter.1201: $ mv letter letter.1201 The next command renames the file so it appears with the same simple filename in the user's ~/archives directory: $ mv letter.1201 ~/archives The following command moves all files in the working directory whose names begin with memo so they appear in the /p04/backup directory: $ mv memo* /p04/backup Using the –u option prevents mv from replacing a newer file with an older one. After the mv –u command shown below is executed, the newer file (memo2) has not been overwritten. The mv command without the –u option overwrites the newer file (memo2's modification time and size have changed to those of memo1). $ ls -l -rw-rw-r-- 1 sam sam -rw-rw-r-- 1 sam sam $ mv -u memo1 memo2 $ ls -l -rw-rw-r-- 1 sam sam -rw-rw-r-- 1 sam sam $ mv memo1 memo2 $ ls -l -rw-rw-r-- 1 sam sam 22 Mar 25 23:34 memo1 19 Mar 25 23:40 memo2 22 Mar 25 23:34 memo1 19 Mar 25 23:40 memo2 22 Mar 25 23:34 memo2 nice 773 nice nice [option] [command-line] The nice utility reports the priority of the shell or alters the priority of a command. An ordinary user can decrease the priority of a command. Only a user working with root privileges can increase the priority of a command. The nice builtin in the TC Shell has a different syntax. Refer to the "Notes" section for more information. Arguments The command-line is the command line you want to execute at a different priority. Without any options or arguments, nice displays the priority of the shell running nice. Options Without an option, nice defaults to an adjustment of 10, lowering the priority of the command by 10—typically from 0 to 10. As you raise the priority value, the command runs at a lower priority. The option preceded by a double hyphen (––) works under Linux only. The option named with a single letter and preceded by a single hyphen works under Linux and OS X. ––adjustment=value –n value Changes the priority by the increment (or decrement) specified by value. The priorities range from –20 (the highest priority) to 19 (the lowest priority). A positive value lowers the priority, whereas a negative value raises the priority. Only a user working with root privileges can specify a negative value. When you specify a value outside this range, the priority is set to the limit of the range. Notes You can use renice (page 802) or top's r command (page 860) to change the priority of a running process. Higher (more positive) priority values mean that the kernel schedules a job less often. Lower (more negative) values cause the job to be scheduled more often. When a user working with root privileges schedules a job to run at the highest priority, this change can affect the performance of the system for all other jobs, including the operating system itself. For this reason you should be careful when using nice with negative values. The TC Shell includes a nice builtin. Under tcsh, use the following syntax to change the priority at which command-line is run. The default priority is 4. You must include the plus sign for positive values. nice [±value] command line nice Changes the priority of a command 774 nice The tcsh nice builtin works differently from the nice utility: When you use the builtin, nice –5 decrements the priority at which command-line is run. When you use the utility, nice –n –5 increments the priority at which command-line is run. Examples The following command executes find in the background at the lowest possible priority. The ps –l command displays the nice value of the command in the NI column: # nice -n 19 find / -name core -print > corefiles.out & [1] 422 # ps -l F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY 4 R 0 389 8657 0 80 0 - 4408 pts/4 4 D 0 422 389 28 99 19 - 1009 pts/4 0 R 0 433 389 0 80 0 - 1591 pts/4 TIME 00:00:00 00:00:04 00:00:00 The next command finds very large files and runs at a high priority (–15): # nice -n -15 find / -size +50000k CMD bash find ps nohup 775 Runs a command that keeps running after you log out nohup command line The nohup utility executes a command line such that the command keeps running after you log out. In other words, nohup causes a process to ignore a SIGHUP signal. Depending on how the local shell is configured, it may kill your background processes when you log out. The TC Shell includes a nohup builtin. Refer to the "Notes" section for more information. Arguments The command line is the command line you want to execute. Notes Under Linux, nohup accepts the common options described on page 603. If you do not redirect the output from a command you execute using nohup, both standard output and standard error are sent to the file named nohup.out in the working directory. If you do not have write permission for the working directory, nohup sends output to ~/nohup.out. Unlike the nohup utility, the TC Shell's nohup builtin does not send output to nohup.out. Background jobs started from tcsh continue to run after you log out. Examples The following command executes find in the background, using nohup: $ nohup find / -name core -print > corefiles.out & [1] 14235 nohup nohup 776 od od Dumps the contents of a file od od [options] [file-list] The od (octal dump) utility dumps the contents of a file. The dump is useful for viewing executable (object) files and text files with embedded nonprinting characters. This utility takes its input from the file you specify on the command line or from standard input. The od utility is available under Linux only. L Arguments The file-list specifies the pathnames of the files that od displays. When you do not specify a file-list, od reads from standard input. Options The od utility accepts the common options described on page 603. ––address-radix=base –A base Specifies the base used when displaying the offsets shown for positions in the file. By default offsets are given in octal. Possible values for base are d (decimal), o (octal), x (hexadecimal), and n (no offsets printed). –j n Skips n bytes before displaying data. ––skip-bytes=n ––read-bytes=n –N n Reads a maximum of n bytes and then quits. –S n Outputs from the file only those bytes that contain runs of n or more printable ASCII characters that are terminated by a NULL byte. ––strings=n ––format=type[n] –t type[n] Specifies the output format for displaying data from a file. You can repeat this option with different format types to see the file in several different formats. Table V-21 lists the possible values for type. By default od dumps a file as 2-byte octal numbers. You can specify the number of bytes od uses to compose each number by specifying a length indicator, n. You can specify a length indicator for all types except a and c. Table V-23 lists the possible values of n. Table V-21 Output formats type Type of output a Named character. Displays nonprinting control characters using their official ASCII names. For example, FORMFEED is displayed as ff. od 777 Table V-21 Output formats (continued) type Type of output c ASCII character. Displays nonprinting control characters as backslash escape sequences (Table V-22) or three-digit octal numbers. d Signed decimal. f Floating point. o Octal (default). u Unsigned decimal. x Hexadecimal. Table V-22 Output format type c backslash escape sequences Sequence Meaning \0 NULL \a BELL \b BACKSPACE \f FORMFEED NEWLINE \r RETURN \t HORIZONTAL TAB \v VERTICAL TAB Table V-23 Length indicators n Number of bytes to use Integers (types d, o, u, and x) C (character) Uses single characters for each decimal value S (short integer) Uses 2 bytes I (integer) Uses 4 bytes L (long) Uses 4 bytes on 32-bit machines and 8 bytes on 64-bit machines 778 od Table V-23 Length indicators (continued) Floating point (type f) Notes F (float) Uses 4 bytes D (double) Uses 8 bytes L (long double) Typically uses 8 bytes To retain backward compatibility with older, non-POSIX versions of od, the od utility includes the options listed in Table V-24 as shorthand versions of many of the preceding options. Table V-24 Examples Shorthand format specifications Shorthand Equivalent specification –a –t a –b –t oC –c –t c –d –t u2 –f –t fF –h –t x2 –i –t d2 –l –t d4 –o –t o2 –x –t x2 The file ac, which is used in the following examples, contains all of the ASCII characters. In the first example, the bytes in this file are displayed as named characters. The first column shows the offset of the first byte on each line of output from the start of the file. The offsets are displayed as octal values. $ od -t 0000000 0000020 0000040 0000060 0000100 0000120 0000140 0000160 0000200 0000220 0000240 a ac nul soh stx etx eot enq ack bel bs ht nl vt ff dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc sp ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~ del nul soh stx etx eot enq ack bel bs ht nl vt ff dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc sp ! " # $ % & ' ( ) * + , - . / cr so si fs gs rs us cr so si fs gs rs us od 779 0000260 0000300 0000320 0000340 0000360 0000400 0000401 0 1 @ A P Q ` a p q nl 2 B R b r 3 C S c s 4 D T d t 5 E U e u 6 F V f v 7 G W g w 8 H X h x 9 I Y i y : J Z j z ; K [k { < L \ l | = M] m } > N ^ n ~ ? O _ o del In the next example, the bytes are displayed as octal numbers, ASCII characters, or printing characters preceded by a backslash (refer to Table V-22 on page 777): $ od -t 0000000 0000020 0000040 0000060 0000100 0000120 0000140 0000160 0000200 0000220 0000240 0000260 0000300 0000320 0000340 0000360 0000400 0000401 c ac \0 001 002 003 004 005 006 \a \b \t 020 021 022 023 024 025 026 027 030 ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~ 177 200 201 202 203 204 205 206 207 210 220 221 222 223 224 225 226 227 230 240 241 242 243 244 245 246 247 250 260 261 262 263 264 265 266 267 270 300 301 302 303 304 305 306 307 310 320 321 322 323 324 325 326 327 330 340 341 342 343 344 345 346 347 350 360 361 362 363 364 365 366 367 370 \v \f \r 016 017 031 032 033 034 035 036 037 211 231 251 271 311 331 351 371 212 232 252 272 312 332 352 372 213 233 253 273 313 333 353 373 214 234 254 274 314 334 354 374 215 235 255 275 315 335 355 375 216 236 256 276 316 336 356 376 217 237 257 277 317 337 357 377 The final example finds in the file /usr/bin/who all strings that are at least three characters long (the default) and are terminated by a null byte. See strings on page 837 for another way of displaying a similar list. The offset positions are given as decimal offsets instead of octal offsets. $ od -A ... 0028455 0028473 0028487 0028503 0028520 0028525 0028539 0028543 0028558 0028573 0028585 0028602 0028606 0028612 0028617 0028625 0028629 ... d -S 3 /usr/bin/who /usr/share/locale Michael Stone David MacKenzie Joseph Arceneaux 6.10 GNU coreutils who abdlmpqrstuwHT %Y-%m-%d %H:%M %b %e %H:%M extra operand %s all count dead heading ips login 780 open O open O open O Opens files, directories, and URLs open [option] [file-list] The open utility opens one or more files, directories, or URLs. Arguments The file-list specifies the pathnames of the files, directories, or URLs that open is to open. Options Without any options, open opens the files in file-list as though you had doubleclicked each of the files' icons in the Finder. –a application Opens file-list using application. This option is equivalent to dragging file-list to the application's icon in the Finder. –b bundle Opens file-list using the application with bundle identifier bundle. A bundle identifier is a string, registered with the system, that identifies an application that can open files. For example, the bundle identifier com.apple.TextEdit specifies the TextEdit editor. –e (edit) Opens file-list using the TextEdit application. –f (file) Opens standard input as a file in the default text editor. This option does not accept file-list. –t (text) Opens file-list using the default text editor (see the "Discussion" section). Discussion Opening a file brings up the application associated with that file. For example, opening a disk image file mounts it. The open command returns immediately, without waiting for the application to launch. LaunchServices is a system framework that identifies applications that can open files. It maintains lists of available applications and user preferences about which application to use for each file type. LaunchServices also keeps track of the default text editor used by the –t and –f options. Notes When a file will be opened by a GUI application, you must run open from Terminal or another terminal emulator that is running under a GUI. Otherwise, the operation will fail. open O Examples 781 The first example mounts the disk image file backups.dmg. The disk is mounted in /Volumes, using the name it was formatted with. $ ls /Volumes House Spare Leopard $ open backups.dmg $ ls /Volumes Backups House Spare Leopard The next command opens the file picture.jpg. You must run this and the following example from a textual window within a GUI (e.g., Terminal). The application selected depends on the file attributes. If the file's type and creator code specify a particular application, open opens the file using that application. Otherwise, open uses the system's default program for handling .jpg files. $ open picture.jpg The final example opens the /usr/bin directory in the Finder. The /usr directory is normally hidden from the Finder because its invisible file attribute flag (page 931) is set. However, the open command can open any file you can access from the shell, even if it is not normally accessible from the Finder. $ open /usr/bin 782 otool O otool O otool O Displays object, library, and executable files otool options file-list The otool utility displays information about, or part of, object, library, and executable files. Arguments The file-list specifies the pathnames of files that otool is to display. Options You must use at least one of the –L, –M, –t, or –T options to specify which part of each file in file-list otool is to display. –L (libraries) Displays the names and version numbers of the shared libraries an object file uses. –M (module) Displays the module table of a shared library. –p name (print) Begins output at the symbol named name. This option requires the –t option and either the –v or –V option. –T (table of contents) Displays the table of contents of a shared library. –t (text) Displays the TEXT section of an object file. –V (very verbose) Displays even more data (than the –v option) symbolically. When displaying code, this option causes otool to display the names of called routines instead of their addresses. –v (verbose) Displays data symbolically. When displaying code, this option causes otool to display the names of instructions instead of numeric codes. Discussion The otool utility displays information about the contents and dependencies of object files. This information can prove helpful when you are debugging a program. For example, when you are setting up a chroot jail, otool can report which libraries are needed to run a given program. Some options are useful only with certain types of object modules. For example, the –T option does not report anything for a typical executable file. Notes The otool utility is part of the Developer Tools optional install. An otool –L command performs a function similar to the ldd utility on systems using the ELF binary format. Examples The examples in this section use the compiled version of the following C program: otool O 783 $ cat myname.c #include int main(void) { printf("My name is Sam."); return 0; } In the first example, otool displays the libraries the program is linked with: $ otool -L myname myname: /usr/lib/libmx.A.dylib (compatibility version 1.0.0, current version 92.0.0) /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 88.0.0) In some cases, a library used by a program will depend on other libraries. You can use otool -L on a library to see whether it uses other libraries: $ otool -L /usr/lib/libmx.A.dylib /usr/lib/libSystem.B.dylib /usr/lib/libmx.A.dylib: /usr/lib/libmx.A.dylib (compatibility version 1.0.0, current version 92.0.0) /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 88.0.0) /usr/lib/libSystem.B.dylib: /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 88.0.0) /usr/lib/system/libmathCommon.A.dylib (compatibility version 1.0.0, current ... The next example disassembles the code for the main function. When compiling programs, the compiler sometimes modifies symbol names. The compiler gives functions, such as main, a leading underscore; thus the symbol name is _main. $ otool -Vt -p _main myname myname: (__TEXT,__text) section _main: 00002ac0 mfspr r0,lr 00002ac4 stmw r30,0xfff8(r1) 00002ac8 stw r0,0x8(r1) 00002acc stwu r1,0xffb0(r1) 00002ad0 or r30,r1,r1 00002ad4 bcl 20,31,0x2ad8 00002ad8 mfspr r31,lr 00002adc addis r2,r31,0x0 00002ae0 addi r3,r2,0x4b8 00002ae4 bl _printf$LDBLStub 00002ae8 li r0,0x0 00002aec or r3,r0,r0 00002af0 lwz r1,0x0(r1) 00002af4 lwz r0,0x8(r1) 00002af8 mtspr lr,r0 00002afc lmw r30,0xfff8(r1) 00002b00 blr ... 784 paste paste paste Joins corresponding lines from files paste [option] [file-list] The paste utility reads lines from the file-list and joins corresponding lines in its output. By default output lines are separated by a TAB character. Arguments The file-list is a list of ordinary files. When you omit the file-list, paste reads from standard input. Options Under Linux, paste accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––delimiter=dlist –d dlist The dlist is a list of characters used to separate output fields. If dlist contains a single character, paste uses that character instead of the default TAB character to separate fields. If dlist contains more than one character, the characters are used in turn to separate fields and are then reused from the beginning of the list as necessary. ––serial –s Processes one file at a time; pastes horizontally. See the "Examples" section. Notes The paste utility is often used to rearrange the columns of a table. A utility, such as cut, can place the desired columns in separate files, and then paste can join them in any order. Examples The following example uses the files fnames and acctinfo. You can create these files using cut (page 652) and the /etc/passwd file. The paste command puts the fullname field first, followed by the remaining user account information. A TAB character separates the two output fields. Although this example works under Mac OS X, /etc/passwd does not contain information about most users; see "Open Directory" on page 926 for more information. $ cat fnames Sam the Great Max Wild Zach Brill Helen Simpson $ cat acctinfo sam:x:401:50:/home/sam:/bin/zsh max:x:402:50:/home/max:/bin/bash zach:x:504:500:/home/zach:/bin/tcsh hls:x:505:500:/home/hls:/bin/bash paste 785 $ paste fnames acctinfo Sam the Great sam:x:401:50:/home/sam:/bin/zsh Max Wild max:x:402:50:/home/max:/bin/bash Zach Brill zach:x:504:500:/home/zach:/bin/tcsh Helen Simpson hls:x:505:500:/home/hls:/bin/bash The next examples use the files p1, p2, p3, and p4. In the last example in this group, the –d option gives paste a list of characters to use to separate output fields: $ cat 1 one ONE $ cat 2 two TWO extra $ cat 3 three THREE $ cat 4 four FOUR p1 p2 p3 p4 $ paste 4 four FOUR p4 p3 p2 p1 3 2 three two THREE TWO extra 1 one ONE $ paste -d="+-=" p3 p2 p1 p4 3+2-1=4 three+two-one=four THREE+TWO-ONE=FOUR +extra-= The final example uses the ––serial option to paste the files one at a time: $ paste --serial p1 p2 p3 p4 1 one ONE 2 two TWO extra 3 three THREE 4 four FOUR 786 pax pax Creates an archive, restores files from an archive, or copies a directory hierarchy pax pax [options] [pattern-list] pax –w [options] [source-files] pax –r [options] [pattern-list] pax –rw [options] [source-files] destination-directory The pax utility has four modes of operation: List mode displays the list of files in an archive, create mode places multiple files into a single archive file, extract mode restores files from an archive, and copy mode copies a directory hierarchy. The archive files created by pax can be saved on disk, removable media, or a remote system. A major option determines the mode pax runs in. List mode (absence of a major option) reads an archive from standard input and displays a list of files stored in the archive. Create mode (–w for write) reads a list of ordinary or directory filenames from the command line or standard input and writes the resulting archive file to standard output. Extract mode (–r for read) reads an archive from standard input and extracts files from that archive; you can restore all files from the archive or only those whose names match a pattern. Copy mode (–rw) reads a list of ordinary or directory filenames from the command line or standard input and copies the files to an existing directory. Arguments In create mode, pax reads the names of files that will be included in the archive from source-files on the command line. If source-files is not present, pax reads the names from standard input, one filename per line. By default, in extract mode pax extracts, and in list mode pax displays the names of, all files in the archive it reads from standard input. You can choose to extract or display the names of files selectively by supplying a pattern-list. If the name of a file in the archive matches a pattern in the pattern-list, pax extracts that file or displays that filename; otherwise, it ignores the file. The pax patterns are similar to shell wildcards (page 136) except they match slashes (/) and a leading period (.) in a filename. See the fnmatch man page for more information about pax patterns (OS X only). In copy mode, pax does not create destination-directory. Thus destination-directory must exist before you give pax a copy mode command. Options A major option specifies the mode in which pax operates: create, extract, or copy. Major Options Three options determine the mode in which pax operates. You must include zero or one of these options. Without a major option, pax runs in list mode. –r (read) Reads the archive from standard input and extracts files. Without a pattern-list on the command line, pax extracts all files from the archive. With a pax 787 pattern-list, pax extracts only those files whose names match one of the patterns in the pattern-list. The following example extracts from the root.pax archive file on an external drive only those files whose names end in .c: $ pax -r *.c < /Volumes/Backups/root.pax The backslash prevents the shell from expanding the argument to pax. * before it passes the –w (write) Constructs an archive from the files named on the command line or standard input. These files may be ordinary or directory files. When the files come from standard input, each must appear on a separate line. The archive is written to standard output. The find utility frequently generates the filenames that pax uses. The following command builds an archive of the /Users directory and writes it to the archive file named Users.pax on /Volumes/Backups: # find -d /Users -print | pax -w > /Volumes/Backups/Users.pax The –d option causes find to search for files in a depth-first manner, reducing the likelihood of permissions problems when you restore the files from the archive. See the discussion of this find option on page 647. –rw (copy) Copies files named on the command line or standard input from one place on the system to another. Instead of constructing an archive file containing the files named on standard input, pax copies them to the destination-directory (the last argument on the pax command line). The effect is the same as if you had created an archive in create mode and then extracted the files in extract mode, except using copy mode avoids creating an archive. The following example copies the working directory hierarchy to the /Users/max/code directory. The code directory must exist before you give this command. Make sure that ~max/code is not a subdirectory of the working directory or you will perform a recursive copy. $ pax -rw . ~max/code Other Options The following options alter the behavior of pax. These options work with one or more of the major options. –c (complement) Reverses the sense of the test performed on pattern-list. Files are listed or put in the archive only if they do not match any of the patterns in pattern-list. –f archive (file) Uses archive as the name of the archive file. In list and extract modes, this option reads from archive instead of standard input. In create mode, it writes to archive instead of standard output. You can use this option to access a device on another system on a network; see the ––file option to tar (page 847) for more information. 788 pax –H (partial dereference) For each file that is a symbolic link, copies the file the link points to, not the symbolic link itself. This option affects all files and treats files that are not symbolic links normally. See page 623 for an example of the use of the –H versus –L options. –l (link) In copy mode, when possible, makes hard links instead of copying files. –L (dereference) For each file that is a symbolic link, copies the file the link points to, not the symbolic link itself. This option affects all files and treats files that are not symbolic links normally. See page 623 for an example of the use of the –H versus –L options. –P (no dereference) For each file that is a symbolic link, copies the symbolic link, not the file the link points to. This option affects all files and treats files that are not symbolic links normally. This behavior is the default for pax. See page 625 for an example of the use of the –P option. –p preserve-list Preserves or discards the file attributes specified by preserve-list. The preservelist is a string of one or more letters as shown in Table V-25. By default pax preserves file access and modification times but does not preserve ownership or file permissions. This option works in extract and copy modes only. Table V-25 Preserve flags Letter Meaning a Discard the access time e Preserve everything m Discard the modification time o Preserve ownership p Preserve permissions –s subcmd Executes subcmd, a substitution command, on filenames while storing them in any of the modes. The subcmd has the following format: s/search-string/replacement-string/[gp] The pax utility replaces occurrences of the regular expression search-string with replacement-string. A trailing g indicates a global replacement; without it pax replaces only the first instance of search-string in each filename. A trailing p (for print) causes pax to display each substitution it makes. The subcmd is similar to vim's search and replace feature described on page 178, except it lacks an address. pax 789 –v (verbose) In list mode, displays output similar to that produced by ls –l. In other modes, displays a list of files being processed. –X In copy and create modes, prevents pax from searching directories in filesystems other than those holding source-files. –x format In create mode, writes the archive in format format, as shown in Table V-26. If you do not specify a format, pax writes a (POSIX) tar format file (ustar in the table). –z (gzip) In create mode, compresses archives using gzip. In extract mode, decompresses archives using gunzip. Table V-26 Discussion Archive formats format Description cpio The format specified for cpio archives in POSIX sv4cpio The format used for cpio archives under UNIX System V, release 4 tar The historical Berkeley tar format ustar The POSIX tar format (default) The pax utility is a general replacement for tar, cpio, and other archive programs. In create and copy modes, pax processes specified directories recursively. In list and extract modes, if a pattern in the pattern-list matches a directory name, pax lists or extracts the files from the named directory. Notes There is no native pax format. Instead, the pax utility can read and write archives in a number of formats. By default it creates POSIX tar format archives. See Table V26 for the list of formats pax supports. The pax utility determines the format of an archive file; it does not allow you to specify an archive format (–x option) in list or extract mode. Under Mac OS X version 10.4 and later, pax copies extended attributes (page 928). Examples In the first example, pax creates an archive named corres.0901.pax. This archive stores the contents of the corres directory. $ pax -w corres > corres.0901.pax The –w option puts pax in create mode, where the list of files to be put in an archive is supplied by command-line arguments (corres in the preceding example) or from standard input. The pax utility sends the archive to standard output. The preceding example redirects that output to a file named corres.0901.pax. 790 pax Next, without any options, pax displays a list of files in the archive it reads from standard input: $ pax < corres.0901.pax corres corres/hls corres/max corres/max/0823 corres/max/0828 corres/max/0901 corres/memo1 corres/notes The following command uses the –f option to name the input file and performs the same function as the preceding command: $ pax -f corres.0901.pax When pax reads an archive file, as in the previous examples, it determines the format of the file. You do not have to (nor are you allowed to) specify the format. The next example, run under Mac OS X, attempts to create an archive of the /etc directory hierarchy in the default tar format. In this case pax is run by a user with root privileges because some of the files in this hierarchy cannot be read by ordinary users. Because pax does not follow (dereference) symbolic links by default, and because under Mac OS X /etc is a symbolic link, pax copies the link and not the directory that the link points to. The first command creates the archive, the second command shows the link in the archive, and the third command uses the –v option to show that it is a link: # pax -w -f /tmp/etc.tar /etc # pax -f /tmp/etc.tar /etc # pax -v -f /tmp/etc.tar lrwxr-xr-x 1 root admin 0 May 21 01:48 /etc => private/etc pax: ustar vol 1, 1 files, 10240 bytes read, 0 bytes written. The –L option follows (dereferences) links. In the next example, pax creates the desired archive of /etc: # pax -wLf /tmp/etc.tar /etc # pax -f /tmp/etc.tar /etc /etc/6to4.conf /etc/AFP.conf /etc/afpovertcp.cfg /etc/aliases /etc/aliases.db /etc/amavisd.conf /etc/amavisd.conf.personal /etc/appletalk.cfg ... pax 791 The next example uses pax to create a backup of the memos directory, preserving ownership and file permissions. The destination directory must exist before you give pax a copy mode command. $ mkdir memos.0625 $ pax -rw -p e memos memos.0625 $ ls memos.0625 memos The preceding example copies the memos directory into the destination directory. You can use pax to make a copy of a directory in the working directory without putting it in a subdirectory. In the next example, the –s option causes pax to replace the name of the memos directory with the name . (the name of the working directory) as it writes files to the memos.0625 directory: $ pax -rw -p e -s /memos/./ memos memos.0625 The final example uses find to build a list of files that begin with the string memo and that are located in the working directory hierarchy. The pax utility writes these files to an archive in cpio format. The output from pax goes to standard output, which is sent through a pipe to bzip2, which compresses it before writing it to the archive file. $ find . -type f -name "memo*" | pax -w -x cpio | bzip2 > memos.cpio.bz2 792 plutil O plutil O plutil O Manipulates property list files plutil [options] file-list The plutil utility converts property list files between formats and checks their syntax. Arguments Options The file-list specifies the pathnames of one or more files that plutil is to manipulate. –convert format Converts file-list to format, which must be either xml1 or binary1. –e extension Gives output files a filename extension of extension. –help Displays a help message. –lint Checks the syntax of files (default). –o file (output) Names the converted file file. –s (silent) Does not display any messages for successfully converted files. Discussion The plutil utility converts files from the XML property list format to binary format, and vice versa. The plutil utility can read—but not write—the older plain-text property list format. Notes The plutil utility accepts a double hyphen (––) to mark the end of the options on the command line. For more information refer to "Common Options" on page 603. Examples The following example checks the syntax of the file named java.plist: $ plutil java.plist java.plist: OK The next example shows the output of plutil as it checks a damaged property list file: $ plutil broken.plist broken.plist: XML parser error: Encountered unexpected element at line 2 (plist can only include one object) Old-style plist parser error: Malformed data byte group at line 1; invalid hex plutil O 793 The next example converts the file StartupParameters.plist to binary format, overwriting the original file: $ plutil -convert binary1 StartupParameters.plist The final example converts the binary format property list file loginwindow.plist in the working directory to XML format and, because of the –o option, puts the converted file in /tmp/lw.p: $ plutil -convert xml1 -o /tmp/lw.p loginwindow.plist 794 pr pr Paginates files for printing pr pr [options] [file-list] The pr utility breaks files into pages, usually in preparation for printing. Each page has a header with the name of the file, date, time, and page number. The pr utility takes its input from files you specify on the command line or from standard input. The output from pr goes to standard output and is frequently redirected through a pipe to a printer. Arguments The file-list is a list of the pathnames of text files that you want pr to paginate. When you omit the file-list, pr reads from standard input. Options Under Linux, pr accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. You can embed options within the file-list. An embedded option affects only those files following it on the command line. ––show-control-chars –c Displays control characters with a caret (^; for example, ^H). Displays other nonprinting characters as octal numbers preceded by a backslash. L ––columns=col –col Displays output in col columns with a default of one. This option may truncate lines and cannot be used with the –m (––merge) option. ––double-space –d Double-spaces the output. ––form-feed –F Uses a FORMFEED character to skip to the next page rather than filling the current page with NEWLINE characters. ––header=head –h head Displays head at the top of each page instead of the filename. If head contains SPACEs, you must enclose it within quotation marks. ––length=lines –l lines Sets the page length to lines lines. The default is 66 lines. ––merge –m Displays all specified files simultaneously in multiple columns. This option cannot be used with the –col (––columns) option. ––number-lines=[c[num]] –n[c[num]] Numbers the lines of output. The pr utility appends the character c to the pr 795 number to separate it from the contents of the file (the default is a TAB). The num specifies the number of digits in each line number (the default is 5). ––indent=spaces –o spaces Indents the output by spaces characters (specifies the left margin). ––separator=c –s[c] Separates columns with the single character c (defaults to TAB when you omit c). By default pr uses TABs as separation characters to align columns unless you use the –w option, in which case nothing separates the columns. ––omit-header –t Causes pr not to display its five-line page header and trailer. The header that pr normally displays includes the name of the file, the date, time, and page number. The trailer is five blank lines. ––width=num –w num Sets the page width to num columns. This option is effective only with multicolumn output (the –m [––merge] or –col [––columns] option). ––pages=firstpage[:lastpage] +firstpage[:lastpage] Output begins with the page numbered firstpage and ends with lastpage. Without lastpage, pr outputs through the last page of the document. The short version of this option begins with a plus sign, not a hyphen. Mac OS X does not accept the lastpage argument; printing always continues through the end of the document. Notes When you use the –col (––columns) option to display the output in multiple columns, pr displays the same number of lines in each column (with the possible exception of the last). Examples The first command shows pr paginating a file named memo and sending its output through a pipe to lpr for printing: $ pr memo | lpr Next memo is sent to the printer again, this time with a special heading at the top of each page. The job is run in the background. $ pr -h 'MEMO RE: BOOK' memo | lpr & [1] 4904 Finally pr displays the memo file on the screen, without any header, starting with page 3: $ pr -t +3 memo ... 796 ps ps Displays process status ps ps [options] [process-list] The ps utility displays status information about processes running on the local system. Arguments The process-list is a comma- or SPACE-separated list of PID numbers. When you specify a process-list, ps reports on just the processes in that list. Options Under Linux, the ps utility accepts three types of options, each preceded by a different prefix. You can intermix the options. See the ps man page for details. Two hyphens: One hyphen: No hyphens: GNU (long) options UNIX98 (short) options BSD options Options preceded by a double hyphen (––) or no hyphen work under Linux only. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. –A (all) Reports on all processes. Also –e. –e (everything) Reports on all processes. Also –A. –f (full) Displays a listing with more columns of information. ––forest f Displays the process tree (no hyphen before the f). L –l (long) Produces a long listing showing more information about each process. See the "Discussion" section for a description of the columns this option displays. ––no-headers Omits the header. This option is useful if you are sending the output to another program. L ––user usernames –uusernames Reports on processes being run by usernames, a comma-separated list of the names or UIDs of one or more users on the local system. –w (wide) Without this option ps truncates output lines at the right side of the screen. This option extends the display to 132 columns; it wraps around one more line, if needed. Use this option twice to extend the display to an unlimited number of columns, which is the default if you redirect the output of ls. Discussion Without any options, ps displays the statuses of all active processes controlled by your terminal or screen. Table V-27 lists the heading and content of each of the four columns ps displays. ps 797 Table V-27 Column headings I Heading Meaning PID The process identification number. TTY (terminal) The name of the terminal that controls the process. TIME The number of hours, minutes, and seconds the process has been running. CMD The command line the process was called with. The command is truncated to fit on one line. Use the –w option to see more of the command line. The columns that ps displays depend on your choice of options. Table V-28 lists the headings and contents of the most common columns. Table V-28 Column headings II The column titles differ, depending on the type of option you use. This table shows the headings for UNIX98 (one-hyphen) options. Heading Meaning %CPU The percentage of total CPU time the process is using. Because of the way Linux/OS X handles process accounting, this figure is approximate, and the total of %CPU values for all processes may exceed 100%. %MEM (memory) The percentage of RAM the process is using. COMMAND or CMD The command line the process was called with. The command is truncated to fit on one line. Use the –w option to see more of the command line. This column is always displayed last on a line. F (flags) The flags associated with the process. NI (nice) PID PPID The nice value of the process (page 773). The process identification number. (parent PID) The process identification number of the parent process. PRI (priority) The priority of the process. RSS (resident set size) The number of blocks of memory the process is using. SIZE or SZ The size, in blocks, of the core image of the process. STIME or START The date the process started. 798 ps Table V-28 Column headings II (continued) Heading STAT or S Meaning (status) The status of the process as specified by one or more letters from the following list: < D L N R S T W X Z TIME TTY High priority Sleeping and cannot be interrupted Pages locked in memory (real-time and custom I/O) Low priority Available for execution (in the run queue) Sleeping Either stopped or being traced Has no pages resident in RAM Dead Zombie process that is waiting for its child processes to terminate before it terminates The number of minutes and seconds that the process has been running. (terminal) The name of the terminal controlling the process. USER or UID The username of the user who owns the process. WCHAN If the process is waiting for an event, the address of the kernel function that caused the process to wait. This value is 0 for processes that are not waiting or sleeping. (wait channel) Notes Use top (page 858) to display process status information dynamically. Examples The first example shows ps, without any options, displaying the user's active processes. The first process is the shell (bash), and the second is the process executing the ps utility. $ ps PID TTY 2697 pts/0 3299 pts/0 TIME CMD 00:00:02 bash 00:00:00 ps With the –l (long) option, ps displays more information about the processes: $ ps -l F S UID 000 S 500 000 R 500 PID 2697 3300 PPID 2696 2697 C PRI 0 75 0 76 NI ADDR 0 0 - SZ WCHAN 639 wait4 744 - The –u option shows information about the specified user: TTY pts/0 pts/0 TIME CMD 00:00:02 bash 00:00:00 ps ps 799 $ ps -u root PID TTY 1 ? 2 ? 3 ? 4 ? 5 ? ... TIME 00:00:01 00:00:00 00:00:00 00:00:00 00:00:00 CMD init kthreadd migration/0 ksoftirqd/0 watchdog/0 The ––forest option causes ps to display what the man page describes as an "ASCII art process tree." Processes that are children of other processes appear indented under their parents, making the process hierarchy, or tree, easier to see. $ ps -ef --forest UID PID PPID root 1 0 root 2 1 ... root 785 1 root 839 1 root 3305 839 max 3307 3305 max 3308 3307 max 3774 3308 ... root 1040 1 root 3351 1040 root 3402 3351 root 3416 3402 root 3764 3416 root 3773 3764 ps and kill C STIME TTY 0 Jul22 ? 0 Jul22 ? TIME CMD 00:00:03 init 00:00:00 [kevent] 0 0 0 0 0 Jul22 Jul22 Aug01 Aug01 Aug01 Aug01 ? ? ? ? pts/1 pts/1 00:00:00 /usr/sbin/apmd -p 10 -w 5 -W -P 00:00:01 /usr/sbin/sshd 00:00:00 _ /usr/sbin/sshd 00:00:00 _ /usr/sbin/sshd 00:00:00 _ -bash 00:00:00 _ ps -ef --forest 0 0 0 0 0 0 Jul22 Aug01 Aug01 Aug01 Aug01 Aug01 ? tty2 tty2 tty2 tty2 tty2 00:00:00 login -- root 00:00:00 _ -bash 00:00:00 _ make modules 00:00:00 _ make -C drivers CFLA 00:00:00 _ make -C scsi mod 00:00:00 _ ld -m elf_i3 The next sequence of commands shows how to use ps to determine the PID number of a process running in the background and how to terminate that process using kill. In this case it is not necessary to use ps because the shell displays the PID number of the background processes. The ps utility verifies the PID number. The first command executes find in the background. The shell displays the job and PID numbers of the process, followed by a prompt. $ find ~ -name memo -print > memo.out & [1] 3343 Next ps confirms the PID number of the background task. If you did not already know this number, using ps would be the only way to obtain it. $ ps PID 3308 3343 3344 TTY pts/1 pts/1 pts/1 TIME 00:00:00 00:00:00 00:00:00 CMD bash find ps Finally kill (page 729) terminates the process: $ kill 3343 $ RETURN [1]+ Terminated $ find ~ -name memo -print >memo.out 800 rcp rcp Copies one or more files to or from a remote system rcp rcp [options] source-file destination-file rcp [options] source-file-list destination-directory The rcp utility copies one or more ordinary files between two systems that can communicate over a network. rcp is not secure security The rcp utility uses host-based trust, which is not secure, to authorize files to be copied. Use scp (page 810) when it is available. Arguments The source-file, source-file-list, and destination-file are the pathnames of the ordinary files, and the destination-directory is the pathname of a directory file. A pathname that does not contain a colon (:) is the name of a file on the local system. A pathname of the form name@host:path names a file on the remote system named host. The path is relative to the home directory of the user name (unless path is an absolute pathname). When you omit the name@ portion of the destination, a relative pathname is relative to the home directory on the host of the user giving the rcp command. Like cp, rcp has two modes of operation: The first copies one file to another, and the second copies one or more files to a directory. The source-file[-list] is a list of the name(s) of the file(s) that rcp will copy; destination-file is the name rcp assigns to the resulting copy of the file, or destination-directory is the name of the directory rcp puts the copied files in. When rcp copies files to a destination-directory, the files maintain their original simple filenames. Options –p (preserve) Sets the modification times and file access permissions of each copy to match those of the source-file. When you do not use –p, rcp uses the filecreation mask (umask; see page 870) on the remote system to modify the access permissions. –r (recursive) When a file in the source-file-list is a directory, copies the contents of that directory and any subdirectories into the destination-directory. You can use this option only when the destination is a directory. Notes You must have an account on the remote system to copy files to or from it using rcp. The rcp utility does not prompt for a password but rather uses several alternative methods to verify you have the authority to read or write files on the remote system. rcp 801 One method requires that the name of the local system be specified in the /etc/hosts.equiv file on the remote system. If the name is there, rcp allows you to copy files if your usernames are the same on both systems and your account on the remote system has the necessary permissions to access files there. Authorization can also be specified on a per-user basis. With this method the remote user's home directory must contain a file named ~/.rhosts that lists trusted remote systems and users. Your local and remote user names do not have to match but your local username must appear on the line in the remote ~/.rhosts file that starts with the name of the local system. See page 803 for a sample .rhosts file. If you use a wildcard (such as *) in a remote pathname, you must quote the wildcard character or pathname so the wildcard is interpreted by the shell on the remote system and not by the local shell. As with cp, if the destination-file exists before you execute rcp, rcp overwrites the file. Examples The first example copies the files with filenames ending in .c into the archives directory on the remote system named bravo. Because the command does not specify a username, rcp uses the local user's username on the remote system. Because the command does not specify the full pathname of the archives directory, rcp assumes it is a subdirectory of the user's home directory on bravo. Each of the copied files retains its simple filename. $ rcp *.c bravo:archives The next example copies memo from the /home/sam directory on bravo to the working directory on the local system: $ rcp bravo:/home/sam/memo . Next rcp copies the files named memo.new and letter to Sam's home directory on the remote system bravo. The absolute pathnames of the copied files on bravo are /home/sam/memo.new and /home/sam/letter, respectively: $ rcp memo.new letter bravo:/home/sam The final command copies all the files in Sam's reports directory on bravo to the oldreports directory on the local system, preserving the original modification dates and file access permissions on the copies: $ rcp -p 'bravo:reports/*' oldreports 802 renice renice renice Changes the priority of a process renice priority process-list renice –n increment process-list The renice utility alters the priority of a running process. An ordinary user can decrease the priority of a process that he owns. Only a user running with root privileges can increase the priority of a process or alter the priority of another user's process. Arguments The process-list specifies the PID numbers of the processes that are to have their priorities altered. Each process has its priority set to priority or, using the second format, has its priority incremented by a value of increment (which can be negative). Options The options, which you can specify throughout the process-list, change the interpretation of the arguments that follow them on the command line. –p (process) Interprets the following arguments as process ID (PID) numbers (default). –u (user) Interprets the following arguments as usernames or user ID numbers. Notes The range of priorities is from –20 (the highest priority) to +20 (the lowest priority). Higher (more positive) priority values mean the kernel schedules a job less often. Lower (more negative) values cause the job to be scheduled more often. When a user running with root privileges schedules a job to run at the highest priority, this change can affect the performance of the system for all other jobs, including the operating system itself. For this reason you should be careful when using renice with negative values. See nice (page 773) if you want to start a process with a nondefault priority. Examples The first example decreases the priority of all tasks owned by Zach: $ renice -n 5 -u zach In the following example, root uses ps to check the priority of the process running find. The NI (nice) column shows a value of 19 and the administrator decides to increase the priority by 5: # ps -l UID PID PPID CPU PRI NI 501 9705 9701 0 31 0 501 10548 9705 0 12 19 # renice -n -5 10548 VSZ 27792 27252 RSS WCHAN 856 516 - STAT Ss RN TT p1 p1 TIME COMMAND 0:00.15 -bash 0:00.62 find / rlogin 803 Logs in on a remote system rlogin [option] remote-system The rlogin utility establishes a login session on a remote system over a network. rlogin is not secure security The rlogin utility uses host-based trust, which is not secure, to authorize your login. Alternatively, it sends your password over the network as cleartext, which is not a secure practice. Use ssh (page 828) when it is available. Arguments Options Notes The remote-system is the name of a system the local system can reach over a network. –lusername (login) Logs you in on the remote system as the user specified by username. If the file named /etc/hosts.equiv located on the remote system specifies the name of the local system, the remote system will not prompt you to enter your password. Systems that are listed in the /etc/hosts.equiv file are considered to be just as secure as the local system. An alternative way to specify a trusted relationship is on a per-user basis. Each user's home directory can contain a file named ~/.rhosts that holds a list of trusted remote systems and users. See the "Examples" section for a sample .rhosts file. Examples The following example illustrates the use of rlogin. On the local system, Max's username is max; on the remote system bravo, his username is watson. The remote system prompts Max to enter a password because he is logging in using a username different from the one he uses on the local system. $ who am i max tty06 Oct 14 13:26 $ rlogin -l watson bravo Password: ~/.rhosts file If the local system is named hurrah, the following .rhosts file on bravo allows the user max to log in as the user watson without entering a password: $ cat ~watson/.rhosts hurrah max rlogin rlogin 804 rm rm Removes a file (deletes a link) rm rm [options] file-list The rm utility removes hard and/or symbolic links to one or more files. When you remove the last hard link to a file, the file is deleted. Be careful when you use rm with wildcards caution Because this utility enables you to remove a large number of files with a single command, use rm cautiously, especially when you are working with ambiguous file references. If you have any doubts about the effect of an rm command with an ambiguous file reference, first use echo with the same file reference and evaluate the list of files the reference generates. Alternatively, you can use the –i (––interactive) option. Arguments The file-list is a list of the files rm will delete. Options Under Linux, rm accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––force –f Without asking for your consent, removes files for which you do not have write access permission. This option also suppresses informative messages if a file does not exist. ––interactive –i Asks before removing each file. If you use ––recursive with this option, rm also asks you before examining each directory. ––recursive –r Deletes the contents of the specified directory, including all its subdirectories, and the directory itself. Use this option with caution. ––verbose Notes –v Displays the name of each file as it is removed. To delete a file, you must have execute and write access permission to the parent directory of the file, but you do not need read or write access permission to the file itself. If you are running rm interactively (that is, if rm's standard input is coming from the keyboard) and you do not have write access permission to the file, rm displays your access permission and waits for you to respond. If your response starts with a y or Y, rm deletes the file; otherwise, it takes no action. If standard input is not coming from a keyboard, rm deletes the file without querying you. Refer to page 106 for information on hard links and page 108 for information on symbolic links. Page 110 includes a discussion about removing links. Refer to the rmdir utility (page 806) if you need to remove an empty directory. rm 805 When you want to remove a file that begins with a hyphen, you must prevent rm from interpreting the filename as an option. One way to do so is to give the special option –– (double hyphen) before the name of the file. This option tells rm that no more options follow: Any arguments that come after it are filenames, even if they look like options. Use shred to remove a file securely security Using rm does not securely delete a file—it is possible to recover a file deleted with rm. Use the shred utility to delete files more securely. See the example "Wiping a file" on page 659 for another method of deleting files more securely. Examples The following commands delete files both in the working directory and in another directory: $ rm memo $ rm letter memo1 memo2 $ rm /home/sam/temp The next example asks the user before removing each file in the working directory and its subdirectories: $ rm -ir . This command is useful for removing filenames that contain special characters, especially SPACEs, TABs, and NEWLINEs. (You should not create filenames containing these characters on purpose, but it may happen accidentally.) 806 rmdir rmdir rmdir Removes directories rmdir directory-list The rmdir utility deletes empty directories. Arguments The directory-list is a list of pathnames of empty directories that rmdir removes. Options Under Linux, rmdir accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––ignore-fail-on-non-empty Suppresses the message rmdir normally displays when it fails because a directory is not empty. With the ––parents option, rmdir does not quit when it finds a directory that is not empty. L ––parents –p Removes a hierarchy of empty directories. ––verbose –v Displays the names of directories as they are removed. L Notes Use the rm utility with the –r option if you need to remove directories that are not empty, together with their contents. Examples The following command deletes the empty literature directory from the working directory: $ rmdir literature The next command removes the letters directory, using an absolute pathname: $ rmdir /home/sam/letters The final command removes the letters, march, and 05 directories, assuming the directories are empty except for other directories named in the path: $ rmdir -p letters/march/05 rsh 807 rsh rsh [option] host [command-line] The rsh utility runs command-line on host by starting a shell on the remote system. Without a command-line, rsh calls rlogin, which logs you in on the remote system. rsh is not secure security The rsh utility uses host-based trust, which is not secure, to authorize your login. Alternatively, it sends your password over the network as cleartext, which is not a secure practice. Use ssh (page 828) when it is available. Arguments Options Notes The host is the name of the remote system. The rsh utility runs command-line on the remote system. You must quote special characters in command-line so the local shell does not expand them prior to passing them to rsh. –lusername (login) Logs you in on the remote system as the user specified by username. If the file named /etc/hosts.equiv located on the remote system specifies the name of the local system, the remote system will not prompt you to enter your password. Systems that are listed in the /etc/hosts.equiv file are considered as secure as the local system. An alternative way to specify a trusted relationship is on a per-user basis. Each user's home directory can contain a file named ~/.rhosts that holds a list of trusted remote systems and users. See "Examples" under rlogin (page 803) for a sample .rhosts file. Examples In the first example, Max uses rsh to obtain a listing of the files in his home directory on bravo: $ rsh bravo ls cost_of_living info preferences work Next the output of the previous command is redirected to the file bravo.ls. Because the redirection character (>) is not escaped, it is interpreted by the local shell and the file bravo.ls is created on the local system. rsh Executes commands on a remote system 808 rsh $ rsh bravo ls > bravo_ls $ cat bravo_ls cost_of_living info preferences work The next example quotes the redirection character (>) so the file bravo.ls is created on the remote system (bravo), as shown by ls run on bravo: $ rsh bravo ls ">" bravo.ls $ rsh bravo ls bravo.ls cost_of_living info preferences work In the final example, rsh without command-line logs in on the remote system. Max has used the –l watson option to log in on bravo as watson. The ~watson/.rhosts file must be configured to allow Max to log in on the account in this manner. See "Examples" under rlogin (page 803) for a sample .rhosts file. $ rsh -l watson bravo Last login: Sat Jul 30 16:13:53 from kudos $ hostname bravo $ exit rlogin: connection closed. rsync 809 Copies files and directory hierarchies securely over a network rsync [options] [[user@]from-host:]source-file [[user@]to-host:][destination-file] The rsync (remote synchronization) utility copies an ordinary file or directory hierarchy locally or from the local system to or from another system on a network. By default, this utility uses OpenSSH to transfer files and the same authentication mechanism as OpenSSH; as a consequence, it provides the same security as OpenSSH. The rsync utility prompts for a password when it needs one. Alternatively you can use the rsyncd daemon as a transfer agent. See Chapter 14 for information on rsync tip See Chapter 14 starting on page 583 for information on rsync. rsync rsync 810 scp scp Securely copies one or more files to or from a remote system scp scp [[user@]from-host:]source-file [[user@]to-host:][destination-file] The scp (secure copy) utility copies an ordinary or directory file from one system to another on a network. This utility uses ssh to transfer files and the same authentication mechanism as ssh; as a consequence, it provides the same security as ssh. The scp utility prompts for a password when it needs one. Arguments The from-host is the name of the system you are copying files from and the to-host is the system you are copying to. When you do not specify a host, scp assumes the local system. The user on either system defaults to the user on the local system who is giving the command; you can specify a different user with user@. The scp utility permits you to copy between two remote systems. The source-file is the file you are copying, and the destination-file is the resulting copy. You can specify plain or directory files as relative or absolute pathnames. A relative pathname is relative to the specified or implicit user's home directory. When the source-file is a directory, you must use the –r option to copy its contents. When the destination-file is a directory, each of the source files maintains its simple filename. If source-file is a single file, you can omit destination-file and the copied file will have the same simple filename as source-file. Options –Pport Connects to port port on the remote system. Be aware that ssh uses (lowercase) –p to specify a port. –p (preserve) Preserves the modification and access times as well as the permissions of the original file. –q (quiet) Does not display the progress meter. –r (recursive) Recursively copies a directory hierarchy. –v (verbose) Displays debugging messages about the connection and transfer. This option is useful if things are not going as expected. Notes The scp utility belongs to the OpenSSH suite of secure network connectivity tools. See the "Discussion" section on page 829 for a discussion of OpenSSH security. Refer to "First-time authentication" on page 829 for information about a message you may get when using scp to connect to a remote system for the first time. scp 811 Using this utility, you can copy files from or to the local system or between two remote systems. Make sure you have read permission for the file you are copying and write permission for the directory you are copying it into. You must quote a wildcard character (such as *) in a remote pathname so it is interpreted by the shell on the remote system and not by the local shell. As with cp, if the destination-file exists before you run scp, scp overwrites the file. Examples The first example copies the files with filenames ending in .c from the working directory on the local system into the ~sam/archives directory on bravo. The wildcard character is not quoted so the local shell will expand it. Because archives is a relative pathname, scp assumes it is a subdirectory of Sam's home directory on bravo. Each of the copied files retains its simple filename. $ scp *.c sam@bravo:archives Next Max copies the directory structure under ~max/memos on the system named bravo to ~sam/max.memos.bravo on kudos. He must have the necessary permissions to write to Sam's home directory on kudos. $ scp -r bravo:memos sam@kudos:max.memos.bravo Finally Max copies the files with filenames ending in .c from Sam's archives directory on bravo to the sam.c.bravo directory in his working directory. The wildcard character is quoted to protect it from expansion by the local shell; it will be interpreted by the remote system, bravo. $ scp -r 'sam@bravo:archives/*.c' sam.c.bravo Whenever you copy multiple files or directories, the destination—either local or remote—must be an existing directory and not an ordinary or nonexistent file. 812 sed sed Edits a file noninteractively sed sed [–n] program [file-list] sed [–n] –f program-file [file-list] The sed (stream editor) utility is a batch (noninteractive) editor. It transforms an input stream that can come from a file or standard input. It is frequently used as a filter in a pipe. Because it makes only one pass through its input, sed is more efficient than an interactive editor such as ed. Most Linux distributions provide GNU sed; Mac OS X supplies BSD sed. Chapter 13 applies to both versions. See Chapter 13 for information on sed tip See Chapter 13 starting on page 565 for information on sed. 813 SetFile O Sets file attributes SetFile [options] file-list The SetFile utility sets file attributes (page 931), including the file's type and creator codes, creation and last modification times, and attribute flags such as the invisible and locked flags. Arguments The file-list specifies the pathnames of one or more files that SetFile works on. Options The options for SetFile correspond to the options for GetFileInfo (page 717). –a flags (attribute) Sets the attribute flags specified by flags. An uppercase letter for a flag sets that flag and a lowercase letter unsets the flag. The values of unspecified flags are not changed. See Table D-2 on page 931 or the SetFile man page for a list of attribute flags. –c creator Sets the creator code to creator. –d date Sets the creation date to date. You must format the date as mm/dd/[yy]yy [hh:mm:[:ss] [AM | PM]]. If you do not specify AM or PM, SetFile assumes a 24-hour clock. You must enclose a date string that contains SPACEs within quotation marks. –m date (modification) Sets the modification date to date. The format of date is the same as that used with the –d option. –P (no dereference) For each file that is a symbolic link, sets information about the symbolic link, not the file the link points to. This option affects all files and treats files that are not symbolic links normally. See page 625 for an example of the –P option. –t type Sets the type code to type. Notes The SetFile utility is part of the Developer Tools optional install. The options to

SetFile and the corresponding options to GetFileInfo have minor differences. For example, you can specify multiple attribute flags with the –a option to SetFile but only a single flag with GetFileInfo. Also, SetFile requires a SPACE between the –a option and the list of flags; GetFileInfo does not allow a SPACE there. SetFile O SetFile O 814 SetFile O Examples The first example sets the type and creator codes of the file named arch to SIT5 and SIT!, respectively, indicating that it is a StuffIt archive. The GetFileInfo utility displays these codes. $ SetFile -t SIT5 -c SIT! arch $ GetFileInfo -c arch "SIT!" $ GetFileInfo -t arch "SIT5" The next example marks the file named secret as invisible and locked. The file will not be visible in the Finder, and most OS X applications will be unable to overwrite it. $ SetFile -a VL secret The final example clears the invisible attribute flag from every file in the working directory: $ SetFile -a v * sleep 815 Creates a process that sleeps for a specified interval sleep time sleep time-list L The sleep utility causes the process executing it to go to sleep for the time specified. Arguments Traditionally the amount of time a process sleeps is given as a single integer argument, time, which denotes a number of seconds. The time does not have to be an integer, however: You can specify a decimal fraction. Under Linux you can also append a unit specification to time: s (seconds), m (minutes), h (hours), or d (days). Under Linux you can construct a time-list by including several times on the command line: The total time the process sleeps is the sum of these times. For example, if you specify 1h 30m 100s, the process will sleep for 91 minutes and 40 seconds. Examples You can use sleep from the command line to execute a command after a period of time. The following example executes in the background a process that reminds you to make a phone call in 20 minutes (1,200 seconds): $ (sleep 1200; echo "Remember to make call.") & [1] 4660 Alternatively, under Linux, you could give the following command to get the same reminder: $ (sleep 20m; echo "Remember to make call.") & [2] 4667 You can also use sleep within a shell script to execute a command at regular intervals. For example, the per shell script executes a program named update every 90 seconds: $ cat per #!/bin/bash while true do update sleep 90 done If you execute a shell script such as per in the background, you can terminate it only by using kill. The final shell script accepts the name of a file as an argument and waits for that file to appear on the disk. If the file does not exist, the script sleeps for 1 minute and 45 seconds before checking for the file again. sleep sleep 816 sleep $ cat wait_for_file #!/bin/bash if [$# != 1]; then echo "Usage: wait_for_file filename" exit 1 fi while true do if [-f "$1"]; then echo "$1 is here now" exit 0 fi sleep 1m 45 done Under Mac OS X replace 1m 45 with 105. sort 817 sort sort [options] [file-list] The sort utility sorts and/or merges one or more text files. Arguments The file-list is a list of pathnames of one or more ordinary files that contain the text to be sorted. If the file-list is omitted, sort takes its input from standard input. Without the –o option, sort sends its output to standard output. This utility sorts and merges files unless you use the –m (merge only) or –c (check only) option. Options When you do not specify an option, sort orders the file in the machine collating sequence (usually ASCII). Without a ––key option, sort orders a file based on full lines. Use ––key to specify sort fields within a line. You can follow a ––key option with additional options without a leading hyphen; see the "Discussion" section for more information. The Mac OS X version of sort accepts long options tip Options for sort preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––ignore-leading-blanks –b Blanks (TAB and SPACE characters) normally mark the beginning of fields in the input file. Without this option, sort considers leading blanks to be part of the field they precede. This option ignores leading blanks within a field, so sort does not consider these characters in sort comparisons. –c Checks whether the file is properly sorted. The sort utility does not display anything if everything is in order. It displays a message if the file is not in sorted order and returns an exit status of 1. ––check ––dictionary-order –d Ignores all characters that are not alphanumeric characters or blanks. For example, sort does not consider punctuation with this option. –f (fold) Considers all lowercase letters to be uppercase letters. Use this option when you are sorting a file that contains both uppercase and lowercase letters. ––ignore-case ––ignore-nonprinting –i Ignores nonprinting characters. This option is overridden by the ––dictionaryorder option. ––key=start[,stop] –k start[,stop] Specifies a sort field within a line. Without this option sort orders a file based on full lines. The sort field starts at the position on the line specified by start and sort Sorts and/or merges files 818 sort ends at stop, or the end of the line if stop is omitted. The start and stop positions are in the format f[.c], where f is the field number and c is the optional character within the field. Numbering starts with 1. When c is omitted from start, it defaults to the first character in the field; when c is omitted from stop, it defaults to the last character in the field. See the "Discussion" section for further explanation of sort fields and the "Examples" section for illustrations of their use. ––merge –m Assumes that each of the multiple input files is in sorted order and merges them without verifying they are sorted. ––numeric-sort –n Sorts in arithmetic sequence; does not order lines or sort fields in the machine collating sequence. With this option, minus signs and decimal points take on their arithmetic meaning. ––output=filename –o filename Sends output to filename instead of standard output; filename can be the same as one of the names in the file-list. ––reverse –r Reverses the sense of the sort (for example, z precedes a). ––field-separator=x –t x Specifies x as the field separator. See the "Discussion" section for more information on field separators. ––unique Discussion –u Outputs repeated lines only once. When you use this option with ––check, sort displays a message if the same line appears more than once in the input file, even if the file is in sorted order. Without any options, sort bases its ordering on full lines. In the following description, a field is a sequence of characters in a line of input. Without the ––field-separator option, fields are bounded by the empty string preceding a group of one or more blanks (TAB and SPACE characters). You cannot see the empty string that delimits the fields; it is an imaginary point between two fields. Fields are also bounded by the beginning and end of the line. The line shown in Figure V-1 holds the fields Toni, SPACEBarnett, and SPACESPACESPACESPACE55020. These fields define sort fields. Sometimes fields and sort fields are the same. Sort field A sort field is a sequence of characters that sort uses to put lines in order. A sort field can contain all or part of one or more fields (Figure V-1). The ––key option specifies pairs of pointers that define subsections of each line (sort fields) for comparison. See the ––key option (page 817) for details. Leading blanks The –b option causes sort to ignore leading blanks in a sort field. If you do not use this option, sort considers each leading blank to be a character in the sort field and includes it in the sort comparison. Options You can specify options that pertain only to a given sort field by immediately following the stop pointer (or the start pointer if there is no stop pointer) with one sort Field Field 819 Field Toni.Barnett....55020 Possible sort fields Figure V-1 Fields and sort fields of the options b, d, f, i, n, or r. In this case you must not precede the option with a hyphen. Multiple sort fields Examples When you specify more than one sort field, sort examines them in the order you specify them on the command line. If the first sort field of two lines is the same, sort examines the second sort field. If these are again the same, sort looks at the third field. This process continues for all the sort fields you specify. If all the sort fields are the same, sort examines the entire line. The examples in this section demonstrate some of the features and uses of the sort utility. The examples assume the following list file is in the working directory: $ cat list Tom Winstrom Janet Dempsey Alice MacLeod David Mack Toni Barnett Jack Cooper Richard MacDonald 94201 94111 94114 94114 95020 94072 95510 This file contains a list of names and ZIP codes. Each line of the file contains three fields: the first name field, the last name field, and the ZIP code field. For the examples to work, the blanks in the file must be SPACE s, and not TABs. The first example demonstrates sort without any options—the only argument is the name of the input file. In this case sort orders the file on a line-by-line basis. If the first characters on two lines are the same, sort looks at the second characters to determine the proper order. If the second characters are the same, sort looks at the third characters. This process continues until sort finds a character that differs between the lines. If the lines are identical, it does not matter which one sort puts first. In this example, sort needs to examine only the first three characters (at most) of each line. It displays a list that is in alphabetical order by first name. $ sort list Alice MacLeod David Mack Jack Cooper Janet Dempsey Richard MacDonald Tom Winstrom Toni Barnett 94114 94114 94072 94111 95510 94201 95020 820 sort You can instruct sort to skip any number of fields and characters on a line before beginning its comparison. Blanks normally mark the beginning of a field. The next example sorts the same list by last name, the second field: The ––key=2 argument instructs sort to begin its comparison with the this field. Because there is no second pointer, the sort field extends to the end of the line. Now the list is almost in lastname order, but there is a problem with Mac. $ sort --key=2 list Toni Barnett Jack Cooper Janet Dempsey Richard MacDonald Alice MacLeod David Mack Tom Winstrom 95020 94072 94111 95510 94114 94114 94201 In the preceding example, MacLeod comes before Mack. After finding that the sort fields of these two lines were the same through the third letter (Mac), sort put L before k because it arranges lines based on ASCII character codes, in which uppercase letters come before lowercase ones. The ––ignore-case option makes sort treat uppercase and lowercase letters as equals and fixes the problem with MacLeod and Mack: $ sort --ignore-case Toni Barnett Jack Cooper Janet Dempsey Richard MacDonald David Mack Alice MacLeod Tom Winstrom --key=2 list 95020 94072 94111 95510 94114 94114 94201 The next example attempts to sort list on the third field, the ZIP code. In this case sort does not put the numbers in order but rather puts the shortest name first in the sorted list and the longest name last. The ––key=3 argument instructs sort to begin its comparison with the third field, the ZIP code. A field starts with a blank and includes subsequent blanks. In the case of the list file, the blanks are SPACEs. The ASCII value of a SPACE character is less than that of any other printable character, so sort puts the ZIP code that is preceded by the most SPACEs first and the ZIP code that is preceded by the fewest SPACEs last. $ sort --key=3 list David Mack Jack Cooper Tom Winstrom Toni Barnett Janet Dempsey Alice MacLeod Richard MacDonald 94114 94072 94201 95020 94111 94114 95510 The –b (––ignore-leading-blanks) option causes sort to ignore leading SPACEs within a field. With this option, the ZIP codes come out in the proper order. When sort determines sort 821 that MacLeod and Mack have the same ZIP codes, it compares the entire lines, putting Alice MacLeod before David Mack (because A comes before D). $ sort -b --key=3 list Jack Cooper 94072 Janet Dempsey 94111 Alice MacLeod 94114 David Mack 94114 Tom Winstrom 94201 Toni Barnett 95020 Richard MacDonald 95510 To sort alphabetically by last name when ZIP codes are the same, sort needs to make a second pass that sorts on the last name field. The next example shows how to make this second pass by specifying a second sort field and uses the –f (––ignore-case) option to keep the Mack/MacLeod problem from cropping up again: $ sort -b -f --key=3 Jack Cooper Janet Dempsey David Mack Alice MacLeod Tom Winstrom Toni Barnett Richard MacDonald --key=2 list 94072 94111 94114 94114 94201 95020 95510 The next example shows a sort command that skips not only fields but also characters. The –k 3.4 option (equivalent to ––key=3.4) causes sort to start its comparison with the fourth character of the third field. Because the command does not define an end to the sort field, it defaults to the end of the line. The sort field is the last two digits in the ZIP code. $ sort -fb -k 3.4 list Tom Winstrom 94201 Richard MacDonald 95510 Janet Dempsey 94111 Alice MacLeod 94114 David Mack 94114 Toni Barnett 95020 Jack Cooper 94072 The problem of how to sort by last name within the last two digits of the ZIP code is solved by a second pass covering the last name field. The f option following the –k 2 affects the second pass, which orders lines by last name only. $ sort -b -k 3.4 -k 2f list Tom Winstrom 94201 Richard MacDonald 95510 Janet Dempsey 94111 David Mack 94114 Alice MacLeod 94114 Toni Barnett 95020 Jack Cooper 94072 822 sort The next set of examples uses the cars data file. From left to right, the columns in this file contain each car's make, model, year of manufacture, mileage, and price: $ cat cars plym fury chevy malibu ford mustang volvo s80 ford thundbd chevy malibu bmw 325i honda accord ford taurus toyota rav4 chevy impala ford explor 1970 1999 1965 1998 2003 2000 1985 2001 2004 2002 1985 2003 73 60 45 102 15 50 115 30 10 180 85 25 2500 3000 10000 9850 10500 3500 450 6000 17000 750 1550 9500 Without any options sort displays a sorted copy of the file: $ sort cars bmw 325i chevy impala chevy malibu chevy malibu ford explor ford mustang ford taurus ford thundbd honda accord plym fury toyota rav4 volvo s80 1985 1985 1999 2000 2003 1965 2004 2003 2001 1970 2002 1998 115 85 60 50 25 45 10 15 30 73 180 102 450 1550 3000 3500 9500 10000 17000 10500 6000 2500 750 9850 The objective of the next example is to sort by manufacturer and by price within manufacturer. Unless you specify otherwise, a sort field extends to the end of the line. The –k 1 sort field specifier sorts from the beginning of the line. The command line instructs sort to sort on the entire line and then make a second pass, sorting on the fifth field all lines whose first-pass sort fields were the same (–k 5): $ sort -k 1 -k 5 cars bmw 325i 1985 chevy impala 1985 chevy malibu 1999 chevy malibu 2000 ford explor 2003 ford mustang 1965 ford taurus 2004 ford thundbd 2003 honda accord 2001 plym fury 1970 toyota rav4 2002 volvo s80 1998 115 85 60 50 25 45 10 15 30 73 180 102 450 1550 3000 3500 9500 10000 17000 10500 6000 2500 750 9850 sort 823 Because no two lines are the same, sort makes only one pass, sorting on each entire line. (If two lines differed only in the fifth field, they would be sorted properly on the first pass anyway, so the second pass would be unnecessary.) Look at the lines containing taurus and thundbd. They are sorted by the second field rather than by the fifth field, demonstrating that sort never made a second pass and so never sorted on the fifth field. The next example forces the first-pass sort to stop at the end of the first field. The –k 1,1 option specifies a start pointer of the first character of the first field and a stop pointer of the last character of the first field. When you do not specify a character within a start pointer, it defaults to the first character; when you do not specify a character within a stop pointer, it defaults to the last character. Now taurus and thundbd are properly sorted by price. But look at explor: It is less expensive than the other Fords, but sort has it positioned as the most expensive. The sort utility put the list in ASCII collating sequence order, not in numeric order: 9500 comes after 10000 because 9 comes after 1. $ sort -k 1,1 -k 5 cars bmw 325i 1985 chevy impala 1985 chevy malibu 1999 chevy malibu 2000 ford mustang 1965 ford thundbd 2003 ford taurus 2004 ford explor 2003 honda accord 2001 plym fury 1970 toyota rav4 2002 volvo s80 1998 115 85 60 50 45 15 10 25 30 73 180 102 450 1550 3000 3500 10000 10500 17000 9500 6000 2500 750 9850 The –n (numeric) option on the second pass puts the list in the proper order: $ sort -k 1,1 -k 5n cars bmw 325i 1985 115 chevy impala 1985 85 chevy malibu 1999 60 chevy malibu 2000 50 ford explor 2003 25 ford mustang 1965 45 ford thundbd 2003 15 ford taurus 2004 10 honda accord 2001 30 plym fury 1970 73 toyota rav4 2002 180 volvo s80 1998 102 450 1550 3000 3500 9500 10000 10500 17000 6000 2500 750 9850 The next example again demonstrates that, unless you instruct it otherwise, sort orders a file starting with the field you specify and continuing to the end of the line. It does not make a second pass unless two of the first sort fields are the same. Because there is no stop pointer on the first sort field specifier, the sort field for the 824 sort first pass includes the third field through the end of the line. Although this example sorts the cars by years, it does not sort the cars by model within manufacturer within years (ford thundbd comes before ford explor, these lines should be reversed). $ sort -k 3 -k 1 cars ford mustang 1965 plym fury 1970 bmw 325i 1985 chevy impala 1985 volvo s80 1998 chevy malibu 1999 chevy malibu 2000 honda accord 2001 toyota rav4 2002 ford thundbd 2003 ford explor 2003 ford taurus 2004 45 73 115 85 102 60 50 30 180 15 25 10 10000 2500 450 1550 9850 3000 3500 6000 750 10500 9500 17000 Specifying an end to the sort field for the first pass allows sort to perform its secondary sort properly: $ sort -k 3,3 -k 1 cars ford mustang 1965 plym fury 1970 bmw 325i 1985 chevy impala 1985 volvo s80 1998 chevy malibu 1999 chevy malibu 2000 honda accord 2001 toyota rav4 2002 ford explor 2003 ford thundbd 2003 ford taurus 2004 45 73 115 85 102 60 50 30 180 25 15 10 10000 2500 450 1550 9850 3000 3500 6000 750 9500 10500 17000 The next examples demonstrate important sorting techniques: putting a list in alphabetical order, merging uppercase and lowercase entries, and eliminating duplicates. The unsorted list follows: $ cat short Pear Pear apple pear Apple Following is a plain sort: $ sort short Apple Pear Pear apple pear sort 825 The following folded sort is a good start, but it does not eliminate duplicates: $ sort -f short Apple apple Pear Pear pear The –u (unique) option eliminates duplicates but without the –f the uppercase entries come first: $ sort -u short Apple Pear apple pear When you attempt to use both –u and –f, some of the entries get lost: $ sort -uf short apple Pear Two passes is the answer. Both passes are unique sorts, and the first folds lowercase letters onto uppercase ones: $ sort -u -k 1f -k 1 short Apple apple Pear pear 826 split split Divides a file into sections split split [options] [filename [prefix]] The split utility breaks its input into 1,000-line sections named xaa, xab, xac, and so on. The last section may be shorter. Options can change the sizes of the sections and lengths of the names. Arguments The filename is the pathname of the file that split processes. If you do not specify an argument or if you specify a hyphen (–) instead of the filename, split reads from standard input. The prefix is one or more characters that split uses to prefix the names of the files it creates; the default is x. Options Under Linux, split accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––suffix-length=len –a len Specifies that the filename suffix is len characters long (the default is 2). ––bytes=n[u] –b n[u] Breaks the input into files that are n bytes long. The u is an optional unit of measure that can be k (kilobyte or 1,024-byte blocks) or m (megabyte or 1,048,576-byte blocks). If you include the unit of measure, split counts by this unit in place of bytes. Under Linux, u can be b (512-byte blocks) or any of the suffixes listed in Table V-1 on page 603. ––numeric-suffixes –d Specifies numeric suffixes instead of alphabetic suffixes. ––lines=num –l num Breaks the input into files that are num lines long (the default is 1,000). Discussion By default split names the first file it creates xaa. The x is the default prefix. You can change the prefix with the prefix argument on the command line. You can change the number of characters in each filename following the prefix with the ––suffix-length option. Examples By default split breaks a file into 1,000-line sections with the names xaa, xab, xac, and so on. The wc utility with the –l option shows the number of lines in each file. The last file, xar, is smaller than the rest. split 827 $ split /usr/share/dict/words $ wc -l * 1000 xaa 1000 xab 1000 xac ... 1000 xdt 569 xdu 98569 total The next example uses the prefix argument to specify a filename prefix of SEC and uses –c (––suffix-length) to change the number of letters in the filename suffix to 3: $ split $ ls SECaaa SECaab ... SECaaj -a 3 /usr/share/dict/words SEC SECaak SECaal SECaau SECaav SECabe SECabf SECabo SECabp SECaby SECabz SECaci SECacj SECacs SECact SECadc SECadd SECaat SECabd SECabn SECabx SECach SECacr SECadb SECadl SECadm SECadn 828 ssh ssh Securely executes commands on a remote system ssh ssh [option] [user@]host [command-line] The ssh utility logs in on a remote system and starts a shell. Optionally, ssh executes a command on the remote system and logs off. The ssh utility, which can replace rsh and rlogin, provides secure, encrypted communication between two systems over an unsecure network. Arguments The host is the system you want to log in or run a command on. Unless you have one of several kinds of authentication established, ssh prompts you for a username and password for the remote system. When ssh is able to log in automatically, it logs in as the user running the ssh command or as user if you specify user@ on the ssh command line. The command-line runs on the remote system. Without command-line, ssh logs in on the remote system. You must quote special characters in command-line if you do not want the local shell to expand them. Options –f (not foreground) Sends ssh to the background after asking for a password and before executing command-line. This option is useful when you want to run the command-line in the background but must supply a password. Its use implies –n. –l user (login) Attempts to log in as user. This option is equivalent to using user@ on the command line. –n (null) Redirects standard input to ssh to come from /dev/null. See –f. –p port Connects to port port on the remote system. Be aware that scp uses (uppercase) –P to specify a port. –q (quiet) Suppresses warning and diagnostic messages. –t (tty) Allocates a pseudo-tty to the ssh process on the remote system. Without this option, when you run a command on a remote system, ssh does not allocate a tty (terminal) to the process. Instead, ssh attaches standard input and standard output of the remote process to the ssh session—that is normally, but not always, what you want. This option forces ssh to allocate a tty on the remote system so programs that require a tty will work. –v (verbose) Displays debugging messages about the connection and transfer. This option is useful if things are not going as expected. ssh 829 –X (X11) Turns on X11 forwarding. You may not need this option—X11 forwarding may be turned on in a configuration file. –x (X11) Turns off X11 forwarding. Discussion OpenSSH Using public key encryption, OpenSSH provides two levels of authentication: server and client/user. First the client verifies that it is connected to the correct server. Then OpenSSH encrypts communication between the systems. Once a secure, encrypted connection has been established, OpenSSH makes sure the user is authorized to log in on or copy files to and from the server. After verifying the system and user, OpenSSH can allow various services to pass through the connection, including interactive shell sessions (ssh), remote command execution (ssh), file copying (scp), FTP services (sftp), X11 client/server connections, and TCP/IP port tunneling. ssh The ssh utility allows you to log in on a remote system over a network. For example, you might choose to use a remote system to access a special-purpose application or to take advantage of a device that is available only on that system, or you might use a remote system because you know it is faster or less busy than the local system. While they are traveling, many businesspeople use ssh on a laptop to log in on a system at company headquarters. From a GUI you can use several systems simultaneously by logging in on each one from a different terminal emulator window. X11 forwarding With trusted X11 forwarding turned on, it is a simple matter to run an X11 program over an ssh connection: Run ssh from a terminal emulator running on an X11 server and give an X11 command such as xclock; the graphical output appears on the local display. To turn on trusted X11 forwarding, set ForwardX11Trusted to yes in the /etc/ssh/ssh_config configuration file. To increase security (and in some cases reduce usability) set ForwardX11Trusted to no. known_hosts, ssh_known_hosts Two files list the hosts the local system has connected to and positively identified: ~/.ssh/known_hosts (user) and /etc/ssh/ssh_known_hosts (global). No one except the owner (a user working with root privileges in the case of the second file) should be able to write to either of these files. No one except the owner should have any access to a ~/.ssh directory. First-time authentication When you connect to an OpenSSH server for the first time, the OpenSSH client prompts you to confirm you are connected to the right system. This check can help prevent a man-in-the-middle attack: The authenticity of host 'plum (192.168.0.10)' can't be established. RSA key fingerprint is d1:9d:1b:5b:97:5c:80:e9:4b:41:9a:b7:bc:1a:ea:a1. Are you sure you want to continue connecting (yes/no)? yes Warning: Permanently added 'plum,192.168.0.10' (RSA) to the list of known hosts. 830 ssh Before you respond to the preceding query, make sure you are logging in on the correct system and not on an imposter. If you are not sure, a telephone call to someone who logs in on that system locally can help verify you have accessed the right system. When you answer yes (you must spell it out), the client appends the server's public host key (the single line in the /etc/ssh/ssh_host_rsa_key.pub or /etc/ssh/ssh_host_dsa_key.pub file on the server) to the user's ~/.ssh/known_hosts file on the local system, creating the ~/.ssh directory if necessary. To ensure that it can keep track of which line in known_hosts applies to which server, OpenSSH prepends the name of the server and the server's IP address to the line. When you subsequently use OpenSSH to connect to that server, the client verifies it is connected to the correct server by comparing this key to the one supplied by the server. After a remote system's public key is stored in one of the known-hosts files, if the remote system supplies a different fingerprint when the systems connect, OpenSSH displays the following message and does not complete the connection: @@@ @ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @ @@@ IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY! Someone could be eavesdropping on you right now (man-in-the-middle attack)! It is also possible that the RSA host key has just been changed. The fingerprint for the RSA key sent by the remote host is f1:6f:ea:87:bb:1b:df:cd:e3:45:24:60:d3:25:b1:0a. Please contact your system administrator. Add correct host key in /home/sam/.ssh/known_hosts to get rid of this message. Offending key in /home/sam/.ssh/known_hosts:1 RSA host key for plum has changed and you have requested strict checking. Host key verification failed. If you see this message, you may be the subject of a man-in-the-middle attack. More likely, however, something on the remote system has changed, causing it to supply a new fingerprint. Check with the remote system's administrator. If all is well, remove the offending key from the specified file (the third line from the bottom in the preceding example points to the line you need to remove) and try connecting again. You can use ssh-keygen with the –R option followed by the name of a host to remove a hashed entry. In this scenario you will be subject to first-time authentication (page 829) again as OpenSSH verifies you are connecting to the correct system. Follow the same steps as when you initially connected to the remote host. Authorized keys: automatic login You can configure OpenSSH so you do not have to enter a password each time you connect to a server (remote system). To set up this feature, you need to generate a personal authentication key on the client (local system), place the public part of the key on the server, and keep the private part of the key on the client. When you connect to the server, it issues a challenge based on the public part of the key. The private part of the key must then respond properly to this challenge. If the client provides the appropriate response, the server logs you in. ssh 831 The first step in setting up an automatic login is to generate your personal authentication keys. First check whether these authentication keys already exist on the local system (client) by looking in ~/.ssh for either id_dsa and id_dsa.pub or id_rsa and id_rsa.pub. If one of these pairs of files is present, skip the next step (i.e., do not create a new key). On the client, the ssh-keygen utility creates the public and private parts of an RSA key. The key's randomart image is a visual representation of the public key; it is designed to be easy to recall. Display of the randomart image by a client is controlled by the VisualHostKey declaration in the ssh_config file. ssh-keygen $ ssh-keygen -t rsa Generating public/private rsa key pair. Enter file in which to save the key (/home/sam/.ssh/id_rsa):RETURN Enter passphrase (empty for no passphrase):RETURN Enter same passphrase again:RETURN Your identification has been saved in /home/sam/.ssh/id_rsa. Your public key has been saved in /home/sam/.ssh/id_rsa.pub. The key fingerprint is: f2:eb:c8:fe:ed:fd:32:98:e8:24:5a:76:1d:0e:fd:1d sam@peach The key's randomart image is: +--[RSA 2048]----+ | oE| | o . . o| | . o + ..| | . + o o | ... Replace rsa with dsa to generate DSA keys. In this example, the user pressed RETURN in response to each query. You have the option of specifying a passphrase (10–30 characters is a good length) to encrypt the private part of the key. There is no way to recover a lost passphrase. See the following security tip for more information about the passphrase. When you encrypt your personal key security The private part of the key is kept in a file that only you can read. If a malicious user compromises your account, an account that can use sudo to gain root privileges, or the root account on the local system, that user then has access to your account on the remote system because she can read the private part of your personal key. Encrypting the private part of your personal key protects the key and, therefore, restricts access to the remote system should someone compromise your local account. However, if you encrypt your personal key, you must supply the passphrase you used to encrypt the key each time you use the key, negating the benefit of not having to type a password when logging in on the remote system. Also, most passphrases that you can easily remember can be cracked quite quickly by a powerful computer. A better idea is to store the private keys on a removable medium, such as a USB flash drive, and use your ~/.ssh directory as the mount point for the filesystem stored on this drive. You may want to encrypt these keys with a passphrase in case you lose the flash drive. 832 ssh The ssh-keygen utility generates two keys: a private key or identification in ~/.ssh/id_rsa and a public key in ~/.ssh/id_rsa.pub. No one except the owner should be able to write to either of these files, and only the owner should be able to read from the private key file. authorized_keys Examples To enable you to log in on or copy files to and from another system without supplying a password, first create a ~/.ssh directory with permissions set to 700 on the server (remote system). Next copy ~/.ssh/id_rsa.pub from the client (local system) to a file named ~/.ssh/authorized_keys on the server (remote system). Set its permissions to 600 so that no one except the owner can read from or write to this file. Now when you run ssh or scp to access the server, you do not have to supply a password. To make the server even more secure, you can disable password authentication by setting PasswordAuthentication to no in /etc/ssh/sshd_config (remove the # from the beginning of the PasswordAuthentication line and change the yes to no). In the first example, Max uses ssh to display a list of the files in his home directory on kudos: $ ssh kudos ls max@kudos's password: Work code graphs reports Next the output of the previous command is redirected to the file kudos_ls. Because the redirection character (>) is not escaped, it is interpreted by the local shell, and the file kudos_ls is created on the local system. $ ssh kudos ls > kudos_ls max@kudos's password: $ cat kudos_ls Work code graphs reports The next example quotes the entire command that will run on the remote system. As a result, the local shell does not interpret the redirection character (>) but rather passes it to the remote shell. The file kudos.ls is created on the remote system (kudos), as shown by ls run on kudos: $ ssh kudos max@kudos's $ ssh kudos max@kudos's Work code graphs kudos.ls reports "ls > kudos.ls" password: ls password: ssh 833 The next command does not quote the pipe symbol (|). As a result the pipe is interpreted by the local shell, which sends the output of the remote ls to standard input of less on the local system: $ ssh kudos ls | less Next ssh executes a series of commands, connected with pipes, on a remote system. The commands are enclosed within single quotation marks so the local shell does not interpret the pipe symbols and all the commands are run on the remote system. $ ssh kudos 'ps -ef | grep nmbd | grep -v grep | cut -c10-15 |xargs kill -1' The output of ps is piped through grep, which passes all lines containing the string nmbd to another invocation of grep. The second grep passes all lines not containing the string grep to cut (page 652). The cut utility extracts the process ID numbers and passes them to xargs (page 881), which kills the listed processes by calling kill to issue a HUP signal (kill –1). In the following example, ssh without command-line logs in on the remote system. Here Max has used watson@kudos to log in on kudos as watson: $ ssh watson@kudos watson@kudos's password: Last login: Sat Sep 17 06:51:59 from bravo $ hostname kudos $ exit Max now decides to change the password for his watson login on kudos: $ ssh watson@kudos passwd watson@kudos's password: (current) UNIX password: por Max stops as soon as he sees passwd (running on kudos) displaying his password: He knows something is wrong. For the passwd to work, it must run with a tty (terminal) so it can turn off character echo (stty –echo); then it will not display passwords as the user enters them. The –t option solves the problem by associating a pseudo-tty with the process running passwd on the remote system: $ ssh -t watson@kudos passwd watson@kudos's password: Changing password for watson (current) UNIX password: New UNIX password: Retype new UNIX password: passwd: all authentication tokens updated successfully Connection to kudos closed. $ The –t option is also useful when you are running a program that uses a characterbased/pseudographical interface. 834 ssh The following example uses tar (page 846) to create an archive file of the contents of the working directory hierarchy. The f – option causes tar to send its output to standard output. A pipe sends the output of tar running on the local system, via ssh, to dd (page 658) running on the remote system. $ cat buwd #! /bin/bash # back up the working directory to the user's # home directory on the remote system specified # by $machine # remote system: machine=speedy dir=$(basename $(pwd)) filename=$$.$dir.tar echo Backing up $(pwd) to your home directory on $machine tar -cf - . | ssh $machine "dd obs=256k of=$filename" echo done. Name of file on $machine is $filename stat 835 stat stat [options] [file-list] The stat utility displays information about files. Arguments The file-list specifies the pathnames of one or more files that stat displays information about. Without a file-list, stat displays information about standard input. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. Without any options, stat displays all available information about each file it processes. ––format=fmt –c fmt Formats output using fmt. See the stat man page for more information. L –F (file type) Displays a slash (/) after each directory, an asterisk (*) after each executable file, an at sign (@) after a symbolic link, an equal sign (=) after a socket, and a pipe (|) after a FIFO. O –f (filesystem) Displays filesystem information instead of file information. L –f format Formats output using format. The format string is similar to that used by printf. See the stat man page for more information. O ––dereference –L For each file that is a symbolic link, displays information about the file the link points to, not the symbolic link itself. This option treats files that are not symbolic links normally. –l (long) Uses the same format as ls –l (page 93). O –q (quiet) Suppresses error messages. O –s (shell) Displays information in a format that can be used to initialize shell variables. See the "Examples" section. O –x (Linux) Displays a more verbose format that is compatible with the version of stat found on Linux. O Examples The Linux and Mac OS X versions of stat display different information. The examples show the output from the Linux version. stat Displays information about files 836 stat The first example displays information about the /bin/bash file: $ stat /bin/bash File: `/bin/bash' Size: 813912 Blocks: 1600 IO Block: 4096 regular file Device: 812h/2066d Inode: 146674 Links: 1 Access: (0755/-rwxr-xr-x) Uid: (1003/ sam) Gid: (1002/ pubs) Access: 2009-05-28 11:36:01.000000000 -0700 Modify: 2008-05-12 11:36:49.000000000 -0700 Change: 2009-05-21 11:34:41.000000000 -0700 The next example displays information about the root filesystem: $ stat -f / File: "/" ID: 586ac700c664ac38 Namelen: 255 Type: ext2/ext3 Block size: 4096 Fundamental block size: 4096 Blocks: Total: 960602 Free: 703655 Available: 654858 Inodes: Total: 488640 Free: 467723 strings 837 Displays strings of printable characters strings [options] file-list The strings utility displays strings of printable characters from object and other nontext files. Arguments The file-list is a list of files that strings processes. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––all ––print-file-name –a Processes whole files. Without this option strings processes only the initialized and loaded parts of an object file. –f Precedes each string with the name of the file that the string comes from. L ––bytes=min –min Displays strings of characters that are at least min characters long (the default is 4). Discussion The strings utility can help you determine the contents of nontext files. One notable application for strings is determining the owner of files in a lost+found directory. Examples The following example displays strings of four or more printable characters in the executable file for the man utility. If you did not know what this file was, these strings could help you determine that it was the man executable. $ strings /usr/bin/man ... --Man-- next: %s [view (return) | skip (Ctrl-D) | quit (Ctrl-C)] format: %d, save_cat: %d, found: %d cannot write to %s in catman mode creating temporary cat for %s can't write to temporary cat for %s can't create temporary cat for %s cat-saver exited with status %d found ultimate source file %s ... strings strings 838 stty stty Displays or sets terminal parameters stty stty [options] [arguments] Without any arguments, stty displays parameters that affect the operation of the terminal or terminal emulator. For a list of some of these parameters and an explanation of each, see the "Arguments" section. The arguments establish or change parameters. Options Under Linux, stty accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––all –a Reports on all parameters. This option does not accept arguments. –F device Affects device. Without this option stty affects the device attached to standard input. You can change the characteristics of a device only if you own its device file or if you are working with root privileges. L ––file=device –f device Affects device. Performs the same function as –F. O ––save –g Generates a report of the current settings in a format you can use as arguments to another stty command. This option does not accept arguments. Arguments The arguments to stty specify which terminal parameters stty is to alter. To turn on each of the parameters that is preceded by an optional hyphen (indicated in the following list as [–]) specify the parameter without the hyphen. To turn it off, use the hyphen. Unless specified otherwise, this section describes the parameters in their on states. Special Keys and Characteristics columns n Sets the line width to n columns. ek (erase kill) Sets the erase and line kill keys to their default values. Many systems use DELETE and CONTROL-U, respectively, as the defaults. erase x Sets the erase key to x. To specify a control character, precede x with CONTROL-V (for example, use CONTROL-V CONTROL-H to indicate CONTROL-H) or use the notation ^h, where ^ is a caret (SHIFT-6 on most keyboards). intr x Sets the interrupt key to x. See erase x for conventions. kill x Sets the line kill key to x. See erase x for conventions. rows n Sets the number of screen rows to n. stty 839 sane Sets the terminal parameters to values that are usually acceptable. The sane argument is useful when several stty parameters have changed, making it difficult to use the terminal to run stty to set things right. If sane does not appear to work, try entering the following characters: CONTROL-J stty sane CONTROL-J susp x (suspend) Sets the suspend (terminal stop) key to x. See erase x for conventions. werase x (word erase) Sets the word erase key to x. See erase x for conventions. Modes of Data Transmission [–]cooked See raw. L cooked See sane. O [–]cstopb (stop bits) Selects two stop bits (–cstopb specifies one stop bit). [–]parenb (parity enable) Enables parity on input and output. When you specify –parenb, the system does not use or expect a parity bit when communicating with the terminal. [–]parodd (parity odd) Selects odd parity (–parodd selects even parity). [–]raw The normal state is –raw. When the system reads input in its raw form, it does not interpret the following special characters: erase (usually DELETE), line kill (usually CONTROL-U), interrupt execution (CONTROL-C), and EOF (CONTROL-D). In addition, the system does not use parity bits. Reflecting the humor that is typical of Linux's heritage, under Linux you can also specify –raw as cooked. Treatment of Characters [–]echo Echoes characters as they are typed (full-duplex operation). If a terminal is half- duplex and displays two characters for each one it should display, turn the echo parameter off (–echo). Use –echo when the user is entering passwords. [–]echoe (echo erase) The normal setting is echoe, which causes the kernel to echo the charac- ter sequence BACKSPACE SPACE BACKSPACE when you use the erase key to delete a character. The effect is to move the cursor backward across the line, removing characters as you delete them. [–]echoke (echo kill erase) The normal setting is echoke. When you use the kill character to delete a line while this option is set, all characters back to the prompt are erased on the current line. When this option is negated, pressing the kill key moves the cursor to the beginning of the next line. [–]echoprt (echo print) The normal setting is –echoprt, which causes characters to disappear as you erase them. When you set echoprt, characters you erase are displayed between a backslash (\) and a slash (/). For example, if you type the word sort and then erase it by pressing BACKSPACE four times, Linux displays sort\tros/ when echoprt is set. If you use the kill character to delete the entire line, having echoprt set causes the entire line to be displayed as if you had BACKSPACEd to the beginning of the line. 840 stty [–]lcase For uppercase-only terminals, translates all uppercase characters into lowercase as they are entered (also [–]LCASE). L [–]nl Accepts only a NEWLINE character as a line terminator. With –nl in effect, the system accepts a RETURN character from the terminal as a by a NEWLINE to the terminal in place of a NEWLINE. NEWLINE but sends a RETURN followed [–]tabs Transmits each TAB character to the terminal as a TAB character. When tabs is turned off (–tabs), the kernel translates each TAB character into the appropriate number of SPACEs and transmits them to the terminal (also [–]tab3). Job Control Parameters [–]tostop Stops background jobs if they attempt to send output to the terminal (–tostop allows background jobs to send output to the terminal). Notes The name stty is an abbreviation for set teletypewriter, or set tty (page 867), the first terminal UNIX was run on. Today stty is commonly thought of as meaning set terminal. The shells retain some control over standard input when you use them interactively. As a consequence, a number of the options available with stty appear to have no effect. For example, the command stty –echo appears to have no effect under tcsh: tcsh $ stty -echo tcsh $ date Thu May 28 16:53:01 PDT 2009 While stty –echo does work when you are using bash interactively, stty –echoe does not. However, you can still use these options to affect shell scripts and other utilities. $ cat testit #!/bin/bash stty -echo echo -n "Enter a value: " read a echo echo "You entered: $a" stty echo $ ./testit Enter a value: You entered: 77 The preceding example does not display the user's response to the Enter a value: prompt. The value is retained by the a variable and is displayed by the echo "You entered: $a" statement. Examples The first example shows that stty without any arguments displays several terminal operation parameters. (Your system may display more or different parameters.) The character following the erase = is the erase key. A ^ preceding a character indicates stty 841 a CONTROL key. In the example the erase key is set to CONTROL-H. If stty does not display the erase character, it is set to its default value of DELETE. If it does not display a kill character, it is set to its default of ^U. $ stty speed 38400 baud; line = 0; erase = ^H; Next the ek argument returns the erase and line kill keys to their default values: $ stty ek The next display verifies the change. The stty utility does not display either the erase character or the line kill character, indicating both are set to their default values: $ stty speed 38400 baud; line = 0; The next example sets the erase key to CONTROL-H. The CONTROL-V quotes the that the shell does not interpret it and it is passed to stty unchanged: CONTROL-H so $ stty erase CONTROL–V CONTROL–H $ stty speed 38400 baud; line = 0; erase = ^H; Next stty sets the line kill key to CONTROL-X. This time the user entered a caret (^) followed by an x to represent CONTROL-X. You can use either a lowercase or uppercase letter. $ stty kill ^X $ stty speed 38400 baud; line = 0; erase = ^H; kill = ^X; Now stty changes the interrupt key to CONTROL-C: $ stty intr CONTROL–VCONTROL–C In the following example, stty turns off TABs so the appropriate number of SPACEs is sent to the terminal in place of a TAB. Use this command if a terminal does not automatically expand TABs. $ stty -tabs If you log in under Linux and everything appears on the terminal in uppercase letters, give the following command and then check the CAPS LOCK key. If it is set, turn it off. $ STTY -LCASE Turn on lcase if you are using a very old terminal that cannot display lowercase characters. Although no one usually changes the suspend key from its default of CONTROL-Z, you can. Give the following command to change the suspend key to CONTROL-T: $ stty susp ^T 842 sysctl O sysctl O sysctl O Displays and alters kernel variables sysctl [options] [variable-list] sysctl [options] –w [var=value ...] The sysctl utility displays and alters kernel variables, including kernel tuning parameters. Arguments Options The variable-list is a list of kernel variables whose values sysctl displays. In the second format, each value is assigned to the variable named var. –a (all) Displays all kernel variables. –b (binary) Displays kernel variables as binary data without terminating NEWLINEs. –n (no label) Displays variables without labels. Discussion The sysctl utility provides access to a number of kernel variables, including the maximum number of processes the kernel will run at one time, the filename used for core files, and the system security level. Some variables cannot be altered or can be altered only in certain ways. For example, you can never lower the security level. Examples The sysctl utility is commonly used for tuning the kernel. The process limits can be displayed by anyone, but can be altered only by a user who is working with root privileges. The following example shows a user changing the maximum number of processes per UID: $ sysctl -a | grep proc kern.maxproc = 532 kern.maxfilesperproc = 10240 kern.maxprocperuid = 266 kern.aioprocmax = 16 kern.proc_low_pri_io = 0 ... # sysctl -w kern.maxprocperuid=200 kern.maxprocperuid: 266 -> 200 tail 843 tail tail [options] [file-list] The tail utility displays the last part, or end, of a file. Arguments The file-list is a list of pathnames of the files that tail displays. When you specify more than one file, tail displays the filename of each file before displaying lines from that file. If you do not specify an argument or, under Linux, if you specify a hyphen (–) instead of a filename, tail reads from standard input. Options Under Linux, tail accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. –b [+]n Counts by 512-byte blocks instead of lines. The n argument is an integer that specifies the number of blocks. Thus the command tail –b 5 displays the last five blocks of a file. See the note about using a plus sign (+) in the next option. O ––bytes=[+]n[u] –c [+]n[u] Counts by bytes (characters) instead of lines. The n argument is an integer that specifies the number of bytes. Thus the command tail –c 5 displays the last five bytes of a file. Under Linux only, the u is an optional multiplicative suffix as described on page 602, except that tail uses a lowercase k for kilobyte (1,024-byte blocks). If you include a multiplicative suffix, tail counts by this unit instead of by bytes. If you put a plus sign (+) in front of n, tail counts from the start of the file instead of the end. The tail utility still displays characters through the end of the file, even though it starts counting from the beginning. Thus tail –c +5 causes tail to display from the fifth character through the end of the file. ––follow –f After copying the last line of the file, tail enters an endless loop, waiting and copying additional lines from the file if the file grows. If you specify multiple files in file-list with this option, tail includes a new header each time it displays output from a different file so you know which file is being added to. This option is useful for tracking the progress of a process that is running in the background and sending its output to a file. The tail utility continues to wait indefinitely, so you must use the interrupt key to terminate it. See also the –s option. tail Displays the last part (tail) of a file 844 tail ––lines=[+]n[u] –n [+]n[u] Counts by lines (the default). The n argument is an integer that specifies the number of lines. Under Linux, the u is an optional unit of measure; see the –c (––bytes) option for an explanation of its use. Although it is not documented, you can use ±n to specify a number of lines without using this option. If you put a plus sign (+) in front of n, tail counts from the start of the file instead of the end. The tail utility still displays lines through the end of the file, even though it starts counting from the beginning. Thus tail –n +5 causes tail to display from the fifth line through the last line of the file. ––quiet –q Suppresses header information when you specify multiple files in file-list. L ––sleep-interval=n –s n When used with –f, causes tail to sleep for n seconds between checks for additional output. L Notes The tail utility displays the last ten lines of its input by default. Examples The examples are based on the eleven file: $ cat eleven line one line two line three line four line five line six line seven line eight line nine line ten line eleven First tail displays the last ten lines of the eleven file (no options): $ tail eleven line two line three line four line five line six line seven line eight line nine line ten line eleven Next it displays the last three lines (–n 3 or ––lines 3) of the file: tail 845 $ tail -n 3 eleven line nine line ten line eleven The following example displays the file starting at line 8 (+8): $ tail -n +8 eleven line eight line nine line ten line eleven The next example displays the last six characters in the file (–c 6 or ––bytes 6). Only five characters are evident (leven); the sixth is a NEWLINE. $ tail -c 6 eleven leven The final example demonstrates the –f option. Here tail tracks the output of a make command, which is being sent to the file accounts.out: $ make accounts > accounts.out & $ tail -f accounts.out cc -c trans.c cc -c reports.c ... CONTROL-C $ In the preceding example, running tail with –f displays the same information as running make in the foreground and not redirecting its output to a file. However, using tail offers some advantages. First, the output of make is saved in a file. (The output would not be saved if you did not redirect its output.) Second, if you decide to do something else while make is running, you can kill tail and the screen will be free for you to use while make continues running in the background. When you are running a large job such as compiling a large program, you can use tail with the –f option to check on its progress periodically. 846 tar tar Stores or retrieves files to/from an archive file tar tar option [modifiers] [file-list] The tar (tape archive) utility creates, adds to, lists, and retrieves files from an archive file. Arguments The file-list is a list of pathnames of the files that tar archives or extracts. Options Use only one of the following options to indicate which action you want tar to take. You can alter the action of the option by following it with one or more modifiers. The Mac OS X version of tar accepts long options tip Options for tar preceded by a double hyphen (––) work under Mac OS X as well as under Linux. ––create –c Creates an archive. This option stores the files named in file-list in a new archive. If the archive already exists, tar destroys it before creating the new archive. If a file-list argument is a directory, tar copies the directory hierarchy into the archive. Without the ––file option, tar writes the archive to standard output. ––compare or ––diff –d Compares an archive with the corresponding disk files and reports on the differences. ––help Displays a list of options and modifiers, along with short descriptions of each. ––append –r Writes the files named in file-list to the end of the archive. This option leaves files that are already in the archive intact, so duplicate copies of files may appear in the archive after tar finishes. When tar extracts the files, the most recent copy of a file in the archive is the one that ends up on the disk. ––list –t (table of contents) Without a file-list, this option produces a table of contents listing all files in an archive. With a file-list, it displays the name of each file in the file-list each time it occurs in the archive. You can use this option with the ––verbose option to display detailed information about each file in an archive. ––update –u Adds the files from file-list if they are not already in the archive or if they have been modified since they were last written to the archive. Because of the additional checking required, tar runs more slowly when you specify this option. ––extract –x Extracts file-list from the archive and writes it to the disk, overwriting existing files with the same names. Without a file-list, this option extracts all files from the archive. If the file-list includes a directory, tar extracts the directory hierarchy. The tar utility attempts to keep the owner, modification time, and access privileges the same as those of the original file. If tar reads the same file more than once, the last version read will appear on the disk when tar is finished. tar 847 Modifiers ––blocking-factor n –b n Uses n as the blocking factor for creating an archive. Use this option only when tar is creating an archive directly to removable media. (When tar reads an archive, it automatically determines the blocking factor.) The value of n is the number of 512-byte blocks to write as a single block on the removable media. ––directory dir –C dir Changes the working directory to dir before processing. ––checkpoint Displays periodic messages. This option lets you know tar is running without forcing you to view all the messages displayed by ––verbose. ––exclude=file Does not process the file named file. If file is a directory, no files or directories in that directory hierarchy are processed. The file can be an ambiguous file reference; quote special characters as needed. ––file filename –f filename Uses filename as the name of the file the archive is created in or extracted from. The filename can be the name of an ordinary file or a device (e.g., a CD or USB flash drive). You can use a hyphen (–) instead of the filename to refer to standard input when creating an archive and to standard output when extracting files from an archive. The following two commands are equivalent ways of creating a compressed archive of the files in the /home directory hierarchy on /dev/sde1: $ tar -zcf /dev/sde1 /home $ tar -cf - /home | gzip > /dev/sde1 ––dereference –h For each file that is a symbolic link, archives the file the link points to, not the symbolic link itself. ––ignore-failed-read When creating an archive, tar normally quits with a nonzero exit status if any of the files in file-list is unreadable. This option causes tar to continue processing, skipping unreadable files. ––bzip2 –j Uses bzip2 (pages 60 and 615) to compress/decompress files when creating an archive and extracting files from an archive. ––tape–length n –L n Asks for a new medium after writing n *1,024 bytes to the current medium. This feature is useful when you are building archives that are too big to fit on a single USB flash drive, partition, CD, or other storage device. ––touch –m Sets the modification time of the extracted files to the time of extraction. Without this option tar attempts to maintain the modification time of the original file. 848 tar ––one-file-system When a directory name appears in file-list while it is creating an archive, tar recursively processes the files and directories in the named directory hierarchy. With this option tar stays in the filesystem that contains the named directory and does not process directories in other filesystems. Under OS X, you can use –l (lowercase "l") in place of ––one-file-system. Under Linux, –l is used for a different purpose. ––absolute-names –P The default behavior of tar is to force all pathnames to be relative by stripping leading slashes from them. This option disables this feature, so absolute pathnames remain absolute. ––sparse –S Linux allows you to create sparse files (large, mostly empty files). The empty sections of sparse files do not take up disk space. When tar extracts a sparse file from an archive, it normally expands the file to its full size. As a result, when you restore a sparse file from a tar backup, the file takes up its full space and may no longer fit in the same disk space as the original. This option causes tar to handle sparse files efficiently so they do not take up unnecessary space either in the archive or when they are extracted. ––verbose –v Lists each file as tar reads or writes it. When combined with the –t option, –v causes tar to display a more detailed listing of the files in the archive, showing their ownership, permissions, size, and other information. ––interactive –w Asks you for confirmation before reading or writing each file. Respond with y if you want tar to take the action. Any other response causes tar not to take the action. ––exclude-from filename –X filename Similar to the ––exclude option, except filename specifies a file that contains a list of files to exclude from processing. Each file listed in filename must appear on a separate line. ––compress or –Z Uses compress when creating an archive and uncompress when extracting files ––uncompress from an archive. ––gzip or ––gunzip Notes –z Causes tar to use gzip to compress an archive while it is being created and to decompress an archive when extracting files from it. This option also works to extract files from archives that have been compressed with the compress utility. The ––help option displays all tar options and modifiers; the ––usage option provides a brief summary of the same information. The info page on tar provides extensive information, including a tutorial for this utility. You can use ambiguous file references in file-list when you create an archive but not when you extract files from an archive. The name of a directory file within the file-list references the directory hierarchy (all files and directories within that directory). tar 849 The file that tar sends its output to by default is compilation specific; typically it goes to standard output. Use the –f option to specify a different filename or device to hold the archive. When you create an archive using a simple filename in file-list, the file appears in the working directory when you extract it. If you use a relative pathname when you create an archive, the file appears with that relative pathname, starting from the working directory when you extract it. If you use the –P option and an absolute pathname when you create an archive, and if you use the –P option when you extract files from the archive, tar extracts the file with the same pathname. Leading hyphens The tar utility does not require leading hyphens on options and modifiers. However, it behaves slightly differently with regard to the order of options and modifiers it requires with and without the hyphen. You can specify one or more modifiers following an option. With a leading hyphen, the following tar command generates an error: $ tar -cbf 10 /dev/sde1 memos tar: f: Invalid blocking factor Try `tar --help' or `tar --usage' for more information. The error occurs because the –b modifier takes an argument but is not the last modifier in a group. The same command works correctly if you omit the leading hyphen. You must use leading hyphens if you separate options: $ tar -cb 10 -f /dev/sde1 memos Examples The following example makes a copy of the /home/max directory hierarchy on a USB flash drive mounted at /dev/sde1. The v modifier causes the command to list the files it writes to the device. This command erases anything that was already on the device. The message from tar explains that the default action is to store all pathnames as relative paths instead of absolute paths, thereby allowing you to extract the files into a different directory on the disk. $ tar -cvf /dev/sde1 /home/max tar: Removing leading '/' from member names. /home/max/ /home/max/.bash_history /home/max/.bash_profile ... In the next example, the same directory is saved on the device mounted on /dev/sde1 with a blocking factor of 100. Without the v modifier, tar does not display the list of files it is writing to the device. The command runs in the background and displays any messages after the shell issues a new prompt. $ tar -cb 100 -f /dev/sde1 /home/max & [1] 4298 $ tar: Removing leading '/' from member names. 850 tar The next command displays the table of contents of the archive on the device mounted on /dev/sde1: $ tar -tvf drwxrwxrwx -rw-r--r--rw-r--r-- drwx------rw------... /dev/sde1 max/group max/group max/group max/group max/group 0 678 571 0 2799 Jun 30 21:39 2008 home/max/ Aug 6 14:12 2009 home/max/.bash_history Aug 6 14:06 2009 home/max/.bash_profile Nov 6 22:34 2009 home/max/mail/ Nov 6 22:34 2009 home/max/mail/sent-mail Next, Max creates a gzipped tar archive in /tmp/max.tgz. This approach is a popular way to bundle files that you want to transfer over a network or otherwise share with others. Ending a filename with .tgz is one convention for identifying gzipped tar archives. Another convention is to end the filename with .tar.gz. $ tar -czf /tmp/max.tgz literature The final command lists the files in the compressed archive max.tgz: $ tar -tzvf /tmp/max.tgz ... tee 851 tee tee Copies standard input to standard output and one or more files tee [options] file-list The tee utility copies standard input to standard output and to one or more files. Arguments The file-list is a list of the pathnames of files that receive output from tee. If a file in file-list does not exist, tee creates it. Options Options preceded by a double hyphen (––) work under Linux only. Options named with a single letter and preceded by a single hyphen work under Linux and OS X. Without any options, tee overwrites the output files if they exist and responds to interrupts. ––append –a Appends output to existing files rather than overwriting them. –i Causes tee not to respond to the SIGINT interrupt. O ––ignore-interrupts –i Causes tee not to respond to interrupts. L Examples In the following example, a pipe sends the output from make to tee, which copies that information to standard output and to the file accounts.out. The copy that goes to standard output appears on the screen. The cat utility displays the copy that was sent to the file. $ make accounts | tee accounts.out cc -c trans.c cc -c reports.c ... $ cat accounts.out cc -c trans.c cc -c reports.c ... Refer to page 845 for a similar example that uses tail –f rather than tee. 852 telnet telnet telnet Connects to a remote system over a network telnet [options] [remote-system] The telnet utility implements the TELNET protocol to connect to a remote system over a network. telnet is not secure security The telnet utility is not secure. It sends your username and password over the network as cleartext, which is not a secure practice. Use ssh (page 828) when it is available. Arguments Options The remote-system is the name or IP address of the remote system that telnet connects to. When you do not specify a remote-system, telnet works interactively and prompts you to enter one of the commands described in this section. –a Initiates automatic login (the default behavior under Mac OS X). –e c (escape) Changes the escape character from CONTROL-] to the character c. –K Prevents automatic login. This option is available under Mac OS X and some Linux distributions (it is the default behavior under Linux). –l username Attempts an automatic login on the remote system using username. If the remote system understands how to handle automatic login with telnet, it prompts for a password. Discussion After telnet connects to a remote system, you can put telnet into command mode by typing the escape character (usually CONTROL-]). The remote system typically reports the escape character it recognizes. To leave command mode, type RETURN on a line by itself. In command mode, telnet displays the telnet> prompt. You can use the following commands in command mode: ? (help) Displays a list of commands recognized by the telnet utility on the local system. close Closes the connection to the remote system. If you specified the name of a system on the command line when you started telnet, close has the same effect as quit: The telnet program quits, and the shell displays a prompt. If you used the open com- mand instead of specifying a remote system on the command line, close returns telnet to command mode. logout Logs you out of the remote system; similar to close. telnet 853 open remote-computer If you did not specify a remote system on the command line or if the attempt to connect to the system failed, you can specify the name of a remote system interactively with the open command. quit Quits the telnet session. z Suspends the telnet session. When you suspend a session, you return to the login shell on the local system. To resume the suspended telnet session, type fg at a shell prompt. Notes Under Linux, telnet does not attempt to log in automatically. Under OS X, telnet attempts to log in automatically. When telnet attempts to log in automatically, it uses your username on the local system unless you specify a different name using the –l option. Many computers, including non-Linux/OS X systems, support the TELNET protocol. The telnet utility is a user interface to this protocol for Linux/OS X systems that allows you to connect to many different types of systems. Although you typically use telnet to log in, the remote computer may offer other services through telnet, such as access to special databases. Examples In the following example, the user connects to the remote system named bravo. After running a few commands, the user escapes to command mode and uses the z command to suspend the telnet session so he can run a few commands on the local system. The user gives an fg command to the shell to resume using telnet. The logout command on the remote system ends the telnet session, and the local shell displays a prompt. kudos% telnet bravo Trying 192.168.0.55 ... Connected to bravo. Escape character is '^]'. login: watson Password: Last login: Wed Jul 31 10:37:16 from kudos bravo $ ... bravo $CONTROL-] telnet> z [1]+ Stopped kudos $ ... kudos $fg telnet bravo telnet bravo bravo$ logout Connection closed by foreign host. kudos $ 854 test test Evaluates an expression test test expression [expression] The test utility evaluates an expression and returns a condition code indicating the expression is either true (0) or false (not 0). You can place brackets ([]) around the expression instead of using the word test (second format). Arguments The expression contains one or more criteria (see the following list) that test evaluates. A –a separating two criteria is a Boolean AND operator: Both criteria must be true for test to return a condition code of true. A –o is a Boolean OR operator. When –o separates two criteria, one or the other (or both) of the criteria must be true for test to return a condition code of true. You can negate any criterion by preceding it with an exclamation point (!). You can group criteria with parentheses. If there are no parentheses, –a takes precedence over –o, and test evaluates operators of equal precedence from left to right. Within the expression you must quote special characters, such as parentheses, so the shell does not interpret them but rather passes them to test unchanged. Because each element, such as a criterion, string, or variable within the expression, is a separate argument, you must separate each element from other elements with a SPACE. Table V-29 lists the criteria you can use within the expression. Table V-30 on page 856 lists test's relational operators. Table V-29 Criteria Criterion Meaning string True if string has a length greater than zero. L True if string is not a null string. O –n string True if string has a length greater than zero. –z string True if string has a length of zero. string1 = string2 True if string1 is equal to string2. string1 != string2 True if string1 is not equal to string2. int1 relop int2 True if integer int1 has the specified algebraic relationship to integer int2. The relop is a relational operator from Table V-30. As a special case, –l string, which gives the length of string, may be used for int1 or int2. test Table V-29 855 Criteria (continued) Criterion Meaning file1 –ef file2 True if file1 and file2 have the same device and inode numbers. file1 –nt file2 True if file1 was modified after file2 (the modification time of file1 is newer than that of file2). file1 –ot file2 True if file1 was modified before file2 (the modification time of file1 is older than that of file2). –b filename True if the file named filename exists and is a block special file. –c filename True if the file named filename exists and is a character special file. –d filename True if the file named filename exists and is a directory. –e filename True if the file named filename exists. –f filename True if the file named filename exists and is an ordinary file. –g filename True if the file named filename exists and its setgid bit (page 96) is set. –G filename True if the file named filename exists and is associated with the group that is the primary group of the user running the command (same effective group ID). –k filename True if the file named filename exists and its sticky bit (page 980) is set. –L filename True if the file named

filename exists and is a symbolic link. –O filename True if the file named filename exists and is owned by the user running the command (same effective user ID). –p filename True if the file named filename exists and is a named pipe. –r filename True if the file named filename exists and the user running the command has read permission for it. –s filename True if the file named filename exists and contains information (has a size greater than 0 bytes). –t file-descriptor True if file-descriptor is open and associated with the screen or keyboard. The file-descriptor for standard input is 0, for standard output is 1, and for standard error is 2. –u filename True if the file named filename exists and its setuid bit (page 96) is set. –w filename True if the file named filename exists and you have write permission for it. –x filename True if the file named filename exists and you have execute/search permission for it. 856 test Table V-30 Relational operators Relational operator Meaning –eq Equal to –ge Greater than or equal to –gt Greater than –le Less than or equal to –lt Less than –ne Not equal to Notes The test command is built into the Bourne Again and TC Shells. Examples The following examples demonstrate the use of the test utility in Bourne Again Shell scripts. Although test works from a command line, it is more commonly employed in shell scripts to test input or verify access to a file. The first example prompts the user, reads a line of input into a variable, and uses the synonym for test, [], to see whether the user entered yes: $ cat user_in echo -n "Input yes or no: " read user_input if ["$user_input" = "yes"] then echo You input yes. fi The next example prompts for a filename and then uses the synonym for test, [], to see whether the user has read access permission (–r) for the file and (–a) whether the file contains information (–s): $ cat validate echo -n "Enter filename: " read filename if [-r "$filename" -a -s "$filename"] then echo File $filename exists and contains information. echo You have read access permission to the file. fi The –t 1 criterion checks whether the process running test is sending standard output to the screen. If it is, the test utility returns a value of true (0). The shell stores the exit status of the last command it ran in the $? variable. The following script tests whether its output is going to a terminal: test 857 $ cat term test -t 1 echo "This program is (=0) or is not (=1) sending its output to a terminal:" $? First term is run with the output going to the terminal: $ term This program is (=0) or is not (=1) sending its output to a terminal: 0 The next example runs term and redirects its output to a file. The contents of the file temp show that test returned 1, indicating its output was not going to a terminal. $ term > temp $ cat temp This program is (=0) or is not (=1) sending its output to a terminal: 1 858 top top Dynamically displays process status top top [options] The top utility displays information about the status of the local system, including information about current processes. Options Although top does not require the use of hyphens with options, it is a good idea to include them for clarity and consistency with other utilities. You can cause top to run as though you had specified any of the options by giving commands to the utility while it is running. See the "Discussion" section for more information. –ca Causes top to run in accumulative mode. In this mode, times and events are counted cumulatively since top started. See the top man page if you want to use the –c option to run top in another mode. O –d ss.tt (delay) Specifies ss.tt as the number of seconds and tenths of seconds of delay from one display update to the next. The default is 3 seconds. L –i Ignores idle and zombie processes (processes without a parent). L –n n (number) Specifies the number of iterations: top updates the display n times and exits. L –p n (PID) Monitors the process with a PID of n. You can use this option up to 20 times on a command line or specify n as a comma-separated list of up to 20 PID numbers. L –S (sum) Causes top to run in cumulative mode. In cumulative mode, the CPU times reported for processes include forked processes and CPU times accumulated by child processes that are now dead. L –s (secure) Runs top in secure mode, restricting the commands you can use while top is running to those that pose less of a security risk. L –s ss (seconds) Specifies ss as the number of seconds of delay from one display update to the next. The default is 1 second. O Discussion The first few lines top displays summarize the status of the local system. You can turn each of these lines on or off with the toggle switches (interactive command keys) top 859 specified in the following descriptions. The first line is the same as the output of the uptime utility and shows the current time, the amount of time the local system has been running since it was last booted, the number of users logged in, and the load averages from the last 1, 5, and 15 minutes (toggle l [lowercase "l"]). The second line indicates the number of running processes (toggle t). The next three lines report on CPU (also toggle t), memory (toggle m), and swap space (also toggle m) use. The rest of the display reports on individual processes, which are listed in order by descending CPU usage (i.e., the most CPU-intensive process is listed first). By default top displays the number of processes that fit on the screen. Table V-31 describes the meanings of the fields displayed for each process. Table V-31 Field names Name Meaning PID Process identification number USER Username of the owner of the process PR Priority of the process NI nice value (page 773) VIRT Number of kilobytes of virtual memory used by the process RES Number of kilobytes of physical (nonswapped) memory used by the process SHR Number of kilobytes of shared memory used by the process S Status of the process (see STAT on page 798) %CPU Percentage of the total CPU time the process is using %MEM Percentage of physical memory the process is using TIME[+] Total CPU time used by the process COMMAND Command line that started the process or name of the program (toggle with c) While top is running, you can issue the following commands to modify its behavior: h (help) Displays a summary of the commands you can use while top is running. L ? (help) Displays a summary of the commands you can use while top is running. O k (kill) Allows you to kill a process. Unless you are working with root privileges, you can kill only processes you own. When you use this command, top prompts you for the PID of the process and the signal to send to the process. You can enter either a signal number or a name. (See Table 10-5 on page 453 for a list of signals.) This command is disabled when you are working in secure mode. L 860 top n (number) When you give this command, top asks you to enter the number of processes you want it to display. If you enter 0 (the default), top shows as many processes as fit on the screen. q (quit) Terminates top. r (renice) Changes the priority of a running process (refer to renice on page 802). Unless you are working with root privileges, you can change the priority of only your own processes and even then only to lower the priority by entering a positive value. A user working with root privileges can enter a negative value, increasing the priority of the process. This command is disabled when you are working in secure mode. L S (switch) Toggles top between cumulative mode and regular mode. See the –S option for details. L s (seconds) Prompts for the number of seconds to delay between updates to the display (3 is the default). You may enter an integer, a fraction, or 0 (for continuous updates). Under Linux, this command is disabled when you are working in secure mode. W (write) Writes top's current configuration to your personal configuration file (~/.toprc). L SPACE Refreshes the screen. Notes The Linux and OS X versions of top are very different. Although it applies to both versions, this coverage of top is oriented toward the Linux version. Refer to the OS X top man page for more information about the OS X version. The top utility is similar to ps but periodically updates the display, enabling you to monitor the behavior of the local system over time. This utility shows only as much of the command line for each process as fits on a line. If a process is swapped out, top replaces the command line with the name of the command in parentheses. Under Linux, the top utility uses the proc filesystem. When proc is not mounted, top does not work. Requesting continuous updates is almost always a mistake. The display is updated too quickly and the system load increases dramatically. Example The following display is the result of a typical execution of top: top 861 top - 17:48:53 up 6:21, 1 user, load average: 0.71, 1.00, 0.80 Tasks: 178 total, 1 running, 176 sleeping, 0 stopped, 1 zombie Cpu(s): 16.2%us, 3.0%sy, 0.0%ni, 80.4%id, 0.0%wa, 0.3%hi, 0.0%si, 0.0%st Mem: 8188664k total, 8122636k used, 66028k free, 127200k buffers Swap: 11912112k total, 0k used, 11912112k free, 7217728k cached PID 7841 7810 12084 13428 7147 7676 13437 16542 7672 7747 1 2 3 4 ... USER mark mark mark root mark root mark mark mark root root root PR NI VIRT RES 20 0 108m 29m 20 0 675m 175m 20 0 568m 70m 10 -10 1715m 74m 20 0 449m 102m 20 0 159m 17m 20 0 300m 18m 20 0 18992 1332 20 0 323m 21m 20 0 306m 45m 20 0 4020 876 15 -5 0 0 RT -5 0 0 15 -5 0 0 SHR 12m 27m 32m 62m 14m 11m 10m 940 10m 25m 592 0 0 0 S %CPU %MEM S 15 0.4 S 8 2.2 S 7 0.9 S 5 0.9 S 4 1.3 S 1 0.2 S 1 0.2 R 1 0.0 S 0.3 S 0 0.6 S 0 0.0 S 0 0.0 S 0 0.0 S 0 0.0 TIME+ 5:14.62 24:07.87 5:14.39 6:25.19 57:57.59 0:05.14 0:21.14 0:00.16 1:09.47 1:20.28 0:01.24 0:00.00 0:00.00 0:03.04 COMMAND npviewer.bin firefox amarokapp vmware-vmx Xorg konsole firestarter top artsd vmware init kthreadd migration/0 ksoftirqd/0 862 touch touch touch Creates a file or changes a file's access and/or modification time touch [options] file-list The touch utility changes the access and/or modification time of a file to the current time or a time you specify. You can also use touch to create a file. Arguments The file-list is a list of the pathnames of the files that touch will create or update. Options Under Linux, touch accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. Without any options, touch changes the access and modification times to the current time. When you do not specify the –c (––no-create) option, touch creates files that do not exist. –a Updates the access time only, leaving the modification time unchanged. ––no-create –c Does not create files that do not exist. ––date=datestring –d datestring Updates times with the date specified by datestring. Most familiar formats are permitted for datestring. Components of the date and time not included in datestring are assumed to be the current date and time. This option may not be used with –t. L –m Updates the modification time only, leaving the access time unchanged. ––reference=file –r file Updates times with the times of file. –t [[cc]yy]nnddhhmm[.ss] Changes times to the date specified by the argument. The nn argument is the number of the month (01–12), dd is the day of the month (01–31), hh is the hour based on a 24-hour clock (00–23), and mm is the minutes (00–59). You must specify at least these fields. You can specify the number of seconds past the start of the minute with .ss. The optional cc specifies the first two digits of the year (the value of the century minus 1), and yy specifies the last two digits of the year. When you do not specify a year, touch assumes the current year. When you do not specify cc, touch assumes 20 for yy in the range 0–68 and 19 for yy in the range 69–99. This option may not be used with –d. touch 863 Examples The first three commands show touch updating an existing file. The ls utility with the –l option displays the modification time of the file. The last three commands show touch creating a file. $ ls -l program.c -rw-r--r-1 max group 5860 Apr 21 09:54 program.c $ touch program.c $ ls -l program.c -rw-r--r-1 max group 5860 Aug 13 19:01 program.c $ ls -l read.c ls: read.c: No such file or directory $ touch read.c $ ls -l read.c -rw-rw-r-1 max group 0 Aug 13 19:01 read.c The first of the following ls commands displays the file modification times; the second ls (with the –lu options) displays the file access times: $ ls -l -rw-r--r-- 1 max group 552 Jan 10 19:44 cases -rw-r--r-- 1 max group 2209 Apr 18 04:24 excerpts $ ls -lu -rw-r--r-- 1 max group 552 May 29 15:28 cases -rw-r--r-- 1 max group 2209 May 29 15:28 excerpts The next example works on the two files shown above and demonstrates the use of the –a option to change the access time only and the –t option to specify a date for touch to use instead of the current date and time. After the touch command is executed, ls shows that the access times of the files cases and excerpts have been updated but the modification times remain the same. $ touch -at 02040608 cases excerpts $ ls -l -rw-r--r-- 1 max group 552 Jan 10 19:44 cases -rw-r--r-- 1 max group 2209 Apr 18 04:24 excerpts $ ls -lu -rw-r--r-- 1 max group 552 Feb -rw-r--r-- 1 max group 2209 Feb 4 06:08 cases 4 06:08 excerpts 864 tr tr Replaces specified characters tr tr [options] string1 [string2] The tr utility reads standard input and, for each input character, either maps it to an alternate character, deletes the character, or leaves the character as is. This utility reads from standard input and writes to standard output. Arguments The tr utility is typically used with two arguments, string1 and string2. The position of each character in the two strings is important: Each time tr finds a character from string1 in its input, it replaces that character with the corresponding character from string2. With one argument, string1, and the –d (––delete) option, tr deletes the characters specified in string1. The option –s (––squeeze-repeats) replaces multiple sequential occurrences of characters in string1 with single occurrences (for example, abbc becomes abc). Ranges A range of characters is similar in function to a character class within a regular expression (page 889). GNU tr does not support ranges (character classes) enclosed within brackets. You can specify a range of characters by following the character that appears earlier in the collating sequence with a hyphen and the character that comes later in the collating sequence. For example, 1–6 expands to 123456. Although the range A–Z expands as you would expect in ASCII, this approach does not work when you use the EBCDIC collating sequence, as these characters are not sequential in EBCDIC. See "Character Classes" for a solution to this issue. Character Classes A tr character class is not the same as the character class described elsewhere in this book. (GNU documentation uses the term list operator for what this book calls a character class.) You specify a character class as '[:class:]', where class is one of the character classes from Table V-32. You must specify a character class in string1 (and not string2) unless you are performing case conversion (see the "Examples" section) or you use the –d and –s options together. Table V-32 Character classes Class Meaning alnum Letters and digits alpha Letters tr 865 Table V-32 Options Character classes (continued) Class Meaning blank Whitespace cntrl CONTROL characters digit Digits graph Printable characters but not SPACEs lower Lowercase letters print Printable characters including SPACEs punct Punctuation characters space Horizontal or vertical whitespace upper Uppercase letters xdigit Hexadecimal digits Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––complement –c Complements string1, causing tr to match all characters except those in string1. ––delete –d Deletes characters that match those specified in string1. If you use this option with the –s (––squeeze-repeats) option, you must specify both string1 and string2 (see "Notes"). ––help Summarizes how to use tr, including the special symbols you can use in string1 and string2. L ––squeeze-repeats –s Replaces multiple sequential occurrences of a character in string1 with a single occurrence of the character when you call tr with only one string argument. If you use both string1 and string2, the tr utility first translates the characters in string1 to those in string2; it then replaces multiple sequential occurrences of a character in string2 with a single occurrence of the character. ––truncate-set1 Notes –t Truncates string1 so it is the same length as string2 before processing input. L When string1 is longer than string2, the initial portion of string1 (equal in length to string2) is used in the translation. When string1 is shorter than string2, tr uses the last character of string1 to extend string1 to the length of string2. In this case tr departs from the POSIX standard, which does not define a result. 866 tr If you use the –d (––delete) and –s (––squeeze-repeats) options at the same time, tr first deletes the characters in string1 and then replaces multiple sequential occurrences of a character in string2 with a single occurrence of the character. Examples You can use a hyphen to represent a range of characters in string1 or string2. The two command lines in the following example produce the same result: $ echo abcdef | tr xyzabc $ echo abcdef | tr xyzabc 'abcdef' 'xyzabc' 'a-f' 'x-za-c' The next example demonstrates a popular method for disguising text, often called ROT13 (rotate 13) because it replaces the first letter of the alphabet with the thirteenth, the second with the fourteenth, and so forth. $ echo The punchline of the joke is ... | > tr 'A-M N-Z a-m n-z' 'N-Z A-M n-z a-m' Gur chapuyvar bs gur wbxr vf ... To make the text intelligible again, reverse the order of the arguments to tr: $ echo Gur chapuyvar bs gur wbxr vf ... | > tr 'N-Z A-M n-z a-m' 'A-M N-Z a-m n-z' The punchline of the joke is ... The ––delete option causes tr to delete selected characters: $ echo If you can read this, you can spot the missing vowels! | > tr --delete 'aeiou' If y cn rd ths, y cn spt th mssng vwls! In the following example, tr replaces characters and reduces pairs of identical characters to single characters: $ echo tennessee | tr -s 'tnse' 'srne' serene The next example replaces each sequence of nonalphabetic characters (the complement of all the alphabetic characters as specified by the character class alpha) in the file draft1 with a single NEWLINE character. The output is a list of words, one per line. $ tr -c -s '[:alpha:]' '' < draft1 The next example uses character classes to upshift the string hi there: $ echo hi there | tr '[:lower:]' '[:upper:]' HI THERE tty 867 tty tty Displays the terminal pathname tty [option] The tty utility displays the pathname of standard input if it is a terminal and displays not a tty if it is not a terminal. The exit status is 0 if standard input is a terminal and 1 if it is not. Arguments There are no arguments. Options Under Linux, tty accepts the common options described on page 603. The option preceded by a double hyphen (––) works under Linux only. The option named with a single letter and preceded by a single hyphen works under Linux and OS X. ––silent or ––quiet –s Causes tty not to print anything. The exit status of tty is set. Notes The term tty is short for teletypewriter, the terminal device on which UNIX was first run. This command appears in UNIX, and Linux has kept it for the sake of consistency and tradition. Examples The following example illustrates the use of tty: $ tty /dev/pts/11 $ echo $? 0 $ tty < memo not a tty $ echo $? 1 868 tune2fs tune2fs tune2fs Changes parameters on an ext2, ext3, or ext4 filesystem tune2fs [options] device The tune2fs utility displays and modifies filesystem parameters on ext2, ext3, and ext4 filesystems. This utility can also set up journaling on an ext2 filesystem, turning it into an ext3 filesystem. With typical filesystem permissions, tune2fs must be run by a user working with root privileges. The tune2fs utility is available under Linux only. L Arguments The device is the name of the device, such as /dev/sda8, that holds the filesystem whose parameters you want to display or modify. Options –C n (count) Sets the number of times the filesystem has been mounted without being checked to n. This option is useful for staggering filesystem checks (see "Discussion") and for forcing a check the next time the system boots. –c n (max count) Sets the maximum number of times the filesystem can be mounted between filesystem checks to n. Set n to 0 (zero) to disregard this parameter. –e behavior (error) Specifies what the kernel will do when it detects an error. Set behavior to continue (continues execution), remount-ro (remounts the filesystem readonly), or panic (causes a kernel panic). Regardless of how you set this option, an error will cause fsck to check the filesystem next time the system boots. –i n[u] (interval) Sets the maximum time between filesystem checks to n time periods. Without u or with u set to d, the time period is days. Set u to w to set the time period to weeks; use m for months. Set n to 0 (zero) to disregard this parameter. Because a filesystem check is forced only when the system is booted, the time specified by this option may be exceeded. –j (journal) Adds an ext3 journal to an ext2 filesystem. For more information on journaling filesystems see page 961. –l (list) Lists information about the filesystem. –T date (time) Sets the time the filesystem was last checked to date. The date is the time and date in the format yyyymmdd[hh[mm[ss]]]. Here yyyy is the year, nn is the number of the month (01–12), and dd is the day of the month (01–31). You must specify at least these fields. The hh is the hour based on a 24-hour clock (00–23), mm is the minutes (00–59), and .ss is the number of seconds past the start of the minute. You can also specify date as now. tune2fs 869 Discussion Checking a large filesystem can take a long time. When all filesystem checks occur at the same time, the system may boot slowly. Use the –C and/or –T options to stagger filesystem checks so they do not all happen at the same time. Example Following is the output of tune2fs run with the –l option on a typical ext3 filesystem: # /sbin/tune2fs -l /dev/sda1 tune2fs 1.41.4 (27-Jan-2009) Filesystem volume name: Last mounted on: Filesystem UUID: b6d9714e-ed5d-45b8-8023-716a669c16d8 Filesystem magic number: 0xEF53 Filesystem revision #: 1 (dynamic) Filesystem features: has_journal ext_attr resize_inode dir_index filetype needs_recovery sparse_super large_file Filesystem flags: signed_directory_hash Default mount options: (none) Filesystem state: clean Errors behavior: Continue Filesystem OS type: Linux Inode count: 624624 Block count: 2498099 Reserved block count: 124904 Free blocks: 1868063 Free inodes: 509355 First block: 0 Block size: 4096 Fragment size: 4096 Reserved GDT blocks: 609 Blocks per group: 32768 Fragments per group: 32768 Inodes per group: 8112 Inode blocks per group: 507 Filesystem created: Mon Apr 27 09:41:43 2009 Last mount time: Tue May 5 03:54:59 2009 Last write time: Tue May 5 03:54:59 2009 Mount count: 4 Maximum mount count: 31 Last checked: Mon Apr 27 09:41:43 2009 Check interval: 15552000 (6 months) Next check after: Sat Oct 24 09:41:43 2009 Reserved blocks uid: 0 (user root) Reserved blocks gid: 0 (group root) First inode: 11 Inode size: 256 Required extra isize: 28 Desired extra isize: 28 Journal inode: 8 First orphan inode: 308701 Default directory hash: half_md4 Directory Hash Seed: bceae349-a46f-4d45-a8f1-a21b1ae8d2bd Journal backup: inode blocks 870 umask umask umask Establishes the file-creation permissions mask umask [mask] The umask builtin specifies the mask the system uses to set access permissions when you create a file. This builtin works slightly differently in each of the shells. Arguments The mask can be a three-digit octal number (bash and tcsh) or a symbolic value (bash) such as you would use with chmod (page 626). The mask specifies the permissions that are not allowed. When mask is an octal number, the digits correspond to the permissions for the owner of the file, members of the group the file is associated with, and everyone else. Because the mask specifies the permissions that are not allowed, the system subtracts each of these digits from 7 when you create a file. The resulting three octal numbers specify the access permissions for the file (the numbers you would use with chmod). A mask you specify as a symbolic value also specifies the permissions that are not allowed. See the "Notes" section. Without any arguments, umask displays the file-creation permissions mask. Option Notes –S (symbolic) Displays the file-creation permissions mask symbolically. Most utilities and applications do not attempt to create files with execute permissions, regardless of the value of mask; they assume you do not want an executable file. As a result, when a utility or application such as touch creates a file, the system subtracts each of the digits in mask from 6. An exception occurs with mkdir, which does assume that you want the execute (access in the case of a directory) bit set. See the "Examples" section. The umask program is a builtin in bash and tcsh and generally goes in the initialization file for your shell (~/.bash_profile [bash] or ~/.login [tcsh]). Under bash the argument u=rwx,go=r turns off all bits in the mask for the owner and turns off the read bit in the mask for groups and other users (the mask is 0033), causing those bits to be on in file permissions (744 or 644). Refer to chmod on page 626 for more information about symbolic permissions. Examples The following commands set the file-creation permissions mask and display the mask and its effect when you create a file and a directory. The mask of 022, when subtracted from 777, gives permissions of 644 (rw–r––r––) for a file and 755 (rwxr–xr–x) for a directory: umask 871 $ umask 022 $ umask 0022 $ touch afile $ mkdir adirectory $ ls -ld afile adirectory drwxr-xr-x 2 max max 4096 Jul 24 11:25 adirectory -rw-r--r-- 1 max max 0 Jul 24 11:25 afile The next example sets the same mask using symbolic values. The –S option displays the mask symbolically: $ umask u=rwx,g=rx,o=rx $ umask 0022 $ umask -S u=rwx,g=rx,o=rx 872 uniq uniq Displays unique lines uniq uniq [options] [input-file] [output-file] The uniq utility displays its input, removing all but one copy of successive repeated lines. If the file has been sorted (see sort on page 817), uniq ensures that no two lines it displays are the same. Arguments When you do not specify the input-file, uniq reads from standard input. When you do not specify the output-file, uniq writes to standard output. Options Under Linux, uniq accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. A field is a sequence of characters bounded by SPACEs, TAB s, NEWLINEs, or a combination of these characters. ––count ––repeated –c Precedes each line with the number of occurrences of the line in the input file. –d Displays one copy of lines that are repeated; does not display lines that are not repeated. ––skip-fields=nfield –f nfield Ignores the first nfield blank-separated fields of each line. The uniq utility bases its comparison on the remainder of the line, including the leading blanks of the next field on the line (see the –s [––skip-chars] option). ––ignore-case –i Ignores case when comparing lines. ––skip-chars=nchar –s nchar Ignores the first nchar characters of each line. If you also use the –f (––skip-fields) option, uniq ignores the first nfield fields followed by nchar characters. You can use this option to skip over leading blanks of a field. ––unique –u Displays only lines that are not repeated. ––check-chars=nchar –w nchar Compares up to nchars characters on a line after honoring the –f (––skip-fields) and –s (––skip-chars) options. By default uniq compares the entire line. L Examples These examples assume the file named test in the working directory contains the following text: uniq 873 $ cat test boy took bat home boy took bat home girl took bat home dog brought hat home dog brought hat home dog brought hat home Without any options, uniq displays only one copy of successive repeated lines: $ uniq test boy took bat home girl took bat home dog brought hat home The –c (––count) option displays the number of consecutive occurrences of each line in the file: $ uniq -c test 2 boy took bat home 1 girl took bat home 3 dog brought hat home The –d (––repeated) option displays only lines that are consecutively repeated in the file: $ uniq -d test boy took bat home dog brought hat home The –u (––unique) option displays only lines that are not consecutively repeated in the file: $ uniq -u test girl took bat home In the next example, the –f (––skip-fields) option skips the first field in each line, causing the lines that begin with boy and the one that begins with girl to appear to be consecutive repeated lines. The uniq utility displays only one occurrence of these lines: $ uniq -f 1 test boy took bat home dog brought hat home The next example uses both the –f (––skip-fields) and –s (––skip-chars) arguments first to skip two fields and then to skip two characters. The two characters this command skips include the SPACE that separates the second and third fields and the first character of the third field. Ignoring these characters, all the lines appear to be consecutive repeated lines containing the string at home. The uniq utility displays only the first of these lines: $ uniq -f 2 -s 2 test boy took bat home 874 w w Displays information about system users w w [options] [username] The w utility displays the names of users who are currently logged in, together with their terminal device numbers, the times they logged in, the commands they are running, and other information. Arguments Options The username restricts the display to information about that user. –f (from) Removes the FROM column. For users who are directly connected, this field contains a hyphen. L –h (no header) Suppresses the header line. –s (short) Displays less information: username, terminal device, idle time, and command. L Discussion The first line w displays is the same as that displayed by uptime. This line includes the time of day, how long the system has been running (in days, hours, and minutes), how many users are logged in, and how busy the system is (load average). From left to right, the load averages indicate the number of processes that have been waiting to run in the past 1 minute, 5 minutes, and 15 minutes. The columns of information that w displays have the following headings: USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT The USER is the username of the user. The TTY is the device name for the line that the user logged in on. The FROM is the system name that a remote user is logged in from; it is a hyphen for a local user. The LOGIN@ gives the date and time when the user logged in. The IDLE indicates how many minutes have elapsed since the user last used the keyboard. The JCPU is the CPU time used by all processes attached to the user's tty, not including completed background jobs. The PCPU is the time used by the process named in the WHAT column. The WHAT is the command that user is running. Examples The first example shows the full list produced by the w utility: $ w 10:26am USER max max root sam hls up 1 day, 55 min, TTY FROM tty1 tty2 pts/1 pts/2 pts/3 potato 6 users, load average: 0.15, 0.03, LOGIN@ IDLE JCPU PCPU Fri 9am 20:39m 0.22s 0.01s Fri 5pm 17:16m 0.07s 0.07s Fri 4pm 14:28m 0.20s 0.07s Fri 5pm 3:23 0.08s 0.08s 10:07am 0.00s 0.08s 0.02s 0.01 WHAT vim td -bash -bash /bin/bash w w 875 In the next example, the –s option produces an abbreviated listing: $ w -s 10:30am USER max max root sam hls up 1 day, 58 min, TTY FROM tty1 tty2 pts/1 pts/2 pts/3 potato 6 users, load average: 0.15, 0.03, 0.01 IDLE WHAT 20:43m vim td 17:19m -bash 14:31m -bash 0.20s vim memo.030125 0.00s w -s The final example requests information only about Max: $ w max 10:35am up 1 day, 1:04, USER TTY FROM max tty1 max tty2 - 6 users, load average: 0.06, 0.01, 0.00 LOGIN@ IDLE JCPU PCPU WHAT Fri 9am 20:48m 0.22s 0.01s vim td Fri 5pm 17:25m 0.07s 0.07s -bash 876 wc wc Displays the number of lines, words, and bytes wc wc [options] [file-list] The wc utility displays the number of lines, words, and bytes in its input. When you specify more than one file on the command line, wc displays totals for each file as well as totals for all files. Arguments The file-list is a list of the pathnames of one or more files that wc analyzes. When you omit file-list, wc takes its input from standard input. Options Under Linux, wc accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––bytes –c Displays only the number of bytes in the input. ––max-line-length –L Displays the length of the longest line in the input. L ––lines –l (lowercase "l") Displays only the number of lines (that is, NEWLINE characters) in the input. ––chars –m Displays only the number of characters in the input. ––words –w Displays only the number of words in the input. Notes A word is a sequence of characters bounded by SPACE s, TAB s, NEWLINE s, or a combination of these characters. Examples The following command analyzes the file named memo. The numbers in the output represent the number of lines, words, and bytes in the file. $ wc memo 5 31 146 memo The next command displays the number of lines and words in three files. The line at the bottom, with the word total in the right column, contains the sum of each column. $ wc -lw memo1 memo2 memo3 10 62 memo1 12 74 memo2 12 68 memo3 34 204 total which 877 Shows where in PATH a command is located which command-list For each command in command-list, the which utility searches the directories in the PATH variable (page 297) and displays the absolute pathname of the first file it finds whose simple filename is the same as the command. Arguments The command-list is a list of one or more commands (utilities) that which searches for. For each command, which searches the directories listed in the PATH environment variable, in order, and displays the full pathname of the first command (executable file) it finds. If which does not locate a command, it displays a message. Options Options preceded by a double hyphen (––) work under Linux only; not all of these options are available under all Linux distributions. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––all –a Displays all matching executable files in PATH, not just the first. ––read-alias –i Reads aliases from standard input and reports on matching aliases in addition to executable files in PATH (turn off with ––skip-alias). L ––read-functions Reads shell functions from standard input and reports on matching functions in addition to executable files in PATH (turn off with ––skip-functions). L ––show-dot Displays ./ in place of the absolute pathname when a directory in PATH starts with a period and a matching executable file is in that directory (turn off with ––skip-dot). L ––show-tilde Displays a tilde (~) in place of the absolute pathname of the user's home directory where appropriate. This option is ignored when a user working with root privileges runs which. L ––tty-only Does not process more options (to the right of this option) if the process running which is not attached to a terminal. L Notes Some distributions define an alias for which such as the following: $ alias which alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde' If which is not behaving as you would expect, verify that you are not running an alias. The preceding alias causes which to be effective only when it is run interactively (––tty-only) and to display aliases, display the working directory as a period when appropriate, and display the name of the user's home directory as a tilde. which which 878 which The TC Shell includes a which builtin (see the tcsh man page) that works slightly differently from the which utility (/usr/bin/which). Without any options, the which utility does not locate aliases, functions, and shell builtins because these do not appear in PATH. In contrast, the tcsh which builtin locates aliases, functions, and shell builtins. Examples The first example quotes the first letter of the command (\which) to prevent the shell from invoking the alias (page 326) for which: $ \which vim dir which /usr/bin/vim /bin/dir /usr/bin/which The next example, which works on some Linux systems only, is the same as the first but uses the alias for which (which it displays): $ which vim dir which alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde' /usr/bin/which /usr/bin/vim /usr/bin/dir The final example is the same as the previous one except it is run from tcsh. The tcsh which builtin is used instead of the which utility: tcsh $ which vim dir which /usr/bin/vim /bin/dir which: shell built-in command. who 879 who who Displays information about logged-in users who [options] who am i The who utility displays information about users who are logged in on the local system. This information includes each user's username, terminal device, login time, and, if applicable, the corresponding remote hostname. Arguments When given two arguments (traditionally, am i), who displays information about the user giving the command. If applicable, the username is preceded by the hostname of the system running who (as in kudos!max). Options Under Linux, who accepts the common options described on page 603. Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. ––all ––boot –a Displays a lot of information. L –b Displays the date and time when the system was last booted. L ––heading –H Displays a header. ––login –l (lowercase "l") Lists devices waiting for a user to log in. L ––count –q (quick) Lists the usernames only, followed by the number of users logged in on the system. ––mesg –T Appends after each user's username a character that shows whether that user has messages enabled. A plus (+) means that messages are enabled, a hyphen (–) means that they are disabled, and a question mark (?) indicates that who cannot find the device. If messages are enabled, you can use write to communicate with the user. Refer to "mesg: Denies or Accepts Messages" on page 71. ––users –u Includes each user's idle time in the display. If the user has typed on her terminal in the past minute, a period (.) appears in this field. If no input has occurred for more than a day, the word old appears. In addition, this option includes the PID number and comment fields. See the "Discussion" section. Discussion The line who displays has the following syntax: user [messages] line login-time [idle] [PID] comment 880 who The user is the username of the user. The messages argument indicates whether messages are enabled or disabled (see the –T [––mesg] option). The line is the device name associated with the line the user is logged in on. The login-time is the date and time when the user logged in. The idle argument is the length of time since the terminal was last used (the idle time; see the –u [––users] option). The PID is the process identification number. The comment is the name of the remote system that the user is logged in from (it is blank for local users). Notes The finger utility (pages 68 and 695) provides information similar to that given by who. Examples The following examples demonstrate the use of the who utility: $ who hls sam max tty1 tty2 ttyp3 $ who am i bravo!max Jul 30 06:01 Jul 30 06:02 Jul 30 14:56 (bravo) ttyp3 $ who -HTu USER LINE hls - tty1 sam + tty2 max + ttyp3 Jul 30 14:56 (bravo) TIME IDLE Jul 30 06:01 03:53 Jul 30 06:02 14:47 Jul 30 14:56 . PID COMMENT 1821 2235 14777 (bravo) xargs 881 Converts standard input to command lines xargs [options] [command] The xargs utility is a convenient, efficient way to convert standard output of one command into arguments for another command. This utility reads from standard input, keeps track of the maximum allowable length of a command line, and avoids exceeding that limit by repeating command as needed. Finally xargs executes the constructed command line(s). Arguments The command is the command line you want xargs to use as a base for the command it constructs. If you omit command, it defaults to echo. The xargs utility appends to command the arguments it receives from standard input. If any arguments should precede the arguments from standard input, you must include them as part of command. Options Options preceded by a double hyphen (––) work under Linux only. Except as noted, options named with a single letter and preceded by a single hyphen work under Linux and OS X. –I [marker] (replace) Allows you to place arguments from standard input anywhere within command. All occurrences of marker in command for xargs are replaced by the arguments generated from standard input of xargs. With this option, xargs executes command for each input line. The –l (––max-lines) option is ignored when you use this option. ––max-lines[=num] –l [num] (lowercase "l") Executes command once for every num lines of input (num defaults to 1). L ––max-args=num –n num Executes command once for every num arguments in the input line. ––max-procs=num –P num Allows xargs to run up to maxprocs instances of command simultaneously. (The default is 1, which runs commands sequentially.) This option may improve the throughput if xargs is running on a multiprocessor system. ––interactive –p Prompts prior to each execution of command. ––no-run-if-empty –r Causes xargs not to execute command if standard input is empty. Ordinarily xargs executes command at least once, even if standard input includes only blanks. L xargs xargs 882 xargs Discussion The xargs utility reads arguments to command from standard input, interpreting each whitespace-delimited string as a separate argument. It then constructs a command line from command and a series of arguments. When the maximum command line length would be exceeded by adding another argument, xargs runs the command line it has built. If there is more input, xargs repeats the process of building a command line and running it. This process continues until all input has been read. Notes The xargs utility is often used as an efficient alternative to the –exec option of find (page 689). If you call find with the –exec option to run a command, it runs the command once for each file it processes. Each execution of the command creates a new process, which can drain system resources when you are processing many files. By accumulating as many arguments as possible, xargs can greatly reduce the number of processes needed. The first example in the "Examples" section shows how to use xargs with find. Examples To locate and remove all files with names ending in .o from the working directory and its subdirectories, you can use the find –exec option: $ find . -name *.o -exec rm --force {} \; This approach calls the rm utility once for each .o file that find locates. Each invocation of rm requires a new process. If a lot of .o files exist, the system must spend a significant amount of time creating, starting, and then cleaning up these processes. You can reduce the number of processes by allowing xargs to accumulate as many filenames as possible before calling rm: $ find . -name *.o -print | xargs rm --force In the next example, the contents of all *.txt files located by find are searched for lines containing the word login. All filenames that contain login are displayed by grep. $ find . -name *.txt -print | xargs grep -w -l login The next example shows how to use the –I option to cause xargs to embed standard input within command instead of appending it to command. This option also causes command to be executed each time a NEWLINE character is encountered in standard input; –l (––max-lines) does not override this behavior. $ cat names Tom, Dick, and Harry $ xargs echo "Hello," < names Hello, Tom, Dick, and Harry xargs 883 $ xargs -I xxx echo "Hello xxx. Join me for lunch?" < names Hello Tom,. Join me for lunch? Hello Dick,. Join me for lunch? Hello and Harry. Join me for lunch? The final example uses the same input file as the previous examples as well as the –n (––max-args) and –l (––max-lines) options: $ xargs -n 1 echo "Hi there" < names Hi there Tom, Hi there Dick, Hi there and Hi there Harry $ xargs -l 2 echo "Hi there" < names Hi there Tom, Dick, Hi there and Harry See page 833 for another example of the use of xargs. This page intentionally left blank I PART VI Appendixes APPENDIX A Regular Expressions APPENDIX B Help 887 897 APPENDIX C Keeping the System Up-to-Date APPENDIX D Mac OS X Notes 909 925 885 This page intentionally left blank A Regular Expressions In This Appendix Characters . . . . 888 Delimiters . . . . 888 Simple Strings. . . . . 888 Special Characters . . . . 888 Rules . . . . 891 Bracketing Expressions . . . . 892 The Replacement String . . . . 892 Extended Regular Expressions . . . 893 A regular expression defines a set of one or more strings of characters. A simple string of characters is a regular expression that defines one string of characters: itself. A more complex regular expression uses letters, numbers, and special characters to define many different strings of characters. A regular expression is said to match any string it defines. AAppendixA This appendix describes the regular expressions used by ed, vim, emacs, grep, awk/mawk/gawk, sed, Perl, and many other utilities. Refer to page 517 for more information on Perl regular expressions. The regular expressions used in shell ambiguous file references are different and are described in "Filename Generation/Pathname Expansion" on page 136. 887 888 Appendix A Regular Expressions Characters As used in this appendix, a character is any character except a NEWLINE. Most characters represent themselves within a regular expression. A special character, also called a metacharacter, is one that does not represent itself. If you need to use a special character to represent itself, you must quote it as explained on page 891. Delimiters A character called a delimiter usually marks the beginning and end of a regular expression. The delimiter is always a special character for the regular expression it delimits (that is, it does not represent itself but marks the beginning and end of the expression). Although vim permits the use of other characters as a delimiter and grep does not use delimiters at all, the regular expressions in this appendix use a forward slash (/) as a delimiter. In some unambiguous cases, the second delimiter is not required. For example, you can sometimes omit the second delimiter when it would be followed immediately by RETURN. Simple Strings The most basic regular expression is a simple string that contains no special characters except the delimiters. A simple string matches only itself (Table A-1). In the examples in this appendix, the strings that are matched are underlined and look like this. Table A-1 Simple strings Regular expression Matches Examples /ring/ ring ring, spring, ringing, stringing /Thursday/ Thursday Thursday, Thursday's /or not/ or not or not, poor nothing Special Characters You can use special characters within a regular expression to cause the regular expression to match more than one string. A regular expression that includes a Special Characters 889 special character always matches the longest possible string, starting as far toward the beginning (left) of the line as possible. Periods A period (.) matches any character (Table A-2). Table A-2 Regular expression Periods Matches Examples / .alk/ All strings consisting of a SPACE followed by any character followed by alk will talk, may balk /.ing/ All strings consisting of any character preceding ing sing song, ping, before inglenook Brackets Brackets ([]) define a character class1 that matches any single character within the brackets (Table A-3). If the first character following the left bracket is a caret (^), the brackets define a character class that matches any single character not within the brackets. You can use a hyphen to indicate a range of characters. Within a character-class definition, backslashes and asterisks (described in the following sections) lose their special meanings. A right bracket (appearing as a member of the character class) can appear only as the first character following the left bracket. A caret is special only if it is the first character following the left bracket. A dollar sign is special only if it is followed immediately by the right bracket. Table A-3 Regular expression Brackets Matches Examples /[bB]ill/ Member of the character class b and B followed by ill bill, Bill, billed /t[aeiou].k/ t followed by a lowercase vowel, any character, and a k talkative, stink, teak, tanker /# [6–9]/ # followed by a SPACE and a member of the character class 6 through 9 # 60, # 8:, get # 9 /[^a–zA–Z]/ Any character that is not a letter (ASCII character set only) 1, 7, @, ., }, Stop! 1. GNU documentation calls these List Operators and defines Character Class operators as expressions that match a predefined group of characters, such as all numbers (see Table V-32 on page 864). 890 Appendix A Regular Expressions Asterisks An asterisk can follow a regular expression that represents a single character (Table A-4). The asterisk represents zero or more occurrences of a match of the regular expression. An asterisk following a period matches any string of characters. (A period matches any character, and an asterisk matches zero or more occurrences of the preceding regular expression.) A character-class definition followed by an asterisk matches any string of characters that are members of the character class. Table A-4 Regular expression Asterisks Matches Examples /ab*c/ a followed by zero or more b's followed by ac ac, abc, abbc, debbcaabbbc /ab.*c/ ab followed by zero or more characters followed by c abc, abxc, ab45c, xab 756.345 x cat /t.*ing/ t followed by zero or more characters followed by ing thing, ting, I thought of going A string composed only of letters and SPACEs 1. any string without numbers or punctuation! /(.*)/ As long a string as possible between (and) Get (this) and (that); /([^)]*)/ The shortest string possible that starts with (and ends with) (this), Get (this and that) /[a–zA–Z]*/ Carets and Dollar Signs A regular expression that begins with a caret (^) can match a string only at the beginning of a line. In a similar manner, a dollar sign ($) at the end of a regular expression matches the end of a line. The caret and dollar sign are called anchors because they force (anchor) a match to the beginning or end of a line (Table A-5). Table A-5 Carets and dollar signs Regular expression Matches Examples /^T/ A T at the beginning of a line This line..., That Time..., In Time /^+[0–9]/ A plus sign followed by a digit at the beginning of a line +5 +45.72, +759 Keep this... /:$/ A colon that ends a line ...below: Rules 891 Quoting Special Characters You can quote any special character (but not parentheses [except in Perl; page 521] or a digit) by preceding it with a backslash (Table A-6). Quoting a special character makes it represent itself. Table A-6 Quoted special characters Regular expression Matches Examples /end\./ All strings that contain end followed by a period The end., send., pretend.mail / \\/ A single backslash \ / */ An asterisk *.c, an asterisk (*) / \[5\]/ [5] it was five [5] /and\/or/ and/or and/or Rules The following rules govern the application of regular expressions. Longest Match Possible A regular expression always matches the longest possible string, starting as far toward the beginning of the line as possible. Perl calls this type of mach a greedy match (page 520). For example, given the string This (rug) is not what it once was (a long time ago), is it? the expression /Th.*is/ matches This (rug) is not what it once was (a long time ago), is and /(.*)/ matches (rug) is not what it once was (a long time ago) However, /([^)]*)/ matches (rug) Given the string singing songs, singing more and more the expression /s.*ing/ matches singing songs, singing and /s.*ing song/ matches singing song 892 Appendix A Regular Expressions Empty Regular Expressions Within some utilities, such as vim and less (but not grep), an empty regular expression represents the last regular expression you used. For example, suppose you give vim the following Substitute command: :s/mike/robert/ If you then want to make the same substitution again, you can use the following command: :s//robert/ Alternatively, you can use the following commands to search for the string mike and then make the substitution /mike/ :s//robert/ The empty regular expression (//) represents the last regular expression you used (/mike/). Bracketing Expressions You can use quoted parentheses, \(and \), to bracket a regular expression. (However, Perl uses unquoted parentheses to bracket regular expressions; page 521.) The string that the bracketed regular expression matches can be recalled, as explained in "Quoted Digit." A regular expression does not attempt to match quoted parentheses. Thus a regular expression enclosed within quoted parentheses matches what the same regular expression without the parentheses would match. The expression /\(rexp\)/ matches what /rexp/ would match; /a\(b*\)c/ matches what /ab*c/ would match. You can nest quoted parentheses. The bracketed expressions are identified only by the opening \(, so no ambiguity arises in identifying them. The expression /\([a–z]\([A–Z]*\)x\)/ consists of two bracketed expressions, one nested within the other. In the string 3 t dMNORx7 l u, the preceding regular expression matches dMNORx, with the first bracketed expression matching dMNORx and the second matching MNOR. The Replacement String The vim and sed editors use regular expressions as search strings within Substitute commands. You can use the ampersand (&) and quoted digits () special characters to represent the matched strings within the corresponding replacement string. Extended Regular Expressions 893 Ampersand Within a replacement string, an ampersand (&) takes on the value of the string that the search string (regular expression) matched. For example, the following vim Substitute command surrounds a string of one or more digits with NN. The ampersand in the replacement string matches whatever string of digits the regular expression (search string) matched: :s/[0-9][0-9]*/NN&NN/ Two character-class definitions are required because the regular expression [0–9]* matches zero or more occurrences of a digit, and any character string constitutes zero or more occurrences of a digit. Quoted Digit Within the search string, a bracketed regular expression, \(xxx\) [(xxx) in Perl], matches what the regular expression would have matched without the quoted parentheses, xxx. Within the replacement string, a quoted digit, , represents the string that the bracketed regular expression (portion of the search string) beginning with the nth \(matched. (Perl accepts a quoted digit for this purpose, but Perl's preferred style is to precede the digit with a dollar sign [$n; page 521]). For example, you can take a list of people in the form last-name, first-name initial and put it in the form first-name initial last-name with the following vim command: :1,$s/\([^,]*\), \(.*\)/\2 \1/ This command addresses all lines in the file (1,$). The Substitute command (s) uses a search string and a replacement string delimited by forward slashes. The first bracketed regular expression within the search string, \([^,]*\), matches what the same unbracketed regular expression, [^,]*, would match: zero or more characters not containing a comma (the last-name). Following the first bracketed regular expression are a comma and a SPACE that match themselves. The second bracketed expression, \(.*\), matches any string of characters (the first-name and initial). The replacement string consists of what the second bracketed regular expression matched (\2), followed by a SPACE and what the first bracketed regular expression matched (\1). Extended Regular Expressions This section covers patterns that use an extended set of special characters. These patterns are called full regular expressions or extended regular expressions. In addition 894 Appendix A Regular Expressions to ordinary regular expressions, Perl and vim provide extended regular expressions. The three utilities egrep, grep when run with the –E option (similar to egrep), and mawk/gawk provide all the special characters included in ordinary regular expressions, except for \(and \), as well those included in extended regular expressions. Two of the additional special characters are the plus sign (+) and the question mark (?). They are similar to *, which matches zero or more occurrences of the previous character. The plus sign matches one or more occurrences of the previous character, whereas the question mark matches zero or one occurrence. You can use any one of the special characters *, +, and ? following parentheses, causing the special character to apply to the string surrounded by the parentheses. Unlike the parentheses in bracketed regular expressions, these parentheses are not quoted (Table A-7). Table A-7 Extended regular expressions Regular expression Matches Examples /ab+c/ a followed by one or more b's followed by ac yabcw, abbc57 /ab?c/ a followed by zero or one b followed by c back, abcdef /(ab)+c/ One or more occurrences of the string ab followed by c zabcd, ababc! /(ab)?c/ Zero or one occurrence of the string ab followed by c xc, abcc In full regular expressions, the vertical bar (|) special character is a Boolean OR operator. Within vim, you must quote the vertical bar by preceding it with a backslash to make it special (\|). A vertical bar between two regular expressions causes a match with strings that match the first expression, the second expression, or both. You can use the vertical bar with parentheses to separate from the rest of the regular expression the two expressions that are being ORed (Table A-8). Table A-8 Full regular expressions Regular expression Meaning Examples /ab|ac/ Either ab or ac ab, ac, abac (abac is two matches of the regular expression) /^Exit|^Quit/ Lines that begin with Exit or Quit Exit, Quit, No Exit /(D|N)\. Jones/ D. Jones or N. Jones P.D. Jones, N. Jones Appendix Summary 895 Appendix Summary A regular expression defines a set of one or more strings of characters. A regular expression is said to match any string it defines. In a regular expression, a special character is one that does not represent itself. Table A-9 lists special characters. Table A-9 Special characters Character Meaning . Matches any single character * Matches zero or more occurrences of a match of the preceding character ^ Forces a match to the beginning of a line $ A match to the end of a line \ Quotes special characters \ Forces a match to the end of a word Table A-10 lists ways of representing character classes and bracketed regular expressions. Table A-10 Character classes and bracketed regular expressions Class Defines [xyz] Defines a character class that matches x, y, or z [^xyz] Defines a character class that matches any character except x, y, or z [x–z] Defines a character class that matches any character x through z inclusive \(xyz \) Matches what xyz matches (a bracketed regular expression; not Perl) (xyz) Matches what xyz matches (a bracketed regular expression; Perl only) In addition to the preceding special characters and strings (excluding quoted parentheses, except in vim), the characters in Table A-11 are special within full, or extended, regular expressions. Table A-11 Extended regular expressions Expression Matches + Matches one or more occurrences of the preceding character ? Matches zero or one occurrence of the preceding character 896 Appendix A Regular Expressions Table A-11 Extended regular expressions (continued) Expression Matches (xyz)+ Matches one or more occurrences of what xyz matches (xyz)? Matches zero or one occurrence of what xyz matches (xyz)* Matches zero or more occurrences of what xyz matches xyz|abc Matches either what xyz or what abc matches (use \| in vim) (xy|ab)c Matches either what xyc or what abc matches (use \| in vim) Table A-12 lists characters that are special within a replacement string in sed, vim, and Perl. Table A-12 Replacement strings String Represents & Represents what the regular expression (search string) matched A quoted number, n, represents what the nth bracketed regular expression in the search string matched $n A number preceded by a dollar sign, n, represents what the nth bracketed regular expression in the search string matched (Perl only) B Help In This Appendix Solving a Problem . . . . 898 The Apple Web Site. . . . . 899 Finding Linux and OS X–Related Information . . . . 899 Documentation . . . . 900 Useful Linux and OS X Sites . . . . . 901 Linux and OS X Newsgroups . . . . 902 You need not be a user or system administrator in isolation. A large community of Linux and OS X experts is willing to assist you in learning about, helping you solve problems with, and getting the most out of a Linux or OS X system. Before you ask for help, however, make sure you have done everything you can to solve the problem yourself. No doubt, someone has experienced the same problem before you and the answer to your question exists somewhere on the Internet. Your job is to find it. This appendix lists resources and describes methods that can help you in that task. BAppendixB Mailing Lists . . . . 903 Words. . . . . 903 Software . . . . 904 Office Suites and Word Processors . . . . 906 Specifying a Terminal . . . . 906 897 898 Appendix B Help Solving a Problem Following is a list of steps that can help you solve a problem without asking someone for help. Depending on your understanding of and experience with the hardware and software involved, these steps may lead to a solution. 1. Linux and OS X come with extensive documentation. Read the documentation on the specific hardware or software you are having a problem with. If it is a GNU product, use info; otherwise, use man to find local information. Also look in /usr/share/doc for documentation on specific tools. For more information refer to "Where to Find Documentation" on page 33. 2. When the problem involves some type of error or other message, use a search engine, such as Google (www.google.com) or Google Groups (groups.google.com), to look up the message on the Web. If the message is long, pick a unique part of the message to search for; 10 to 20 characters should be enough. Enclose the search string within double quotation marks. See "Using the Internet to Get Help" on page 39 for an example of this kind of search. 3. Check whether the Linux Documentation Project (www.tldp.org) has a HOWTO or mini-HOWTO on the subject in question. Search its site for keywords that relate directly to the product and problem. Read the FAQs. 4. See Table B-1 on page 900 for other sources of documentation. 5. Use Google or Google Groups to search on keywords that relate directly to the product and problem. 6. When all else fails (or perhaps before you try anything else), examine the system logs in /var/log. First look at the end of the messages (Linux) or system.log (OS X) file using one of the following commands: # tail -20 /var/log/messages # tail -20 /var/log/system.log If messages or system.log contain nothing useful, run the following command. It displays the names of the log files in chronological order, with the most recently modified files appearing at the bottom of the list: $ ls -ltr /var/log Look at the files at the bottom of the list first. If the problem involves a network connection, review the secure or auth.log file (some systems use a different name) on the local and remote systems. Also look at messages or system.log on the remote system. Finding Linux and OS X–Related Information 899 7. The /var/spool directory contains subdirectories with useful information: cups holds the print queues; postfix, mail, or exim4 holds the user's mail files. Other subdirectories of /var contain similarly useful files. If you are unable to solve a problem yourself, a thoughtful question to an appropriate newsgroup (page 902) or mailing list (page 903) can often elicit useful information. When you send or post a question, make sure you describe the problem and identify the local system carefully. Include the version numbers of the operating system and any software packages that relate to the problem. Describe the hardware, if appropriate. There is an etiquette to posting questions—see www.catb.org/~esr/faqs/smart-questions.html for a good paper by Eric S. Raymond and Rick Moen titled "How To Ask Questions the Smart Way." The author's home page (www.sobell.com) contains corrections to this book, answers to selected chapter exercises, and pointers to other Linux and OS X sites. The Apple Web Site The Apple Web site is a rich source of Mac OS X information. Following is a list of some locations that may be of interest: • Manuals for Apple operating systems and products are available at www.info.apple.com/support/manuals.html. • General support material is available at www.apple.com/support. • Apple support forums are online discussions about Apple-related issues. Forums are dedicated to specific applications or OS versions, specific Macintosh models, and so on. Go to discussions.info.apple.com to browse through the forums. • Mailing lists are hosted by Apple; see lists.apple.com for a list of mailing lists. • Even if you are not using Mac OS X Server, the documentation for Server has a lot of material on OS X command-line usage. Finding Linux and OS X–Related Information Linux and OS X come with reference pages stored online. You can read these documents by using the man (page 33) or info (page 36) utility. Under Mac OS X, you can use the documentation browser in Xcode or the Mac Help application. You can read man and info pages to get more information about specific topics while reading this book or to determine which features are available with Linux or OS X. To search for topics, use the apropos utility (see page 35 or give the command man apropos). The Apple support site (www.apple.com/support) has many useful Mac OS X links. 900 Appendix B Help Documentation Good books are available on various aspects of using and managing UNIX systems in general and Linux and OS X systems in particular. In addition, you may find the sites listed in Table B-1 useful.1 Table B-1 Documentation Site About the site URL Apple Manuals Apple manuals (look under Software Manuals/Mac OS X in a box on the right) www.apple.com/support/manuals freedesktop.org Creates standards for interoperability between open-source desktop environments freedesktop.org GNOME GNOME home page www.gnome.org GNU Manuals GNU manuals www.gnu.org/manual info Instructions for using the info utility www.gnu.org/software/texinfo/manual/info Internet FAQ Archives Searchable FAQ archives www.faqs.org KDE Documentation KDE documentation www.kde.org/documentation KDE News KDE news dot.kde.org Linux Documentation Project All things related to Linux documentation (in many languages): HOWTOs, guides, FAQs, man pages, and magazines. This site is the best overall source for Linux documentation. Make sure to visit the Links page. www.tldp.org RFCs Requests for comments; see RFC (page 975) www.rfc-editor.org System Administrators Guild (SAGE) SAGE is a group for system administrators www.sage.org 1. The right-hand columns of most of the tables in this appendix show Internet addresses (URLs). All sites have an implicit http:// prefix unless ftp:// or https:// is shown. Refer to "URLs (Web addresses)" on page 25. Finding Linux and OS X–Related Information 901 Useful Linux and OS X Sites Sometimes the sites listed in Table B-2 are so busy that you cannot connect to them. In this case, you are usually given a list of alternative, or mirror, sites to try. Table B-2 Useful Linux and OS X sites Site About the site URL Accelerate Your Mac Covers improving system performance mostly through hardware upgrades xlr8yourmac.com Apple Developer Connection A site for OS X developers developer.apple.com developer.apple.com/opensource Command-Line Fu The place to record those commandline gems you return to again and again www.commandlinefu.com DistroWatch A survey of Linux distributions, including news, reviews, and articles distrowatch.com Fedora Forums A forum about everything Fedora fedoraforum.org Gentoo Forums A forum about everything Gentoo forums.gentoo.org GNU GNU Project home page www.gnu.org ibiblio A large library and digital archive. Formerly Metalab; formerly Sunsite. www.ibiblio.org www.ibiblio.org/pub/linux www.ibiblio.org/pub/historic-linux Linux Standard Base (LSB) A group dedicated to standardizing Linux www.linuxfoundation.org/collaborate/workgroups/lsb MacFixit A user site full of tips, trivia, and troubleshooting help for Macintosh systems www.macfixit.com Mac Forums Forums about everything Mac, official and unofficial discussions.apple.com macosx.com/forums Mac Hints Mac tips www.macosxhints.com MacLife A magazine with reviews, tips, and articles about Macintosh systems www.maclife.com Mac User Groups Apple's registry of Macintosh user groups; includes a searchable database www.apple.com/usergroups www.apple.com/usergroups/find 902 Appendix B Help Table B-2 Useful Linux and OS X sites (continued) Site About the site URL MacWorld A magazine dedicated to the Mac www.macworld.com Notes from the Field A blog by Charles Edge with coverage of Mac innards and tips for enhancing the Mac experience krypted.com OS X VersionTracker A searchable site covering new releases of OS X software packages www.versiontracker.com/macosx Psychocats Tutorials and random pages for Ubuntu users www.psychocats.net/ubuntu System Administrators Guild (SAGE) SAGE is a group for system administrators www.sage.org Sobell The author's home page contains useful links, errata for this book, code for many of the examples in this book, and answers to selected exercises www.sobell.com Threads Linux Documentation Project threads FAQ tldp.org/FAQ/Threads-FAQ Ubuntu Forums A forum about everything Ubuntu; in some cases discusses other distributions ubuntuforums.org USENIX A large, well-established UNIX group. This site has many links, including a list of conferences. www.usenix.org X.Org The X Window System home www.x.org Linux and OS X Newsgroups One of the best ways of getting specific information is through a newsgroup. You can often find the answer to a question by reading postings to the newsgroup. Try using Google Groups (groups.google.com) to search through newsgroups to see whether the question has already been asked and answered. Alternatively, open a newsreader program and subscribe to appropriate newsgroups. If necessary, you can post a question for someone to answer. Before you do so, make sure you are posting to the correct group and that your question has not already been answered. The comp.os.linux.answers (Linux) and comp.sys.mac.system (OS X) newsgroups provide postings of solutions to common problems and periodic postings of the most up-to-date versions of FAQ and HOWTO documents. The comp.os.linux.misc newsgroup has answers to miscellaneous Linux-related questions. Apple Discussions (discussions.apple.com) has a list of online forums, including a section on Mac OS X and Related Software. Finding Linux and OS X–Related Information 903 Mailing Lists Subscribing to a mailing list allows you to participate in an electronic discussion. With most lists, you can send and receive email dedicated to a specific topic to and from a group of users. Moderated lists do not tend to stray as much as unmoderated lists, assuming the list has a good moderator. The disadvantage of a moderated list is that some discussions may be cut off when they get interesting if the moderator deems that the discussion has gone on for too long. Mailing lists described as bulletins are unidirectional: You cannot post information to these lists but can only receive periodic bulletins. If you have the subscription address for a mailing list but are not sure how to subscribe, put the word help in the body and/or header of email you send to the address. You will usually receive instructions via return email. You can also use a search engine to search for mailing list linux or mailing list os x. You can find Red Hat and Fedora mailing lists at www.redhat.com/mailman/listinfo, Ubuntu mailing lists at www.ubuntu.com/support/community/mailinglists, and Debian mailing lists at www.debian.org/MailingLists/subscribe. Apple supports many mailing lists. Visit www.lists.apple.com/mailman/listinfo to display a list of Mac OS X mailing lists; click on the name of a list to display subscription information. Words Many dictionaries, thesauruses, and glossaries are available online. Table B-3 lists a few of them. Table B-3 Looking up words Site About the site URL DICT.org Multiple-database search for words www.dict.org Dictionary.com Everything related to words dictionary.reference.com DNS Glossary DNS glossary www.menandmice.com/knowledgehub/dnsglossary FOLDOC The Free On-Line Dictionary of Computing foldoc.org GNOME Controls Defines many GUI controls

(widgets) developer.gnome.org/projects/gup/hig/2.0/controls.html The Jargon File An online version of The New Hacker's Dictionary www.catb.org/~esr/jargon Merriam-Webster English language www.merriam-webster.com OneLook Multiple-site word search with a single query www.onelook.com Roget's Thesaurus Thesaurus
humanities.uchicago.edu/forms_unrest/ROGET.html Webopedia Commercial technical dictionary www.webopedia.com 904 Appendix B Help Table B-3 Looking up words (continued) Site About the site URL Wikipedia An open-source (user-contributed) encyclopedia project wikipedia.org Wordsmyth Dictionary and thesaurus www.wordsmyth.net Yahoo
Reference Search multiple sources at the same time education.yahoo.com/reference Software There are many ways to learn about interesting software packages and their availability on the Internet. Table B-4 lists sites you can download software from. Another way to learn about software packages is through a newsgroup (page 902). Table B-4
Software Site About the site URL afp548 OS X essays and tools www.afp548.com BitTorrent BitTorrent efficiently distributes large amounts of static data azureus.sourceforge.net www.transmissionbt.com bombich.com Home of OS X Carbon Copy Center and other useful tools bombich.com CVL Concurrent Versions Librarian, a GUI for CVS under OS
X www.sente.ch/software/cvl CVS CVS (Concurrent Versions System) is a version control system www.nongnu.org/cvs ddd The ddd utility is a graphical frontend for command-line debuggers such as gdb www.gnu.org/software/ddd fink The fink utility builds and installs OS X software automatically www.finkproject.org Firefox Web browser
www.mozilla.com/firefox Free Software Directory Categorized, searchable lists of free software directory.fsf.org Free the Software! A Red Hat site that hosts many opensource projects sourceware.org Freshmeat A large index of UNIX and crossplatform software and themes freshmeat.net gcc Home page for the GNU C compiler gcc.gnu.org Finding
Linux and OS X–Related Information 905 Table B-4 Software (continued) Site About the site URL gdb The gdb utility is a command-line debugger www.gnu.org/software/gdb GNOME Project Links to all GNOME projects projects.gnome.org IceWALKERS Categorized, searchable lists of free software www.icewalkers.com kdbg The kdbg utility is a
graphical user interface to gdb freshmeat.net/projects/kdbg Linux Software Map A database of packages written for, ported to, or compiled for Linux www.boutell.com/lsm MacPorts Project A system for compiling, installing, and upgrading command-line, X11, or Aqua-based open-source software www.macports.org Mtools A collection of utilities to
access DOS floppy diskettes from Linux without mounting the diskettes (page 767) www.gnu.org/software/mtools Network Calculators Subnet mask calculator www.subnetmask.info NTFS driver Driver that enables Linux to read from and write to Windows NTFS filesystems (available in the ntfs-3g package) www.ntfs-3g.org Savannah Central point
for development, distribution, and maintenance of free software savannah.gnu.org SourceForge A development Web site with a large repository of open-source code and applications sourceforge.net splint C program verifier (replaces lint) www.splint.org strace The strace utility is a system call trace debugging tool sourceforge.net/projects/strace
Thunderbird Mail application www.mozillamessaging.com TkCVS A CVS GUI www.twobarleycorns.net/tkcvs.html ups The ups utility is a graphical sourcelevel debugger ups.sourceforge.net versiontracker A large index of OS X software www.versiontracker.com/macosx 906 Appendix B Help Office Suites and Word Processors Several office suites and
many word processors are available for Linux. Table B-5 lists a few of them. If you are exchanging documents with people using Windows, make sure the import from and export to MS Word functionality covers your needs. Table B-5 Office suites and word processors Product name What it does URL AbiWord Word processor www.abisource.com
KOffice Integrated suite of office applications, including the KWord word processing program www.koffice.org NeoOffice OpenOffice tailored for the Mac with native look and feel www.neooffice.org OpenOffice A multiplatform and multilingual office suite www.openoffice.org www.gnome.org/projects/ooo Specifying a Terminal Because vim, emacs,
and other textual and pseudographical programs take advantage of features specific to various kinds of terminals and terminal emulators, you must tell these programs the name of the terminal you are using or the terminal the terminal emulator is emulating. Most of the time the terminal name is set for you. If the terminal name is not specified or is
not specified correctly, the characters on the screen will be garbled or, when you start a program, the program will ask which type of terminal you are using. Terminal names describe the functional characteristics of a terminal or terminal emulator to programs that require this information. Although terminal names are referred to as either Terminfo
or Termcap names, the difference relates to the method each system uses to store the terminal characteristics internally—not to the manner in which you specify the name of a terminal. Terminal names that are often used with terminal emulators and with graphical monitors while they are run in textual mode include ansi, linux, vt100, vt102, vt220,
and xterm. When you are running a terminal emulator, you can specify the type of terminal you want to emulate. Set the emulator to either vt100 or vt220, and set TERM to the same value. When you log in, you may be prompted to identify the type of terminal you are using: TERM = (vt100) Specifying a Terminal 907 You can respond to this prompt
in one of two ways. First you can press RETURN to set your terminal type to the name in parentheses. Alternatively, if that name does not describe the terminal you are using, you can enter the correct name and then press RETURN. TERM = (vt100) ansi You may also receive the following prompt: TERM = (unknown) This prompt indicates that the
system does not know which type of terminal you are using. If you plan to run programs that require this information, enter the name of the terminal or terminal emulator you are using before you press RETURN. TERM If you do not receive a prompt, you can give the following command to display the value of the TERM variable and check whether
the terminal type has been set: $ echo $TERM If the system responds with the wrong name, a blank line, or an error message, set or change the terminal name. From the Bourne Again Shell (bash), enter a command similar to the following to set the TERM variable so the system knows which type of terminal you are using: export TERM=name
Replace name with the terminal name of the terminal you are using, making sure you do not put a SPACE before or after the equal sign. If you always use the same type of terminal, you can place this command in your ~/.bashrc file (page 271), causing the shell to set the terminal type each time you log in. For example, give the following command to
set the terminal name to vt100: $ export TERM=vt100 Use the following format under the TC Shell (tcsh): setenv TERM name Replace name with the terminal name for the terminal you are using. Under tcsh you can place this command in your ~/.login file (page 352). For example, under tcsh you can give this command to set your terminal name to
vt100: $ setenv TERM vt100 LANG For some programs to display information correctly, you may need to set the LANG variable (page 304). Frequently you can set this variable to C. Under bash use the command $ export LANG=C and under tcsh use $ setenv LANG C This page intentionally left blank C Keeping the System Up-to-Date In This
Appendix Using yum . . . . 910 Using apt-get . . . . 916 BitTorrent. . . . . 921 The apt-get and yum utilities both fill the same role: They install and update software packages. Both utilities compare the files in a repository (generally on the Internet) with those on the local system and update the files on
the local system according to your instructions. Both utilities automatically install and update any additional files that a package depends on (dependencies). Most Linux distributions come with apt-get or yum. Debianbased systems such as Ubuntu are set up to use apt-get, which works with deb packages. Red Hat and Fedora use yum, which works
with rpm packages. There are also versions of apt-get that work with rpm packages. On a Mac OS X system it is easiest to use the software update GUI to keep the system up-to-date. CAppendixC To facilitate the update process, apt-get and yum keep locally a list of packages that are held in each of the repositories they use. Any software you want to
install or update must reside in a repository. When you give apt-get or yum a command to install a package, they look for the package in a local package list. If the package appears in the list, apt-get or yum fetches both that package and any packages the package you are installing depends on and installs the packages. 909 910 Appendix C Keeping
the System Up-to-Date The yum examples in this section are from a Fedora system and the apt-get examples are from an Ubuntu system. Although the files, input, and output on the local system may look different, how you use the tools—and the results—will be the same. In contract to apt-get and yum, BitTorrent efficiently distributes large amounts
of static data, such as installation ISO images. It does not examine files on the local system and does not check for dependencies. Using yum Early releases of Linux did not include a tool for managing updates. Although the rpm utility could install or upgrade individual software packages, it was up to the user to locate a package and any packages it
depended on. When Terra Soft produced its Red Hat–based Linux distribution for the PowerPC, named Yellow Dog, the company created the Yellow Dog Updater to fill this gap. This program has since been ported to other architectures and distributions. The result is named Yellow Dog Updater, Modified (yum; yum.baseurl.org). rpm packages The
yum utility works with rpm packages. When yum installs or upgrades a software package, it also installs or upgrades packages that the package depends on. Repositories The yum utility downloads package headers and packages from servers called repositories. Frequently, yum is set up to use copies of repositories kept on mirror sites. See
“Configuring yum” on page 914 for information on selecting a repository. Using yum to Install, Remove, and Update Packages Installing packages The behavior of yum depends on the options you specify. To install a new package together with the packages it depends on, while working with root privileges give the command yum install, followed by
the name of the package. After yum determines what it needs to do, it asks for confirmation. The next example installs the tcsh package: # yum install tcsh Loaded plugins: refresh-packagekit Setting up Install Process Resolving Dependencies --> Running transaction check ---> Package tcsh.i586 0:6.15-8.fc11 set to be updated --> Finished
Dependency Resolution Dependencies Resolved == Package Arch Version Repository Size ==
Installing: tcsh i586 6.15-8.fc11 fedora 477 k Transaction Summary == Install 1 Package(s) Update 0 Package(s) Remove 0 Package(s) Using yum 911 Total download size: 477 k Is this ok [y/N]: y Downloading Packages:
tcsh-6.15-8.fc11.i586.rpm Running rpm_check_debug Running Transaction Test Finished Transaction Test Transaction Test Succeeded Running Transaction Installing : tcsh-6.15-8.fc11.i586 | 477 kB 00:00 1/1 Installed: tcsh.i586 0:6.15-8.fc11 Complete! Removing packages You can also use yum to remove packages, using a similar syntax. The
following example removes the tcsh package: # yum remove tcsh Loaded plugins: refresh-packagekit Setting up Remove Process Resolving Dependencies --> Running transaction check ---> Package tcsh.i586 0:6.15-8.fc11 set to be erased --> Finished Dependency Resolution Dependencies Resolved
== Package Arch Version Repository Size == Removing: tcsh i586 6.15-8.fc11 installed 1.1 M
Transaction Summary == Install 0 Package(s) Update 0 Package(s) Remove 1 Package(s) Is this ok [y/N]: y Downloading Packages: Running rpm_check_debug Running Transaction Test Finished Transaction Test
Transaction Test Succeeded Running Transaction Erasing : tcsh-6.15-8.fc11.i586 1/1 Removed: tcsh.i586 0:6.15-8.fc11 Complete! Updating packages The update option, without additional parameters, updates all installed packages. It downloads summary files that hold information about package headers for installed packages, determines which
packages need to be updated, prompts to continue, and downloads and installs the updated packages. Unlike the situation with apt-get, the yum upgrade command is very similar to yum update. In the following example, yum determines that two packages, at and firefox, need to be updated and checks dependencies. Once it has determined what it
needs to do, yum advises you of the action(s) it will take, prompts with Is this ok [y/N], and, if you approve, downloads and installs the packages. 912 Appendix C Keeping the System Up-to-Date # yum update Loaded plugins: refresh-packagekit Setting up Update Process Resolving Dependencies --> Running transaction check ---> Package at.i586
0:3.1.10-31.fc11 set to be updated ---> Package firefox.i586 0:3.5.1-3.fc11 set to be updated --> Finished Dependency Resolution Dependencies Resolved == Package Arch Version Repository Size
== Updating: at i586 3.1.10-31.fc11 updates 63 k firefox i586 3.5.1-3.fc11 updates 15 M Transaction Summary
== Install 0 Package(s) Update 2 Package(s) Remove 0 Package(s) Total download size: 15 M Is this ok [y/N]: y Downloading Packages: (1/2): at-3.1.10-31.fc11.i586.rpm | 63 kB 00:00 (2/2): firefox-3.5.1-3.fc11.i586.rpm | 15
MB 00:10 --Total 1.3 MB/s | 15 MB 00:11 Running rpm_check_debug Running Transaction Test Finished Transaction Test Transaction Test Succeeded Running Transaction Updating : firefox-3.5.1-3.fc11.i586 1/4 Updating : at-3.1.10-31.fc11.i586 2/4 Cleanup : firefox-3.5.1-1.fc11.i586 3/4 Cleanup :
at-3.1.10-30.fc11.i586 4/4 Updated: at.i586 0:3.1.10-31.fc11 firefox.i586 0:3.5.1-3.fc11 Complete! You can update individual packages by specifying the names of the packages on the command line following the word update. Other yum Commands Many yum commands and options are available. A few of the more useful commands are described here.
The yum man page contains a complete list. check-update Lists packages installed on the local system that have updates available in the yum repositories. clean all Removes header files that yum uses for resolving dependencies. Also removes cached packages—yum does not automatically remove packages once they have been downloaded and
installed, unless you set keepcache (page 914) to 0. clean metadata Removes the files yum uses to determine remote package availability. Using this option forces yum to download all metadata the next time you run it. list available Lists all packages that can be installed from the yum repositories. Using yum 913 search word Searches for word in the
package description, summary, packager, and name. yum Groups In addition to working with single packages, yum can work with groups of packages. The next example shows how to display a list of installed and available package groups: $ yum grouplist Loaded plugins: refresh-packagekit Setting up Group Process Installed Groups: Administration
Tools Arabic Support Armenian Support Assamese Support Base ... Web Server X Window System Available Groups: ... Italian Support Java Java Development KDE (K Desktop Environment) KDE Software Development Kashmiri Support Done The command yum groupinfo followed by the name of a group displays information about the group, including
a description of the group and a list of mandatory, default, and optional packages. The next example displays information about the Web Server group of packages. If the name of the package includes a SPACE, you must quote it. $ yum groupinfo "Web Server" Loaded plugins: refresh-packagekit Setting up Group Process Group: Web Server
Description: These tools allow you to run a Web server on the system. Mandatory Packages: httpd Default Packages: crypto-utils distcache httpd-manual mod_perl mod_python ... Optional Packages: Pound apachetop awstats boa cherokee 914 Appendix C Keeping the System Up-to-Date To install a group of packages, give the command yum
groupinstall followed by the name of the group. Downloading rpm Package Files with yumdownloader The yumdownloader utility locates and downloads—but does not install—rpm package files. If this utility is not available on the local system, use yum to download and install the yum-utils package before you attempt to run yumdownloader. The
following example downloads the samba rpm file to the working directory: $ yumdownloader samba Loaded plugins: refresh-packagekit fedora/primary_db updates/primary_db samba-3.3.2-0.33.fc11.i586.rpm Downloading source files | 8.4 MB | 2.6 MB | 4.4 MB 00:12 00:07 00:08 You can use yumdownloader with the ––source option to download rpm
source package files. The yumdownloader utility automatically enables the necessary source repositories. The following example downloads in the working directory the rpm file for the latest version of the kernel source code for the installed release: $ yumdownloader --source kernel Loaded plugins: refresh-packagekit Enabling updates-source
repository updates-source/metalink updates-source updates-source/primary_db Enabling fedora-source repository fedora-source/metalink fedora-source fedora-source/primary_db kernel-2.6.29.6-213.fc11.src.rpm | 5.7 kB | 3.3 kB | 620 kB 00:00 00:00 00:02 | 14 kB | 2.7 kB | 2.4 MB | 56 MB 00:00 00:00 00:04 02:37 Without the ––source option,
yumdownloader downloads the executable kernel rpm file. Configuring yum Most Linux distributions that use yum for updating files come with yum ready to use; you do not need to configure it. This section describes the yum configuration files for those users who want to modify them. The primary configuration file, /etc/yum.conf, holds global
settings. The first example shows a typical yum.conf file: $ cat /etc/yum.conf [main] cachedir=/var/cache/yum keepcache=0 debuglevel=2 logfile=/var/log/yum.log exactarch=1 obsoletes=1 gpgcheck=1 Using yum 915 plugins=1 installonly_limit=3 ... # PUT YOUR REPOS HERE OR IN separate files named file.repo # in /etc/yum.repos.d The section
labeled [main] defines global configuration options. The cachedir specifies the directory in which yum stores downloaded packages, although with keepcache set to 0, yum does not store headers and packages after installing them. The amount of information logged is specified by debuglevel, with a value of 10 producing the most information. The
logfile specifies where yum keeps its log. Setting exactarch to 1 causes yum to update packages only with packages of the same architecture, thereby preventing an i686 package from replacing an i386 package, for example. You can use retries to specify the number of times yum will try to retrieve a file before returning an error (the default is 6). Set
this parameter to 0 to cause yum to continue trying forever. Setting obsoletes to 1 causes yum to replace obsolete packages when performing an update; it has no effect during an install. When gpgcheck is set to 1, yum checks the GPG (GNU Privacy Guard; GnuPG.org) signatures on packages it installs. This check verifies the authenticity of the
packages. Setting plugins to 1 enables yum plugins, which extend yum functionality. (See wiki.linux.duke.edu/YumPlugins and yum in section 8 of the man pages for more information on yum plugins.) The installonly parameter specifies packages that yum is to install but never upgrade, such as the kernel. The installonly_limit specifies the number of
versions of a given installonly package that are to be installed at one time. Although the balance of the yum configuration information, which specifies the yum repositories, can appear in the yum.conf file, this information is frequently kept in the /etc/yum.repos.d directory. A parameter set in a repository section overrides the same parameter set in
the [main] section. Following is a listing for a yum.repos.d directory on a Fedora system: $ ls /etc/yum.repos.d fedora-rawhide.repo fedora-updates-testing.repo fedora-updates.repo fedora.repo Each of these files contains a header, such as [fedora], which provides a unique name for the repository. The name of the file is generally similar to the
repository name, with the addition of a fedora- (or similar) prefix and a .repo filename extension. On a Fedora system, commonly used repositories include fedora (held in the fedora.repo file), which contains the packages present on the installation DVD; updates (held in the fedora-updates.repo file), which contains updated versions of the stable
packages; updates-testing (held in the fedora-updates-testing.repo file), which contains updates that are not ready for release; and rawhide (held in fedora.rawhide.repo), which contains a daily snapshot of the latest build versions of Fedora packages. The last two repositories are not enabled; do not enable either of these repositories unless you are
testing unstable packages. Never enable them on a production system. 916 Appendix C Keeping the System Up-to-Date Each *.repo file can specify several repositories tip Each *.repo file includes specifications for several related repositories, which are usually disabled. For example, the fedora.repo file holds [fedora-debuginfo] and [fedora-source] in
addition to [fedora]. You cannot download source files using yum. Instead, use yumdownloader (page 914) for this task. The next example shows part of the fedora.repo file that specifies the parameters for the fedora repository: $ cat /etc/yum.repos.d/fedora.repo [fedora] name=Fedora $releasever - $basearch failovermethod=priority #baseurl=
releasever/Everythng/$basearch/os/ mirrorlist= enabled=1 metadata_expire=7d gpgcheck=1 gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-$basearch ... Repository specification Each repository specification contains the name of the repository enclosed in brackets ([]), a name, a failovermethod, a baseurl, and a mirrorlist. The name provides
an informal name for the repository that yum displays. The failovermethod determines the order in which yum contacts an initial mirror site and additional mirror sites if the first one fails; priority selects the sites in the order in which they appear, while roundrobin selects sites randomly. The baseurl indicates the location of the main repository; it is
normally commented out. The mirrorlist specifies the URL of a file that holds a list of baseurls, or mirrors of the main repository. The mirror list server uses geoip (geolocation; www.geoiptool.com) to attempt to return the nearest mirrors for yum to try. Only baseurl or mirrorlist can be enabled at one time. These definitions use two variables: yum
sets $basearch to the architecture of the system and $releasever to the version of the release (such as 11 for Fedora 11). The repository described by the file is enabled (yum will use it) if enabled is set to 1 and is disabled if enabled is set to 0. As described earlier, gpgcheck determines whether yum checks GPG signatures on files it downloads. The
gpgkey specifies the location of the GPG key. Refer to the yum.conf man page for more options. Using apt-get APT (Advanced Package Tool) is a collection of utilities that download, install, remove, upgrade, and report on software packages. APT utilities download packages and call dpkg utilities to manipulate the packages once they are on the local
system. For more information refer to www.debian.org/doc/manuals/apt-howto. Using apt-get 917 Updating the package list The primary APT command is apt-get; its arguments determine what the command does. Working with root privileges, give the command apt-get update to update the local package list: # apt-get update Get:1 stable
Release.gpg [191B] Get:2 testing Release.gpg [191B] Get:3 stable Release [1308B] Get:4 testing Release [1297B] Get:5 stable/non-free Packages [966B] Hit hardy Release.gpg Hit hardy-updates Release.gpg Hit hardy-security Release.gpg Get:6 testing/non-free Packages [734B] Hit hardy Release Hit hardy Release ... Fetched
4687B in 1s (2895B/s) Reading package lists... Done Check the dependency tree The apt-get utility does not tolerate a broken dependency tree. To check the status of the local dependency tree, give the command apt-get check: # apt-get check Reading package lists... Done Building dependency tree Reading state information... Done The easiest way
to fix errors that apt-get reveals is to remove the offending packages and then reinstall them using apt-get. Using apt-get to Install, Remove, and Update Packages Installing packages The following command uses apt-get to install the zsh package: # apt-get install zsh Reading package lists... Done Building dependency tree Reading state information...
Done Suggested packages: zsh-doc The following NEW packages will be installed: zsh 0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded. Need to get 3892kB of archives. After this operation, 11.6MB of additional disk space will be used. Get:1 hardy/main zsh 4.3.4-24ubuntu1 [3892kB] Fetched 3892kB in 23s (165kB/s) Selecting
previously deselected package zsh. (Reading database ... 192924 files and directories currently installed.) Unpacking zsh (from .../zsh_4.3.4-24ubuntu1_amd64.deb) ... Setting up zsh (4.3.4-24ubuntu1) ... 918 Appendix C Keeping the System Up-to-Date Removing packages Remove a package the same way you install a package, substituting remove for
install: # apt-get remove zsh Reading package lists... Done Building dependency tree Reading state information... Done The following packages will be REMOVED: zsh 0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded. After this operation, 11.6MB disk space will be freed. Do you want to continue [Y/n]? y (Reading database ... 193880
files and directories currently installed.) Removing zsh ... To ensure you can later reinstall a package with the same configuration, the apt-get remove command does not remove configuration files from the /etc directory hierarchy. Although it is not recommended, you can use the purge command instead of remove to remove all the package files,
including configuration files. Alternatively, you can move these files to an archive so you can restore them later if necessary. Using apt-get to Upgrade the System Two arguments cause apt-get to upgrade all packages on the system: upgrade upgrades all packages on the system that do not require new packages to be installed and dist-upgrade
upgrades all packages on the system, installing new packages as needed; this argument will install a new version of the operating system if one is available. The following command updates all packages on the system that depend only on packages that are already installed: # apt-get upgrade Reading package lists... Done Building dependency tree
Reading state information... Done The following packages will be upgraded: cups cups-bsd cups-client cups-common firefox firefox-3.0 firefox-3.0-branding firefox-3.0-gnome support firefox-gnome-support 9 upgraded, 0 newly installed, 0 to remove and 0 not upgraded. Need to get 4764kB of archives. After this operation, 0B of additional disk space
will be used. Do you want to continue [Y/n]? y Get:1 jaunty-updates/main cups-common 1.3.9-17ubuntu3.2 [1165kB] ... Get:8 jaunty-updates/main firefox 3.0.12+build1+nobinonly-0ubuntu0.9.04.1 [69.3kB] Get:9 jaunty-updates/main firefox-gnome-support 3.0.12+build1+nobinonly0ubuntu0.9.04.1 [69.2kB] Fetched 4764kB in 44s (107kB/s)
Preconfiguring packages ... (Reading database ... 123035 files and directories currently installed.) Preparing to replace cups-common 1.3.9-17ubuntu3 (using .../cups-common_1.3.9-17ubuntu3.2_all.deb) ... Unpacking replacement cups-common ... Preparing to replace cups 1.3.9-17ubuntu3 (using .../cups_1.3.9-17ubuntu3.2_i386.deb) ... * Stopping
Common Unix Printing System: cupsd [OK] Using apt-get 919 Unpacking replacement cups ... ... Setting up firefox (3.0.12+build1+nobinonly-0ubuntu0.9.04.1) ... Setting up firefox-gnome-support (3.0.12+build1+nobinonly-0ubuntu0.9.04.1) ... When apt-get asks if you want to continue, enter Y to upgrade the listed packages; otherwise, enter N.
Packages that are not upgraded because they depend on packages that are not already installed are listed as kept back. Use dist-upgrade to upgrade all packages, including packages that depend on packages that are not installed. This command also installs dependencies. Other apt-get Commands autoclean Removes old archive files. check Checks
for broken dependencies. clean Removes archive files. dist-upgrade Upgrades packages on the system, installing new packages as needed. If a new version of the operating system is available, this option upgrades to the new version. purge Removes a package and all its configuration files. source Downloads source files. update Retrieves new lists of
packages. upgrade Upgrades all packages on the system that do not require new packages to be installed. Repositories Repositories hold collections of software packages and related information, including headers that describe each package and provide information on other packages the package depends on. Typically, a Linux distribution maintains
repositories for each of its releases. Software package categories Software packages are frequently divided into several categories. Ubuntu uses the following categories: • main—Ubuntu-supported open-source software • universe—Community-maintained open-source software • multiverse—Software restricted by copyright or legal issues • restricted
—Proprietary device drivers • backports—Packages from later releases of Ubuntu that are not available for an earlier release The apt-get utility selects packages from repositories it searches based on the categories specified in the sources.list file. 920 Appendix C Keeping the System Up-to-Date sources.list: Specifies Repositories for apt-get to Search
The /etc/apt/sources.list file specifies the repositories apt-get searches when you ask it to find or install a package. You must modify the sources.list file to enable apt-get to download software from nondefault repositories. Typically, you do not need to configure apt-get to install supported software. Each line in sources.list describes one repository and
has the following format: type URI repository category-list where type is deb for packages of executable files and deb-src for packages of source files; URI is the location of the repository, usually cdrom or an Internet address that starts with http://; repository is the name of the repository apt-get is to search; and category-list is a SPACE-separated list
of categories apt-get selects packages from. Comments begin with a pound sign (#) anywhere on a line and end at the end of the line. The following line from sources.list on an Ubuntu system causes apt-get to search the Jaunty archive located at us.archive.ubuntu.com/ubuntu for deb packages that contain executable files. It accepts packages that
are categorized as main, restricted, and multiverse: deb jaunty main restricted multiverse Replace deb with deb-src to search for packages of source files in the same manner. Use the apt-get source command to download source packages. Default repositories The default sources.list file on an Ubuntu system includes repositories such as jaunty
(Jaunty as originally released), jaunty-updates (major bug fixes after the release of Jaunty), jaunty-security (critical security-related updates), and backports (newer, less-tested software that is not reviewed by the Ubuntu security team). Some repositories in sources.list may be commented out. Remove the leading pound sign (#) on the lines of the
repositories you want to enable. After you modify sources.list, give the command apt-get update (page 917) to update the local package indexes. The next line, which was added to sources.list, enables apt-get to search a third-party repository (but see the following security tip): deb stable non-free In this case, the repository is named stable and the
category is non-free. Although the code is compiled for Debian, it runs on Ubuntu, as is frequently the case. Use repositories you trust security There are many repositories of software packages. Be selective in which repositories you add to sources.list: When you add a repository, you are trusting the person who runs the repository not to put
malicious software in packages you may download. In addition, unsupported packages may conflict with other packages or cause upgrades to fail. BitTorrent 921 BitTorrent The BitTorrent protocol implements a hybrid client/server and P2P (page 970) file transfer mechanism. BitTorrent efficiently distributes large amounts of static data, such as the
Linux installation ISO images. It can replace protocols such as anonymous FTP, where client authentication is not required. Each BitTorrent client that downloads a file provides additional bandwidth for uploading the file, thereby reducing the load on the initial source. In general, BitTorrent downloads proceed more rapidly than FTP downloads.
Unlike protocols such as FTP, BitTorrent groups multiple files into a single package: a BitTorrent file. Tracker, peer, seed, and swarm BitTorrent, like other P2P systems, does not use a dedicated server. Instead, the functions of a server are performed by the tracker, peers, and seeds. The tracker is a server that allows clients to communicate with
each other. Each client—called a peer when it has downloaded part of the BitTorrent file and a seed once it has downloaded the entire BitTorrent file—acts as an additional source for the BitTorrent file. Peers and seeds are collectively called a swarm. As with a P2P network, a member of a swarm uploads to other clients the sections of the BitTorrent
file it has already downloaded. There is nothing special about a seed: It can be removed at any time once the torrent is available for download from other seeds. The torrent The first step in downloading a BitTorrent file is to locate or acquire the torrent, a file with the filename extension of .torrent. A torrent contains pertinent information (metadata)
about the BitTorrent file to be downloaded, such as its size and the location of the tracker. You can obtain a torrent by accessing its URI, or you can acquire it via the Web, an email attachment, or other means. The BitTorrent client can then connect to the tracker to learn the locations of other members of the swarm that it can download the
BitTorrent file from. Manners Once you have downloaded a BitTorrent file (i.e., the local system has become a seed), it is good manners to allow the local BitTorrent client to continue to run so peers (clients that have not downloaded the entire BitTorrent file) can download at least as much information as you have downloaded. Prerequisites If
necessary, use yum (page 910) or apt-get (page 916) to install the bittorrent package. With this package installed, the command apropos bittorrent displays a list of BitTorrent utilities. See /usr/share/doc/bittorrent for more information. You may want to try BitTornado, an experimental BitTorrent client with additional features (bittornado package; see
bittornado.com) Because the BitTorrent utilities are written in Python and run on any platform with a Python interpreter, they do not depend on system architecture. Python is installed in /usr/bin/python and is available in the python package. 922 Appendix C Keeping the System Up-to-Date Using BitTorrent The btdownloadcurses utility is a textual
BitTorrent client that provides a pseudographical interface. Once you have a torrent, give a command such as the following, substituting the name of the torrent you want to download for the Ubuntu torrent in the example: $ btdownloadcurses ubuntu-9.04-desktop-i386.iso.torrent In the preceding command, the torrent specifies that the BitTorrent
file be saved as ubuntu-9.04-desktop-i386.iso in the working directory. The name of the BitTorrent file is not always the same as the name of the torrent. In the case of a multifile torrent, the BitTorrent files may be stored in a directory, also named by the torrent. Depending on the speed of the Internet connection and the number of seeds,
downloading a large BitTorrent file can take anywhere from hours to days. Following is what btdownloadcurses looks like when it is running: --| file: ubuntu-9.04-desktop-i386.iso | | size: 732,909,568 (699.0 M) | | dest: /p02/ubuntu-9.04-desktop-i386.iso | | progress: ########____________________________ | | status:
finishing in 0:21:13 (23.6%) | | speed: 449.7 K/s down | | totals: 23.6 M down | | error(s): | | | -- You can abort the download by pressing q or CONTROL-C. The download will automatically resume from where it left off when you download the same torrent to the same location again. Make sure you have enough room to
download the torrent caution Some torrents are huge. Make sure the partition you are working in has enough room to hold the BitTorrent file you are downloading. See the btdownloadcurses man page for a list of options. One of the most useful options is ––max_upload_rate, which limits how much bandwidth the swarm can use while downloading the
torrent from you. The default is 0, meaning there is no limit to the upload bandwidth. The following command prevents BitTorrent from using more than 10 kilobytes per second of upstream bandwidth: $ btdownloadcurses --max_upload_rate 10 ubuntu-9.04-desktop-i386.iso.torrent BitTorrent usually allows higher download rates for members of the
swarm that upload more data, so it is to your advantage to increase this value if you have spare bandwidth. You need to leave enough free upstream bandwidth for the acknowledgment packets from your download to get through or else the download will be very slow. By default, btdownloadcurses uploads to a maximum of seven other clients at
BitTorrent 923 once. You can change this number by using the ––max_uploads argument, followed by the number of concurrent uploads you wish to permit. If you are downloading over a modem, try setting ––max_upload_rate to 3 and ––max_uploads to 2. The name of the file or directory that BitTorrent saves a file or files in is specified by the torrent.
You can specify a different file or directory name by using the ––saveas option. The btshowmetainfo utility displays the name the BitTorrent file will be saved as, the size of the file, the name of the torrent (metainfo file), and other information: $ btshowmetainfo ubuntu-9.04-desktop-i386.iso.torrent btshowmetainfo 20021207 - decode BitTorrent
metainfo files metainfo file.: info hash.....: file name.....: file size.....: announce url..: ubuntu-9.04-desktop-i386.iso.torrent 60d5d82328b4547511fdeac9bf4d0112daa0ce00 ubuntu-9.04-desktop-i386.iso 732909568 (1397 * 524288 + 479232) This page intentionally left blank D Mac OS X Notes In This Appendix Open Directory . . . . 926 This
appendix is a brief guide to Mac OS X features that differ from those of Linux. See Chapter 1 for a history of UNIX, Linux, and OS X. DAppendixD /etc/group. . . . . 926 Filesystems . . . . 927 Extended Attributes . . . . 928 Activating the META Key . . . . 935 Startup Files . . . . 936 Remote
Logins. . . . . 936 Many Utilities Do Not Respect Apple Human Interface Guidelines . . . . 936 Mac OS X Implementation of Linux Features . . . . 936 925 926 Appendix D Mac OS X Notes Open Directory Open Directory replaced the monolithic NetInfo database in Mac OS X version 10.5. The ni* utilities,
including nireport and nidump, were replaced by dscl (page 674). The work the lookupd daemon did is now done by the DirectoryService daemon. For information on the local system, both Mac OS X and Mac OS X Server now use a hierarchy of small *.plist XML files called nodes, which are stored in the /var/db/dslocal hierarchy. Many of these files
are human readable. On Mac OS X Server, a network-wide Open Directory is based on OpenLDAP, Kerberos, and the SASL-based Password Server. /etc/passwd Because OS X does not use the /etc/passwd file to store most user information, examples in this book that use this file do not run under OS X. In most cases you must use dscl to extract
information from the passwd database. As an example, the whos2 program (following) is a version of whos (page 413) that runs under OS X 10.5 and later. whos2 For each command-line argument, whos2 searches the passwd database. Inside the for loop, the dscl (page 674) –readall command lists all usernames and user IDs on the local system. The
command looks for the RealName keys in the /Users directory and displays the associated values. The four-line sed (page 565) command deletes lines containing only a dash (/^-$/ d), finds lines containing only RealName: (/^RealName:$/), reads and appends the next line (N; page 568), and substitutes a semicolon for the NEWLINE (s//; /). Finally,
grep (page 719) selects lines containing the ID the program was called with. $ cat whos2 #!/bin/bash if [$# -eq 0] then echo "Usage: whos id..." 1>&2 exit 1 fi for id do dscl . -readall /Users RealName | sed '/^-$/ d /^RealName:$/N;s/// N s//; /' | grep -i "$id" done /etc/group Groups (page 956) allow users to share files or programs without allowing all
system users access to them. This scheme is useful if several users are working with files that are not public. In a Linux system, the /etc/group file associates one or more usernames with each group (number). Mac OS X 10.5 and later rely on Filesystems 927 Open Directory (page 926) to provide group information. OS X 10.4 and earlier use NetInfo
for this task. Filesystems Mac OS X supports several types of filesystems. The most commonly used is the default HFS+ (Hierarchical File System plus). Introduced under Mac OS 8.1 to support larger disks, HFS+ is an enhanced version of the original HFS filesystem. When it was introduced in OS 3.0 in 1986, HFS stood in contrast to the then-
standard MFS (Macintosh File System). Some applications will not run correctly under filesystems other than HFS+. HFS+ filesystems do not have inodes (page 959) but, through an elaborate scheme, appear to have inodes. HFS+ is different from Linux filesystems, but because Linux offers a standardized filesystem interface, the differences are
generally transparent to the user. The most significant difference lies in the handling of extended attributes (page 928). OS X also supports Linux filesystems such as UFS (UNIX File System), which it inherited from Berkeley UNIX. Other supported filesystems include FAT16 and FAT32, which were originally used in MS-DOS and Windows. These
filesystems are typically used on removable media, such as digital camera storage cards. Also supported is ISO9660, which is used on CD-ROMs, and the related UDF filesystem, which is used on DVDs. Nondisk Filesystems Mac OS X supports filesystems that do not correspond to a physical volume such as a partition on a hard disk or a CD-ROM. A
.dmg (disk image) file is one example (you can mount a disk image file so you can access the files it holds by double-clicking it in the Finder). Another example is a virtual filesystem in which the filenames represent kernel functionality. For example, the /Network virtual filesystem holds a directory tree representing the local network; most network
filesystem protocols use this filesystem. Also, you can use the Disk Utility to create encrypted or password-protected .iso (ISO9660; page 961) image files you can mount. Case Sensitivity The default OS X filesystem, HFS+, is case preserving but not case sensitive. Case preserving means the filesystem remembers which capitalization you used when
you created a file and displays the filename with that capitalization, but accepts any capitalization to refer to the file. Thus, under HFS+, files named JANUARY, January, and january refer to the same file. The HFS+ Extended (HFSX) filesystem is case sensitive and is fully supported under Mac OS X Server only. Because HFSX is case sensitive, the
three example filenames refer to three different files. 928 Appendix D Mac OS X Notes /Volumes Each physical hard disk in a Mac OS X system is typically divided into one or more logical sections (partitions) that, when formatted, are called volumes. On a Linux system they are called filesystems. Startup disk Each OS X system has a volume, called
the startup disk, that the system boots from. By default the startup disk is named Macintosh HD. When the system boots, Macintosh HD is mounted as the root directory (/). The root directory always has a subdirectory named Volumes. For historical reasons, every volume other than the startup disk is mounted in the /Volumes directory. The pathname
of a disk labeled MyPhotos is /Volumes/MyPhotos. To simplify access, /Volumes holds a symbolic link (page 108) to the startup disk (the root directory). Assuming that the startup disk is named Macintosh HD, /Volumes/Macintosh HD is a symbolic link to /. $ ls -ld '/Volumes/Macintosh HD' lrwxr-xr-x 1 root admin 1 Jul 12 19:03 /Volumes/Macintosh HD
-> / The system automatically mounts all volumes in /Volumes. The desktop presented by the Finder contains an icon for each mounted disk and for files in the user's Desktop directory, making the /Volumes directory the effective root (top level or /) for the Finder and other applications. The Finder presents the pre-UNIX Mac OS view of the filesystem.
The files in /Volumes are the same as the first components of Carbon pathnames. Carbon Pathnames Carbon pathnames differ from OS X pathnames and are discussed here because many applications use them. Some operating system features are accessible through Carbon API calls only. The Carbon environment uses the historical Macintosh model
of a desktop holding mounted disks, which does not correspond well to the Linux filesystem's singledirectory hierarchy. Furthermore, the Carbon environment represents pathnames differently from the Linux environment. Carbon pathnames use a colon (:) as a separator, instead of a slash (/). No top-level root exists for Carbon pathnames; instead,
they start with the name of a disk. For example, the Linux-style pathname /Volumes/Images/backups equates to the Carbon-style pathname Images:backups. Know which kind of pathname you are using caution If you enter a Linux pathname when a Carbon pathname is expected, or vice versa, the system may display an error message, not retrieve the
correct file, or save the file in the wrong location. Extended Attributes Mac OS X files have extended attributes, which include file attributes used by the Finder, Access Control Lists, and resource forks. Not all utilities recognize extended Extended Attributes 929 attributes. This section describes extended attributes, which include resource forks, file
attributes, and ACLs. Resource forks and file attributes are native to the HFS+ filesystem. Mac OS X emulates resource forks and file attributes on other types of filesystems. These features are not found on Linux filesystems. Some utilities do not process extended attributes caution Some third-party programs and most utilities under Mac OS X 10.3
and earlier do not support extended attributes. Some utilities require options to process extended attributes. Some utilities, on systems with ACLs enabled, have a bug involving extended attributes. For more information, see the “Enabling ACLs may cause errors” caution on page 932. See also the tip “Redirection does not support resource forks” on
page 930. File Forks Forks are segments of a single file, each holding different content. Mac OS has supported file forks since its inception. The most widely used are the data fork and the resource fork. Data forks The data fork is equivalent to a Linux file. It consists of an unstructured stream of bytes. Many files have only a data fork. Resource forks
The resource fork holds a database that allows random access to resources, each of which has a type and identifier number. Modifying, adding, or deleting one resource has no effect on the other resources. Resource forks can store different types of information—some critical and some merely useful. For example, a Mac OS graphics program might
save a smaller copy of an image (a preview or thumbnail) in a resource fork. Also, text files created with the BBEdit text editor store display size and tab stop information in the file's resource fork. Because this program is a text editor and not a word processor, this type of information cannot be stored in the data fork. Losing a resource fork that holds
a thumbnail or display information is at most an inconvenience because, for example, the thumbnail can be regenerated from the original image. Other programs store more important data in the resource fork or create files that contain only a resource fork, which contains all the file's information. In this case, losing the resource fork is just as bad as
losing the data fork. It may even be worse: You may not notice a resource fork is missing because the data fork is still there. You might notice the loss only when you try to use the file. A Linux filesystem associates each filename with a single stream of bytes. Forks do not fit this model. As a result, many Linux utilities do not process resource forks, but
instead process only data forks. Most of the file utilities provided with OS X 10.4 and later support resource forks. Many third-party utilities do not support resource forks. Pipes do not work with resource forks caution Pipes work with the data fork of a file only; they do not work with resource forks. 930 Appendix D Mac OS X Notes Redirection does
not support resource forks caution When you redirect input to or output from a utility (page 126), only the information in the data fork is redirected. Information in the resource fork is not redirected. For example, the following command copies the data fork of song.ogg only: $ cat song.ogg > song.bak.ogg optional ..namedfork You can verify the
existence of file forks on an HFS+ filesystem. The ..namedfork pseudodirectory, as a subdirectory of a file, holds files that correspond to a file's forks. The two most common files are data, which holds the data fork, and rsrc, which holds the resource fork. The following example shows how to use ..namedfork to verify the existence of and display the
size of the data and resource forks associated with the pic.jpg file. You cannot list the ..namedfork pseudodirectory. $ ls -l pic.jpg -rwxr-xr-x 1 max max 13432 Jul 18 15:15 pic.jpg $ ls -l pic.jpg/..namedfork ls: pic.jpg/..namedfork: Not a directory $ ls -l pic.jpg/..namedfork/data -rwxr-xr-x 1 max max 13432 Jul 18 15:15 pic.jpg/..namedfork/data $ ls -l
pic.jpg/..namedfork/rsrc -rwxr-xr-x 1 max max 140622 Jul 18 15:15 pic.jpg/..namedfork/rsrc The data file is human readable if the underlying file is human readable. The rsrc file is not in a strictly human-readable format, although you may be able to garner some information by viewing these files. If you are unsure whether a program supports
resource forks, test it before relying on this functionality. Make a backup copy using the Finder, ditto, or, under version 10.4 and later, cp. Then check whether the copied file works correctly. Table D-1 lists utilities that manipulate resource forks. These utilities are installed with the Developer Tools package. Consult the respective man pages for
detailed descriptions. Table D-1 Utilities that manipulate resource forks Utility Function Rez Creates a resource fork from a resource description file DeRez Creates a resource description file from a resource fork RezWack Converts a file with forks to a single flat file that holds all the forks UnRezWack Converts a flat file that holds forks to a file with
forks SplitForks Converts a file with forks to multiple files, each holding a fork Extended Attributes 931 File Attributes A file contains data. Information about the file is called metadata. Examples of metadata include ownership information and permissions for a file. Mac OS X stores more metadata than Linux stores. This section discusses file
attributes, the metadata stored by Mac OS X. File attributes include the following: • Attribute flags • Type codes • Creator codes The same caveats apply to file attributes as apply to resource forks: Some utilities may not preserve them when copying or manipulating files, and many utilities do not recognize attribute flags. Loss of file attributes is
particularly common when you move files to non-Macintosh systems. Attribute Flags Attribute flags (Table D-2) hold information that is distinct from Linux permissions. Two attribute flags are the invisible flag (which keeps a file from being displayed in file dialogs and in the Finder) and the locked flag (which keeps a file from being modified). Flags
are generally ignored by command-line utilities and affect only GUI applications. The ls utility lists files that have the invisible flag set. See GetFileInfo on page 717 and SetFile on page 813 for information on displaying, setting, and clearing attribute flags. Table D-2 Attribute flags Flag Can the flag be set on a directory? Description a No Alias file c
Yes Custom icon l No Locked t No Stationery pad file v Yes Invisible Creator Codes and Type Codes Type codes and creator codes are 32-bit integers, generally displayed as 4-character words, that specify the type and creator of a file. The creator code specifies the application that created a document, not the user who created it. An application can
typically open documents that have the same creator code as the application, but cannot open documents with other creator codes. 932 Appendix D Mac OS X Notes Creator codes Creator codes generally correspond to vendors or product lines. The operating system—and in particular the Finder—uses creator codes to group related files. For
example, the AppleWorks application file and its document files have the creator code BOBO. In the file browser in an application's open file dialog box, grayed-out files generally indicate files that have a different creator code from the application. The open utility (page 780) also looks at creator codes when it opens a file. Type codes Type codes
indicate how a file is used. The type code APPL indicates an application, a program used to open other files. For example, an AppleWorks word processor document has the type code CWWP, a mnemonic for Claris Works Word Processor (AppleWorks used to be named Claris Works). While a few type codes, such as the application type, are
standardized, vendors are free to invent new type codes for their programs. A single application may support several document types. For example, AppleWorks supports spreadsheet files (CWSS), word processor documents (CWWP), and drawings (CWGR). Similarly a graphics program will typically support many document types. Data files used by
an application may also have the same creator code as the application, even though they cannot be opened as documents. For example, the dictionary used by a spelling checker cannot be opened as a document but typically uses the same creator code as the spelling checker. Filename extensions Filename extensions (page 81) can substitute for type
and creator codes. For example, an AppleWorks word processor document is saved with an extension of .cwk. Also, if open cannot determine which application to use to open a file using the file's creator code, it reverts to using the file's filename extension. Finder Aliases Versus Symbolic Links A Macintosh Finder alias is similar to a symbolic link or a
Windows shortcut, but a Finder alias is not a symbolic link. Conversely, the Finder recognizes a symbolic link as though it were a Finder alias. A Finder alias points to a pathname, no matter where it is. Only if no target remains at the pathname does a Finder alias fall back to the file ID. Finder aliases work like resilient symbolic links as long as the
target file is still on the same volume. ACLs ACLs (Access Control Lists) are discussed on page 99. This section discusses how to enable and work with ACLs under Mac OS X. Enabling ACLs may cause errors caution Enabling ACLs on a filesystem prevents some programs that handle extended attributes (page 928) from working correctly. If a file has
a resource fork but no ACL, attempts to copy it may fail with an error message such as the following: $ pax -w -f arch.pax picture.jpg pax: Unable to open ._picture.jpg to read Some utilities, such as cp, are not affected by this problem. Extended Attributes 933 fsaclctl: Enabling and Disabling ACLs Under Mac OS X, the fsaclctl (file system ACL control)
utility can report on or change ACL support status on one or more volumes (filesystems). With the –p option, fsaclctl reports on the support status for the volume you specify as an argument to –p: $ fsaclctl -p / Access control lists are supported on /. Use the –a option without an argument to display the support status for all mounted volumes. You must
use the –p option to specify a volume when you change the status of a volume. The fsaclctl –e option enables ACL support on the specified volume(s); the –d option disables ACL support. You must work using root privileges to enable and disable ACL support. The following example first disables (–d) ACL support on the root filesystem (/) and then
verifies support is disabled. Next it enables (–e) support and again verifies the change. $ sudo fsaclctl -p / -d $ fsaclctl -p / Access control lists are not supported or currently disabled on /. $ sudo fsaclctl -p / -e $ fsaclctl -p / Access control lists are supported on /. chmod: Working with ACLs Under Mac OS X, you can use chmod (page 626) to create,
modify, and delete ACL rules. A chmod command used for this purpose has the following format: chmod option[# n] "who allow|deny permission-list" file-list where option is +a (add rule), –a (remove rule), or =a (change rule); n is an optional rule number; who is a username, user ID number, group name, or group ID number; permission-list is one or
more comma-separated file access permissions selected from read, write, append, and execute; and file-list is the list of files the rule is applied to. The quotation marks are required. In the first example, Sam adds a rule that grants Helen read and write access to the memo file: $ chmod +a "helen allow read,write" memo The chmod utility displays an
error message if you forget the quotation marks: $ chmod +a max deny write memo chmod: Invalid entry format -- expected allow or deny 934 Appendix D Mac OS X Notes ls –l An ls –l command displays a plus sign (+) following the permissions for a file that has an ACL (see also Figure 4-12 on page 94): $ ls -l memo -rw-r--r--+ 1 sam ls –le staff 1680
May 12 13:30 memo Under Mac OS X, the ls –e option displays ACL rules: $ ls -le memo -rw-r--r--+ 1 sam staff 1680 May 12 13:30 memo 0: user:helen allow read,write For each rule, the –e option displays from left to right the rule number followed by a colon, user: or group: followed by the name of the user or group the rule pertains to, allow or deny
depending on whether the rule grants or denies permissions, and a list of the permissions that are granted or denied. The kernel processes multiple rules in an ACL in rule number order. The kernel does not necessarily number rules in the order you enter them. In general, rules denying permissions come before rules granting permissions. You can
override the default order by assigning a number to a rule using the +a# n syntax. The following command bumps rule 0 from the previous example to rule 1 and replaces it with a rule that denies Max write access to memo: $ chmod +a# 0 "max deny write" memo $ ls -le memo -rw-r--r--+ 1 sam staff 1680 May 12 13:30 memo 0: user:max deny write
1: user:helen allow read,write There are two ways to remove access rules. First, you can specify a rule by using its number: $ chmod -a# 1 memo $ ls -le memo -rw-r--r--+ 1 sam staff 0: user:max deny write 1680 May 12 13:30 memo Second, you can specify a rule by using the string you used when you added it: $ chmod –a "max deny write" memo $ ls
-le memo -rw-r--r-- 1 sam staff 1680 May 12 13:30 memo After you remove the last rule, memo does not have an ACL. (There is no + in the line ls –le displays.) When you specify an ACL operation that chmod cannot complete, it displays an error message: $ chmod -a# 0 memo chmod: No ACL present Activating the META Key 935 In the next example,
Sam restores Helen's read and write permission to memo: $ chmod +a "helen allow read,write" memo $ ls -le memo -rw-r--r--+ 1 sam staff 1680 May 12 13:30 memo 0: user:helen allow read,write Sam then removes the write permissions he just gave Helen. When you remove one of several access permissions from a rule, the other permissions remain
unchanged: $ chmod -a "helen allow write" memo $ ls -le memo -rw-r--r--+ 1 sam staff 1680 May 12 13:30 memo 0: user:helen allow read The next example shows that chmod inserts rules in an ACL in a default order. Even though Sam added the allow rule before the deny rule, the allow rule appears first. The rule controlling permission for Helen,
which was added before either of the other rules, appears last. $ chmod +a "max allow read" memo $ chmod +a "max deny read" memo $ ls -le memo -rw-r--r--+ 1 sam staff 1680 May 12 13:30 memo 0: user:max deny read 1: user:max allow read 2: user:helen allow read You can replace a rule using the =a syntax. In the following example, Sam
changes rule 2 to grant Helen read and write permission to memo: $ chmod =a# 2 "helen allow read,write" memo $ ls -le memo -rw-r--r--+ 1 sam staff 1680 May 12 13:30 memo 0: user:max deny read 1: user:max allow read 2: user:helen allow read,write Activating the META Key Under Mac OS X, in the Terminal utility, you can make the OPTION (or
ALT) key work as the META key. From the Terminal utility File menu, select Window Settings to display the Terminal Inspector window. This window provides a drop-down menu of properties you can change. Select Keyboard, check the box labeled Use option key as meta key, and click Use Settings as Defaults. This procedure causes the OPTION key
(on Mac keyboards) or the ALT key (on PC keyboards) to function as the META key while you are using the Terminal utility. 936 Appendix D Mac OS X Notes Startup Files Both Mac OS and application documentation refer to startup files as configuration files or preference files. Many Mac OS X applications store startup files in the Library and
Library/Preferences subdirectories of a user's home directory, which are created when an account is set up. Most of these files do not have invisible filenames. Remote Logins By default Mac OS X does not allow remote logins. You can enable remote logins via ssh by enabling remote login in the Services tab of the Sharing pane of the Preferences
window. OS X does not support telnet logins. Many Utilities Do Not Respect Apple Human Interface Guidelines By default rm under Mac OS X does not act according to the Apple Human Interface Guidelines, which state that an operation should either be reversible or ask for confirmation. In general, Mac OS X command-line utilities do not ask
whether you are sure of what you are doing. Mac OS X Implementation of Linux Features Table D-3 explains how some Linux features are implemented under Mac OS X. Table D-3 Mac OS X implementation of Linux features Linux feature OS X implementation /bin/sh The /bin/sh is a copy of bash (/bin/bash); it is not a link to /bin/bash as it is on most
Linux systems. The original Bourne Shell does not exist under OS X. When you call bash using the command sh, bash tries to mimic the behavior of the original Bourne Shell as closely as possible. Core files By default OS X does not save core files. When core files are saved, they are kept in /cores, not in the working directory. Developer tools The
Developer Tools package, which includes the gcc compiler, is not installed by default. Mac OS X Implementation of Linux Features 937 Table D-3 Mac OS X implementation of Linux features (continued) Linux feature OS X implementation Development APIs Mac OS X uses three software development APIs: Cocoa, Carbon, and BSD UNIX. Dynamic
linker ld.so The OS X dynamic linker is dyld, not ld.so. ELF and a.out binary The primary binary format under OS X is Mach-O, not ELF or a.out. formats /etc/group Mac OS X uses OpenDirectory (page 926), not /etc/group, to store group information. /etc/passwd Mac OS X uses OpenDirectory (page 926), not /etc/passwd, to store user accounts.
Filesystem structure /etc/fstab Instead of filesystems being mounted according to settings in /etc/fstab, filesystems are automatically mounted in the /Volumes directory (page 928). finger By default, Mac OS X disables remote finger support. LD_LIBRARY_PATH The variable used to control the dynamic linker is DYLD_LIBRARY_PATH, not
LD_LIBRARY_PATH. System databases Some system databases, such as passwd and group, are stored by OpenDirectory (page 926), not in the /etc directory (page 92). You can work with OpenDirectory databases using the dscl utility (page 674). vi editor As with many Linux distributions, when you call the vi editor, Mac OS X 10.3 and later run vim
(page 149) because the file /usr/bin/vi is a link to /usr/bin/vim. This page intentionally left blank Glossary All entries marked with FOLDOC are based on definitions in the Free On-Line Dictionary of Computing (www.foldoc.org), Denis Howe, editor. Used with permission. GGlossary Glossary 939 940 Glossary 10.0.0.0 See private address space on page
972. 172.16.0.0 See private address space on page 972. 192.168.0.0 See private address space on page 972. 802.11 A family of specifications developed by IEEE for wireless LAN technology, including 802.11 (1–2 megabits per second), 802.11a (54 megabits per second), 802.11b (11 megabits per second), and 802.11g (54 megabits per second).
absolute pathname A pathname that starts with the root directory (represented by /). An absolute pathname locates a file without regard to the working directory. access In computer jargon, a verb meaning to use, read from, or write to. To access a file means to read from or write to the file. Access Control See ACL. List access permissions
Permission to read from, write to, or execute a file. If you have write access permission to a file (usually just called write permission), you can write to the file. Also access privilege. ACL Access Control List. A system that performs a function similar to file permissions but with much finer-grain control. active window On a desktop, the window that
receives the characters you type on the keyboard. Same as focus, desktop (page 955). address mask See subnet mask on page 981. alias A mechanism of a shell that enables you to define new commands. alphanumeric character One of the characters, either uppercase or lowercase, from A to Z and 0 to 9, inclusive. ambiguous file A reference to a file
that does not necessarily specify any one file but can be used to reference specify a group of files. The shell expands an ambiguous file reference into a list of filenames. Special characters represent single characters (?), strings of zero or more characters (*), and character classes ([]) within ambiguous file references. An ambiguous file reference is a
type of regular expression (page 975). angle bracket A left angle bracket (). The shell uses < to redirect a command's standard input to come from a file and > to redirect the standard output. The shell uses the characters > to append output to a file. animate When referring to a window action, means that the action is slowed down so the user can
view it. For example, when you minimize a window, it can disappear all at once (not animated) or it can slowly telescope into the panel so you can get a visual feel for what is happening (animated). Glossary 941 anti-aliasing Adding gray pixels at the edge of a diagonal line to get rid of the jagged appearance and thereby make the line look smoother.
Anti-aliasing sometimes makes type on a screen look better and sometimes worse; it works best on small and large fonts and is less effective on fonts from 8 to 15 points. See also subpixel hinting (page 981). API Application program interface. The interface (calling conventions) by which an application program accesses an operating system and other
services. An API is defined at the source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code.FOLDOC append To add something to the end of something else. To append text to a file means to add the text to the end of the file. The shell uses >> to
append a command's output to a file. applet A small program that runs within a larger program. Examples are Java applets that run in a browser and panel applets that run from a desktop panel. archive A file that contains a group of smaller, typically related, files. Also, to create such a file. The tar and cpio utilities can create and read archives.
argument A number, letter, filename, or another string that gives some information to a command and is passed to the command when it is called. A command-line argument is anything on a command line following the command name that is passed to the command. An option is a kind of argument. arithmetic expression A group of numbers,
operators, and parentheses that can be evaluated. When you evaluate an arithmetic expression, you end up with a number. The Bourne Again Shell uses the expr command to evaluate arithmetic expressions; the TC Shell uses @, and the Z Shell uses let. array An arrangement of elements (numbers or strings of characters) in one or more dimensions.
The Bourne Again, TC, and Z Shells and mawk/gawk can store and process arrays. ASCII American Standard Code for Information Interchange. A code that uses seven bits to represent both graphic (letters, numbers, and punctuation) and CONTROL characters. You can represent textual information, including program source code and English text, in
ASCII code. Because ASCII is a standard, it is frequently used when exchanging information between computers. See the file /usr/pub/ascii or give the command man ascii to see a list of ASCII codes. Extensions of the ASCII character set use eight bits. The seven-bit set is common; the eight-bit extensions are still coming into popular use. The eighth
bit is sometimes referred to as the metabit. ASCII terminal A textual terminal. Contrast with graphical display (page 956). ASP Application service provider. A company that provides applications over the Internet. 942 Glossary asynchronous event An event that does not occur regularly or synchronously with another event. Linux system signals are
asynchronous; they can occur at any time because they can be initiated by any number of nonregular events. attachment A file that is attached to, but is not part of, a piece of email. Attachments are frequently opened by programs (including your Internet browser) that are called by your mail program so you may not be aware that they are not an
integral part of an email message. authentication The verification of the identity of a person or process. In a communication system, authentication verifies that a message comes from its stated source. Methods of authentication on a Linux system include the /etc/passwd and /etc/shadow files, LDAP, Kerberos 5, and SMB authentication.FOLDOC
automatic mounting A way of demand mounting directories from remote hosts without having them hard configured into /etc/fstab. Also called automounting. avoided An object, such as a panel, that should not normally be covered by another object, such as a window. back door A security hole deliberately left in place by the designers or maintainers
of a system. The motivation for creating such holes is not always sinister; some operating systems, for example, come out of the box with privileged accounts intended for use by field service technicians or the vendor's maintenance programmers. Ken Thompson's 1983 Turing Award lecture to the ACM revealed the existence, in early UNIX versions, of

a back door that may be the most fiendishly clever security hack of all time. The C compiler contained code that would recognize when the login command was being recompiled and would insert some code recognizing a password chosen by Thompson, giving him entry to the system whether or not an account had been created for him. Normally such a back door could be removed by removing it from the source code for the compiler and recompiling the compiler. But to recompile the compiler, you have to use the compiler, so Thompson arranged that the compiler would recognize when it was compiling a version of itself. It would insert into the recompiled compiler the code to insert into the recompiled login the code to allow Thompson entry, and, of course, the code to recognize itself and do the whole thing again the next time around. Having done this once, he was then able to recompile the compiler from the original sources; the hack perpetuated itself invisibly, leaving the back door in place and active but with no trace in the sources. Sometimes called a wormhole. Also trap door.FOLDOC background process A process that is not run in the foreground. Also called a detached process, a background process is initiated by a command line that ends with an ampersand (&). You do not have to wait for a background process to run to completion before giving the shell additional commands. If you have job control, you can move background processes to the foreground, and vice versa. Glossary 943 basename The name of a file that, in contrast with a pathname, does not mention any of the directories containing the file (and therefore does not contain any slashes [/]). For example, hosts is the basename of /etc/hosts.FOLDOC baud The maximum information-carrying capacity of a communication channel in symbols (state transitions or level transitions) per second. It coincides with bits per second only for two-level modulation with no framing or stop bits. A symbol is a unique state of the communication channel, distinguishable by the receiver from all other possible states. For example, it may be one of two voltage levels on a wire for a direct digital connection, or it might be the phase or frequency of a carrier.FOLDOC Baud is often mistakenly used as a synonym for bits per second. baud rate Transmission speed. Usually used to measure terminal or modem speed. Common baud rates range from 110 to 38,400 baud. See baud. Berkeley UNIX One of the two major versions of the UNIX operating system. Berkeley UNIX was developed at the University of California at Berkeley by the Computer Systems Research Group and is often referred to as BSD (Berkeley Software Distribution). BIND Berkeley Internet Name Domain. An implementation of a DNS (page 951) server developed and distributed by the University of California at Berkeley. BIOS Basic Input/Output System. On PCs, EEPROM-based (page 953) system software that provides the lowest-level interface to peripheral devices and controls the first stage of the bootstrap (page 944) process, which loads the operating system. The BIOS can be stored in different types of memory. The memory must be nonvolatile so that it remembers the system settings even when the system is turned off. Also BIOS ROM. bit The smallest piece of information a computer can handle. A bit is a binary digit: either 1 or 0 (on or off). bit depth Same as color depth (page 947). bit-mapped display A graphical display device in which each pixel on the screen is controlled by an underlying representation of zeros and ones. blank character Either a SPACE or a TAB character, also called whitespace (page 987). In some contexts, NEWLINE s are considered blank characters. block A section of a disk or tape (usually 1,024 bytes long but shorter or longer on some systems) that is written at one time. block device A disk or tape drive. A block device stores information in blocks of characters. A block device is represented by a block device (block special) file. Contrast with character device (page 946). block number Disk and tape blocks are numbered so that Linux can keep track of the data on the device. 944 Glossary blocking factor The number of logical blocks that make up a physical block on a tape or disk. When you write 1K logical blocks to a tape with a physical block size of 30K, the blocking factor is 30. Boolean The type of an expression with two possible values: true and false. Also, a variable of Boolean type or a function with Boolean arguments or result. The most common Boolean functions are AND, OR, and NOT.FOLDOC boot See bootstrap. boot loader A very small program that takes its place in the bootstrap process that brings a computer from off or reset to a fully functional state. bootstrap Derived from “Pull oneself up by one’s own bootstraps,” the incremental process of loading an operating system kernel into memory and starting it running without any outside assistance. Frequently shortened to boot. Bourne Again Shell bash. GNU’s command interpreter for UNIX, bash is a POSIX-compliant shell with Bourne Shell sh. This UNIX command processor was developed by Steve Bourne at AT&T Bell full Bourne Shell syntax and some C Shell commands built in. The Bourne Again Shell supports emacs-style command-line editing, job control, functions, and online help.FOLDOC Laboratories. brace A left brace ({) and a right brace (}). Braces have special meanings to the shell. bracket A square bracket (page 980) or an angle bracket (page 940). branch In a tree structure, a branch connects nodes, leaves, and the root. The Linux filesystem hierarchy is often conceptualized as an upside-down tree. The branches connect files and directories. In a source code control system, such as SCCS or RCS, a branch occurs when a revision is made to a file and is not included in subsequent revisions to the file. bridge Typically a two-port device originally used for extending networks at layer 2 (data link) of the Internet Protocol model. broadcast A transmission to multiple, unspecified recipients. On Ethernet a broadcast packet is a special type of multicast packet that has a special address indicating that all devices that receive it should process it. Broadcast traffic exists at several layers of the network stack, including Ethernet and IP. Broadcast traffic has one source but indeterminate destinations (all hosts on the local network). broadcast address The last address on a subnet (usually 255), reserved as shorthand to mean all hosts. broadcast network A type of network, such as Ethernet, in which any system can transmit information at any time, and all systems receive every message. Glossary 945 BSD See Berkeley UNIX on page 943. buffer An area of memory that stores data until it can be used. When you write information to a file on a disk, Linux stores the information in a disk buffer until there is enough to write to the disk or until the disk is ready to receive the information. bug An unwanted and unintended program property, especially one that causes the program to malfunction.FOLDOC builtin (command) A command that is built into a shell. Each of the three major shells—the Bourne Again, TC, and Z Shells—has its own set of builtins. byte A component in the machine data hierarchy, usually larger than a bit and smaller than a word; now most often eight bits and the smallest addressable unit of storage. A byte typically holds one character.FOLDOC C programming language A modern systems language that has high-level features for efficient, modular programming as well as lower-level features that make it suitable for use as a systems programming language. It is machine independent so that carefully written C programs can be easily transported to run on different machines. Most of the Linux operating system is written in C, and Linux provides an ideal environment for programming in C. C Shell csh. The C Shell command processor was developed by Bill Joy for BSD UNIX. It was named for the C programming language because its programming constructs are similar to those of C. See shell on page 978. cable modem A type of modem that allows you to access the Internet by using your cable television connection. cache Holding recently accessed data, a small, fast memory designed to speed up subsequent access to the same data. Most often applied to processor-memory access but also used for a local copy of data accessible over a network, from a hard disk, and so on.FOLDOC calling environment A list of variables and their values that is made available to a called program. Refer to “Executing a Command” on page 308. cascading stylesheet See CSS on page 949. cascading windows An arrangement of windows such that they overlap, generally with at least part of the title bar visible. Opposite of tiled windows (page 983). case sensitive Able to distinguish between uppercase and lowercase characters. Unless you set the ignorecase parameter, vim performs case-sensitive searches. The grep utility performs case-sensitive searches unless you use the –i option. catenate To join sequentially, or end to end. The Linux cat utility catenates files: It displays them one after the other. Also concatenate. 946 Glossary chain loading The technique used by a boot loader to load unsupported operating systems. Used for loading such operating systems as DOS or Windows, it works by loading another boot loader. characterbased A program, utility, or interface that works only with ASCII (page 941) characters. This set of characters includes some simple graphics, such as lines and corners, and can display colored characters. It cannot display true graphics. Contrast with GUI (page 956). characterA terminal that displays only characters and very limited graphics. See character-based. based terminal character class In a regular expression, a group of characters that defines which characters can occupy a single character position. A character-class definition is usually surrounded by square brackets. The character class defined by [abcr] represents a character position that can be occupied by a, b, c, or r. Also list operator. In POSIX, used to refer to sets of characters with a common characteristic, denoted by the notation [:class:]; for example, [:upper:] denotes the set of uppercase letters. This book uses the term character class as explained under “Brackets” on page 889. character device A terminal, printer, or modem. A character device stores or displays characters one at a time. A character device is represented by a character device (character special) file. Contrast with block device (page 943). check box A GUI widget, usually the outline of a square box with an adjacent caption, that a user can click to display or remove a tick (page 983). When the box holds a tick, the option described by the caption is on or true. Also tick box. checksum A computed value that depends on the contents of a block of data and is transmitted or stored along with the data to detect corruption of the data. The receiving system recomputes the checksum based on the received data and compares this value with the one sent with the data. If the two values are the same, the receiver has some confidence that the data was received correctly. The checksum may be 8, 16, or 32 bits, or some other size. It is computed by summing the bytes or words of the data block, ignoring overflow. The checksum may be negated so that the total of the data words plus the checksum is zero. Internet packets use a 32-bit checksum.FOLDOC child process A process that is created by another process, the parent process. Every process is a child process except for the first process, which is started when Linux begins execution. When you run a command from the shell, the shell spawns a child process to run the command. See process on page 972. CIDR Classless Inter-Domain Routing. A scheme that allocates blocks of Internet addresses in a way that allows summarization into a smaller number of routing table entries. A CIDR block is a block of Internet addresses assigned to an ISP by the Internic.FOLDOC Glossary 947 CIFS Common Internet File System. An Internet filesystem protocol based on SMB (page 978). CIFS runs on top of TCP/IP, uses DNS, and is optimized to support slower dial-up Internet connections. SMB and CIFS are used interchangeably.FOLDOC CIPE Crypto IP Encapsulation (page 953). This protocol (page 973) tunnels (page 984) IP packets within encrypted UDP (page 984) packets, is lightweight and simple, and works over dynamic addresses, NAT (page 967), and SOCKS (page 979) proxies (page 973). cipher (cypher) A cryptographic system that uses a key to transpose/substitute characters within a message, the key itself, or the message. ciphertext Text that is encrypted. Contrast with plaintext (page 971). Classless Inter-Domain Routing See CIDR on page 946. cleartext Text that is not encrypted. Also plaintext. Contrast with ciphertext. CLI Command-line interface. See also character-based (page 946). Also textual interface. client A computer or program that requests one or more services from a server. CODEC Coder/decoder or compressor/decompressor. A hardware and/or software technology that codes and decodes data. MPEG is a popular CODEC for computer video. color depth The number of bits used to generate a pixel—usually 8, 16, 24, or 32. The color depth is directly related to the number of colors that can be generated. The number of colors that can be generated is 2 raised to the color-depth power. Thus a 24-bit video adapter can generate about 16.7 million colors. color quality See color depth. combo box A combination of a drop-down list (page 952) and text box (page 983). You can enter text in a combo box. Or, you can click a combo box, cause it to expand and display a static list of selections for you to choose from. command What you give the shell in response to a prompt. When you give the shell a command, it executes a utility, another program, a builtin command, or a shell script. Utilities are often referred to as commands. When you are using an interactive utility, such as vim or mail, you use commands that are appropriate to that utility. command line A line containing instructions and arguments that executes a command. This term usually refers to a line that you enter in response to a shell prompt on a characterbased terminal or terminal emulator. command substitution Replacing a command with its output. The shells perform command substitution when you enclose a command between $(and) or between a pair of back ticks (‘‘), also called grave accent marks. 948 Glossary component architecture A notion in object-oriented programming where “components” of a program are completely generic. Instead of having a specialized set of methods and fields, they have generic methods through which the component can advertise the functionality it supports to the system into which it is loaded. This strategy enables completely dynamic loading of objects. JavaBeans is an example of a component architecture.FOLDOC concatenate See catenate on page 945. condition code See exit status on page 953. connectionoriented protocol A type of transport layer data communication service that allows a host to send data in a continuous stream to another host. The transport service guarantees that all data will be delivered to the other end in the same order as sent and without duplication. Communication proceeds through three well-defined phases: connection establishment, data transfer, and connection release. The most common example is TCP (page 982). Also called connection-based protocol and stream-oriented protocol. Contrast with connectionless protocol and datagram (page 950).FOLDOC connectionless The data communication method in which communication occurs between hosts protocol with no previous setup. Packets sent between two hosts may take different routes. There is no guarantee that packets will arrive as transmitted or even that they will arrive at the destination at all. UDP (page 984) is a connectionless protocol. Also called packet switching. Contrast with circuit switching and connection-oriented protocol.FOLDOC console The main system terminal, usually directly connected to the computer and the one that receives system error messages. Also system console and console terminal. console terminal See console. control character A character that is not a graphic character, such as a letter, number, or punctuation mark. Such characters are called control characters because they frequently act to control a peripheral device. RETURN and FORMFEED are control characters that control a terminal or printer. The word CONTROL is shown in this book in THIS FONT because it is a key that appears on most terminal keyboards. Control characters are represented by ASCII codes less than 32 (decimal). See also nonprinting character on page 969. control structure A statement used to change the order of execution of commands in a shell script or other program. Each shell provides control structures (for example, if and while) as well as other commands that alter the order of execution (for example, exec). Also control flow commands. cookie Data stored on a client system by a server. The client system browser sends the cookie back to the server each time it accesses that server. For example, a catalog shopping service may store a cookie on your system when you place your first Glossary 949 order. When you return to the site, it knows who you are and can supply your name and address for subsequent orders. You may consider cookies to be an invasion of privacy. CPU Central processing unit. The part of a computer that controls all the other parts. The CPU includes the control unit and the arithmetic and logic unit (ALU). The control unit fetches instructions from memory and decodes them to produce signals that control the other parts of the computer. These signals can cause data to be transferred between memory and ALU or peripherals to perform input or output. A CPU that is housed on a single chip is called a microprocessor. Also processor and central processor. cracker An individual who attempts to gain unauthorized access to a computer system. These individuals are often malicious and have many means at their disposal for breaking into a system. Contrast with hacker (page 956).FOLDOC crash The system suddenly and unexpectedly stops or fails. Derived from the action of the hard disk heads on the surface of the disk when the air gap between the two collapses. cryptography The practice and study of encryption and decryption—encoding data so that only a specific individual or machine can decode it. A system for encrypting and decrypting data is a cryptosystem. Such systems usually rely on an algorithm for combining the original data (plaintext) with one or more keys—numbers or strings of characters known only to the sender and/or recipient. The resulting output is called ciphertext (page 947). The security of a cryptosystem usually depends on the secrecy of keys rather than on the supposed secrecy of an algorithm. Because a strong cryptosystem has a large range of keys, it is not possible to try all of them. Ciphertext appears random to standard statistical tests and resists known methods for breaking codes.FOLDOC .cshrc file In your home directory, a file that the TC Shell executes each time you invoke a new TC Shell. You can use this file to establish variables and aliases. CSS Cascading stylesheet. Describes how documents are presented on screen and in print. Attaching a stylesheet to a structured document can affect the way it looks without adding new HTML (or other) tags and without giving up device independence. Also stylesheet. current (process, line, character, directory, event, etc.) The item that is immediately available, working, or being used. The current process is the program you are running, the current line or character is the one the cursor is on, and the current directory is the working directory. cursor A small lighted rectangle, underscore, or vertical bar that appears on a terminal screen and indicates where the next character will appear. Differs from the mouse pointer (page 966). 950 Glossary daemon A program that is not invoked explicitly but lies dormant, waiting for some condition(s) to occur. The perpetrator of the condition need not be aware that a daemon is lurking (although often a program will commit an action only because it knows that it will implicitly invoke a daemon). From the mythological meaning, later rationalized as the acronym Disk And Execution MONitor..FOLDOC data structure A particular format for storing, organizing, working with, and retrieving data. Frequently, data structures are designed to work with specific algorithms that facilitate these tasks. Common data structures include trees, files, records, tables, arrays, etc. datagram A self-contained, independent entity of data carrying sufficient information to be routed from the source to the destination computer without reliance on earlier exchanges between this source and destination computer and the transporting network. UDP (page 984) uses datagrams; IP (page 960) uses packets (page 970). Packets are indivisible at the network layer; datagrams are not.FOLDOC See also frame (page 955). dataless A computer, usually a workstation, that uses a local disk to boot a copy of the operating system and access system files but does not use a local disk to store user files. dbm A standard, simple database manager. Implemented as gdbm (GNU database manager), it uses hashes to speed searching. The most common versions of the dbm database are dbm, ndbm, and gdbm. DDoS attack Distributed denial of service attack. A DoS attack (page 952) from many systems that do not belong to the perpetrator of the attack. debug To correct a program by removing its bugs (that is, errors). default Something that is selected without being explicitly specified. For example, when used without an argument, ls displays a list of the files in the working directory by default. delta A set of changes made to a file that has been encoded by the Source Code Control System (SCCS). denial of service See DoS attack on page 952. dereference When speaking of symbolic links, follow the link rather than working with the reference to the link. For example, the –L or ––dereference option causes ls to list the entry a symbolic link points to, not the symbolic link (the reference) itself. desktop A collection of windows, toolbars, icons, and buttons, some or all of which appear on your display. A desktop comprises one or more workspaces (page 988). desktop manager An icon- and menu-based user interface to system services that allows you to run applications and use the filesystem without using the system’s command-line interface. Glossary 951 detached process See background process on page 942. device A disk drive, printer, terminal, plotter, or other input/output unit that can be attached to the computer. Short for peripheral device. device driver Part of the Linux kernel that controls a device, such as a terminal, disk drive, or printer. device file A file that represents a device. Also special file. device filename The pathname of a device file. All Linux systems have two kinds of device files: block and character device files. Linux also has FIFOs (named pipes) and sockets. Device files are traditionally located in the /dev directory. device number See major device number (page 964) and minor device number (page 966). DHCP Dynamic Host Configuration Protocol. A protocol that dynamically allocates IP addresses to computers on a LAN.FOLDOC dialog box In a GUI, a special window, usually without a titlebar, that displays information. Some dialog boxes accept a response from the user directory Short for directory file. A file that contains a list of other files. directory hierarchy A directory, called the root of the directory hierarchy, and all the directory and ordinary files below it (its children). directory service A structured repository of information on people and resources within an organization, facilitating management and communication.FOLDOC disk partition See partition on page 970. diskless A computer, usually a workstation, that has no disk and must contact another computer (a server) to boot a copy of the operating system and access the necessary system files. distributed computing A style of computing in which tasks or services are performed by a network of cooperating systems, some of which may be specialized. DMZ Demilitarized zone. A host or small network that is a neutral zone between a LAN and the Internet. It can serve Web pages and other data to the Internet and allow local systems access to the Internet while preventing LAN access to unauthorized Internet users. Even if a DMZ is compromised, it holds no data that is private and none that cannot be easily reproduced. DNS Domain Name Service. A distributed service that manages the correspondence of full hostnames (those that include a domain name) to IP addresses and other system characteristics. DNS domain name See domain name on page 952. 952 Glossary document object model See DOM. DOM Document Object Model. A platform-/language-independent interface that enables a program to update the content, structure, and style of a document dynamically. The changes can then be made part of the displayed document. Go to www.w3.org/DOM for more information. domain name A name associated with an organization, or part of an organization, to help identify systems uniquely. Technically, the part of the FQDN (page 955) to the right of the leftmost period. Domain names are assigned hierarchically. The domain berkeley.edu refers to the University of California at Berkeley, for example; it is part of the top-level edu (education) domain. Also DNS domain name. Different than NIS domain name (page 968). Domain Name See DNS. Service door An evolving filesystem-based RPC (page 976) mechanism. DoS attack Denial of service attack. An attack that attempts to make the target host or network unusable by flooding it with spurious traffic. DPMS Display Power Management Signaling. A standard that can extend the life of CRT monitors and conserve energy. DPMS supports four modes for a monitor: Normal, Standby (power supply on, monitor ready to come to display images almost instantly), Suspend (power supply off, monitor takes up to ten seconds to display an image), and Off. drag The motion part of drag-and-drop. drag-and-drop To move an object from one position or application to another within a GUI. To drag an object, the user clicks a mouse button (typically the left one) while the mouse pointer hovers (page 958) over the object. Then, without releasing the mouse button, the user drags the object, which stays attached to the mouse pointer, to a different location. The user can then drop the object at the new location by releasing the mouse button. drop-down list A widget (page 987) that displays a static list for a user to choose from. When the list is not active, it appears as text in a box, displaying the single selected entry. When a user clicks the box, a list appears; the user can move the mouse cursor to select an entry from the list. Different from a list box (page 963). druid In role-playing games, a character that represents a magical user. Red Hat uses the term druid at the ends of names of programs that guide you through a task-driven chain of steps. Other operating systems call these types of programs wizards. DSA Digital Signature Algorithm. A public key cipher used to generate digital signatures. Glossary 953 DSL Digital Subscriber Line/Loop. Provides high-speed digital communication over a specialized, conditioned telephone line. See also xDSL (page 988). Dynamic Host See DHCP on page 951. Configuration Protocol editor A utility, such as vim or emacs, that creates and modifies text files. EEPROM Electrically erasable, programmable, readonly memory. A PROM (page 972) that can be written to. effective user ID The user ID that a process appears to have; usually the same as the user ID. For example, while you are running a setuid program, the effective user ID of the process running the program is that of the owner of the program. element One thing; usually a basic part of a group of things. An element of a numeric array is one of the numbers stored in the array. emoticon See smiley on page 978. encapsulation See tunneling on page 984. environment See calling environment on page 945. EOF End of file. EPROM Erasable programmable readonly memory. A PROM (page 972) that can be written to by applying a higher than normal voltage. escape See quote on page 973. Ethernet A type of LAN (page 962) capable of transfer rates as high as 1,000 megabits per second. event An occurrence, or happening, of significance to a task or program—for example, the completion of an asynchronous input/output operation, such as a keypress or mouse click.FOLDOC exabyte 260 bytes or about 1018 bytes. See also large number (page 962). exit status The status returned by a process; either successful (usually 0) or unsuccessful (usually 1). exploit A security hole or an instance of taking advantage of a security hole.FOLDOC expression See logical expression (page 964) and arithmetic expression (page 941). extranet A network extension for a subset of users (such as students at a particular school or engineers working for the same company). An extranet limits access to private information even though it travels on the public Internet. 954 Glossary failsafe session A session that allows you to log in on a minimal desktop in case your standard login does not work well enough to allow you to log in to fix a login problem. FDDI Fiber Distributed Data Interface. A type of LAN (page 962) designed to transport data at the rate of 100 million bits per second over fiberoptic cable. file A collection of related information referred to with a filename and frequently stored on a disk. Text files typically contain memos, reports, messages, program source code, lists, or manuscripts. Binary or executable files contain utilities or programs that you can run. Refer to “Directory Files and Ordinary Files” on page 78. filename The name of a file. A filename refers to a file. filename completion Automatic completion of a filename after you specify a unique prefix. filename extension The part of a filename following a period. filename generation What occurs when the shell expands ambiguous file references. See ambiguous file reference on page 940. filesystem A data structure (page 950) that usually resides on part of a disk. All Linux systems have a root filesystem, and many have other filesystems. Each filesystem is composed of some number of blocks, depending on the size of the disk partition that has been assigned to the filesystem. Each filesystem has a control block, named the superblock, that contains information about the filesystem. The other blocks in a filesystem are inodes, which contain control information about individual files, and data blocks, which contain the information in the files. filling A variant of maximizing in which window edges are pushed out as far as they can go without overlapping another window. filter A command that can take its input from standard input and send its output to standard output. A filter transforms the input stream of data and sends it to standard output. A pipe usually connects a filter’s input to standard output of one command, and a second pipe connects the filter’s output to standard input of another command. The grep and sort utilities are commonly used as filters. firewall A device for policy-based traffic management used to keep a network secure. A firewall can be implemented in a single router that filters out unwanted packets, or it can rely on a combination of routers, proxy servers, and other devices. Firewalls are widely used to give users access to the Internet in a secure fashion and to separate a company’s public WWW server from its internal network. They are also employed to keep internal network segments more secure. Recently the term has come to be defined more loosely to include a simple packet filter running on an endpoint machine. See also proxy server on page 973. Glossary 955 firmware Software built into a computer, often in ROM (page 976). May be used as part of the bootstrap (page 944) procedure. focus, desktop On a desktop, the window that is active. The window with the desktop focus receives the characters you type on the keyboard. Same as active window (page 940). footer The part of a format that goes at the bottom (or foot) of a page. Contrast with header (page 957). foreground process When you run a command in the foreground, the shell waits for the command to finish before giving you another prompt. You must wait for a foreground process to run to completion before you can give the shell another command. If you have job control, you can move background processes to the foreground, and vice versa. See job control on page 961. Contrast with background process (page 942). fork To create a process. When one process creates another process, it forks a process. Also spawn. FQDN Fully qualified domain name. The full name of a system, consisting of its hostname and its domain name, including the top-level domain. Technically the name that gethostbyname(2) returns for the host named by gethostname(2). For example, speedy is a hostname and speedy.example.com is an FQDN. An FQDN is sufficient to determine a unique Internet address for a machine on the Internet.FOLDOC frame A data link layer packet that contains, in addition to data, the header and trailer information required by the physical medium. Network layer packets are encapsulated to become frames.FOLDOC See also datagram (page 950) and packet (page 970). free list In a filesystem, the list of blocks that are available for use. Information about the free list is kept in the superblock of the filesystem. free space The portion of a hard disk that is not within a partition. A new hard disk has no partitions and contains all free space. full duplex The ability to receive and transmit data simultaneously. A network switch (page 968) is typically a full-duplex device. Contrast with half-duplex (page 956). fully qualified domain name See FQDN. function See shell function on page 978. gateway A generic term for a computer or a special device connected to more than one dissimilar type of network to pass data between them. Unlike a router, a gateway often must convert the information into a different format before passing it on. The historical usage of gateway to designate a router is deprecated. GCOS See GECOS on page 956. 956 Glossary GECOS General Electric Comprehensive Operating System. For historical reasons, the user information field in the /etc/passwd file is called the GECOS field. Also GCOS. gibibyte Giga binary byte. A unit of storage equal to 230 bytes = 1,073,741,824 bytes = 1024 mebibytes (page 965). Abbreviated as GiB. Contrast with gigabyte. gigabyte A unit of storage equal to 109 bytes. Sometimes used in place of gibibyte. Abbreviated as GB. See also large number on page 962. glyph A symbol that communicates a specific piece of information nonverbally. A smiley (page 978) is a glyph. GMT Greenwich Mean Time. See UTC on page 986. graphical display A bitmapped monitor that can display graphical images. Contrast with ASCII terminal (page 941). graphical user interface See GUI. group (of users) A collection of users. Groups are used as a basis for determining file access permissions. If you are not the owner of a file and you belong to the group the file is assigned to, you are subject to the group access permissions for the file. A user can simultaneously belong to several groups. group (of windows) A way to identify similar windows so they can be displayed and acted on similarly. Typically windows started by a given application belong to the same group. group ID A unique number that identifies a set of users. It is stored in the password and group databases (/etc/passwd and /etc/group files or their NIS equivalents). The group database associates group IDs with group names. Also GID. GUI Graphical user interface. A GUI provides a way to interact with a computer system by choosing items from menus or manipulating pictures drawn on a display screen instead of by typing command lines. Under Linux, the X Window System provides a graphical display and mouse/keyboard input. GNOME and KDE are two popular desktop managers that run under X. Contrast with character-based (page 946). hacker A person who enjoys exploring the details of programmable systems and learning how to stretch their capabilities, as opposed to users, who prefer to learn only the minimum necessary. One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming.FOLDOC Contrast with cracker (page 949). half-duplex A half-duplex device can only receive or transmit at a given moment; it cannot do both. A hub (page 958) is typically a half-duplex device. Contrast with full duplex (page 955). hard link A directory entry that contains the filename and inode number for a file. The inode number identifies the location of control information for the file on the disk, which Glossary 957 in turn identifies the location of the file’s contents on the disk. Every file has at least one hard link, which locates the file in a directory. When you remove the last hard link to a file, you can no longer access the file. See link (page 963) and symbolic link (page 982). hash A string that is generated from another string. See one-way hash function on page 969. When used for security, a hash can prove, almost to a certainty, that a message has not been tampered with during transmission: The sender generates a hash of a message, encrypts the message and hash, and sends the encrypted message and hash to the recipient. The recipient decrypts the message and hash, generates a second hash from the message, and compares the hash that the sender generated to the new hash. When they are the same, the message has probably not been tampered with. Hashed versions of passwords can be used to authenticate users. A hash can also be used to create an index called a hash table. Also hash value. hash table An index created from hashes of the items to be indexed. The hash function makes it highly unlikely that two items will create the same hash. To look up an item in the index, create a hash of the item and search for the hash. Because the hash is typically shorter than the item, the search is more efficient. header When you are formatting a document, the header goes at the top, or head, of a page. In electronic mail the header identifies who sent the message, when it was sent, what the subject of the message is, and so forth. Here document A shell script that takes its input from the file that contains the script. hesiod The nameserver of project Athena. Hesiod is a name service library that is derived from BIND (page 943) and leverages a DNS infrastructure. heterogeneous Consisting of different parts. A heterogeneous network includes systems produced by different manufacturers and/or running different operating systems. hexadecimal number A base 16 number. Hexadecimal (or hex) numbers are composed of the hexadecimal digits 0–9 and A–F. See Table G-1, next page. hidden filename A filename that starts with a period. These filenames are called hidden because the ls utility does not normally list them. Use the –a option of ls to list all files, including those with hidden filenames. The shell does not expand a leading asterisk (*) in an ambiguous file reference to match files with hidden filenames. Also hidden file, invisible file. hierarchy An organization with a few things, or thing—one at the top—and with several things below each other thing. An inverted tree structure. Examples in computing include a file tree where each directory may contain files or other directories, a hierarchical network, and a class hierarchy in object-oriented programming.FOLDOC Refer to “The Hierarchical Filesystem” on page 78. 958 Glossary Table G-1 Decimal, octal, and hexadecimal numbers Decimal Octal Hex 1 1 1 2 2 3 Decimal Octal Hex 17 21 11 2 18 22 12 3 3 19 23 13 4 4 4 20 24 14 5 5 5 21 25 15 6 6 6 31 37 1F 7 7 7 32 40 20 8 10 8 33 41 21 9 11 9 64 100 40 10 12 A 96 140 60 11 13 B 100 144 64 12 14 C 128 200 80 13 15 D 254 376 FE 14 16 E 255 377 FF 15 17 F 256 400 100 16 20 10 257 401 101 history A shell mechanism that enables you to modify and reexecute recent commands. home directory The directory that is the working directory when you first log in. The pathname of this directory is stored in the HOME shell variable. hover To leave the mouse pointer stationary for a moment over an object. In many cases hovering displays a tooltip (page 984). HTML Hypertext Markup Language. A hypertext document format used on the World Wide Web. Tags, which are embedded in the text, consist of a less than sign (). Matched pairs of directives, such as and , delimit text that is to appear in a special place or style.FOLDOC For more information on HTML, go to www.htmlhelp.com/faq/ html/all.html. HTTP Hypertext Transfer Protocol. The client/server TCP/IP protocol used on the World Wide Web for the exchange of HTML documents. hub A multiport repeater. A hub rebroadcasts all packets it receives on all ports. This term is frequently used to refer to small hubs and switches, regardless of the device’s intelligence. It is a generic term for a layer 2 shared-media networking device. Today the term hub is sometimes used to refer to small intelligent devices, although that was not its original meaning. Contrast with network switch (page 968). Glossary 959 hypertext A collection of documents/nodes containing (usually highlighted or underlined) cross-references or links, which, with the aid of an interactive browser program, allow the reader to move easily from one document to another.FOLDOC Hypertext Markup Language See HTML. Hypertext Transfer Protocol See HTTP. i/o device Input/output device. See device on page 951. IANA Internet Assigned Numbers Authority. A group that maintains a database of all permanent, registered system services (www.iana.org). ICMP Internet Control Message Protocol. A type of network packet that carries only messages, no data. icon In a GUI, a small picture representing a file, directory, action, program, and so on. When you click an icon, an action, such as opening a window and starting a program or displaying a directory or Web site, takes place. From miniature religious statues.FOLDOC iconify The process of changing a window into an icon. Contrast with restore (page 975). ignored window A state in which a window has no decoration and therefore no buttons or titlebar to control it with. indentation See indention. indention The blank space between the margin and the beginning of a line that is set in from the margin. inode A data structure (page 950) that contains information about a file. An inode for a file contains the file’s length, the times the file was last accessed and modified, the time the inode was last modified, owner and group IDs, access privileges, number of links, and pointers to the data blocks that contain the file itself. Each directory entry associates a filename with an inode. Although a single file may have several filenames (one for each link), it has only one inode. input Information that is fed to a program from a terminal or other file. See standard input on page 980. installation A computer at a specific location. Some aspects of the Linux system are installation dependent. Also site. interactive A program that allows ongoing dialog with the user. When you give commands in response to shell prompts, you are using the shell interactively. Also, when you give commands to utilities, such as vim and mail, you are using the utilities interactively. 960 Glossary interface The meeting point of two subsystems. When two programs work together, their interface includes every aspect of either program that the other deals with. The user interface (page 985) of a program includes every program aspect the user comes into contact with: the syntax and semantics involved in invoking the program, the input and output of the program, and its error and informational messages. The shell and each of the utilities and built-in commands have a user interface. International See ISO on page 961. Organization for Standardization internet A large network that encompasses other, smaller networks. Internet The largest internet in the world. The Internet (uppercase “I”) is a multilevel hierarchy composed of backbone networks (ARPANET, NSFNET, MILNET, and others), midlevel networks, and stub networks. These include commercial (.com or .co), university (.ac or .edu), research (.org or .net), and military (.mil) networks and span many different physical networks around the world with various protocols, including the Internet Protocol (IP). Outside the United States, country code domains are popular (.us, .es, .mx, .de, and so forth), although you will see them used within the United States as well. Internet Protocol See IP. Internet service provider See ISP. intranet An inhouse network designed to serve a group of people such as a corporation or school. The general public on the Internet does not have access to the intranet. invisible file See hidden filename on page 957. IP Internet Protocol. The network layer for TCP/IP. IP is a best-effort, packetswitching, connectionless protocol (page 948) that provides packet routing, fragmentation, and reassembly through the data link layer. IPv4 is slowly giving way to IPv6.FOLDOC IP address Internet Protocol address. A four-part address associated with a particular network connection for a system using the Internet Protocol (IP). A system that is attached to multiple networks that use the IP will have a different IP address for each network interface. IP multicast See multicast on page 967. Glossary 961 IP spoofing A technique used to gain unauthorized access to a computer. The would-be intruder sends messages to the target machine. These messages contain an IP address indicating that the messages are coming from a trusted host. The target machine responds to the messages, giving the intruder (privileged) access to the target. IPC Interprocess communication. A method to communicate specific information between programs. IPv4 IP version 4. See IP and IPv6. IPv6 IP version 6. The next generation of Internet Protocol, which provides a much larger address space (2128 bits versus 232 bits for IPv4) that is designed to accommodate the rapidly growing number of Internet addressable devices. IPv6 also has built-in autoconfiguration, enhanced security, better multicast support, and many other features. ISDN Integrated Services Digital Network. A set of communications standards that allows a single pair of digital or standard telephone wires to carry voice, data, and video at a rate of 64 kilobits per second. ISO International Organization for Standardization. A voluntary, nontreaty organization founded in 1946. It is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of 89 countries, including the American National Standards Institute.FOLDOC ISO9660 The ISO standard defining a filesystem for CD-ROMs. ISP Internet service provider. Provides Internet access to its customers. job control A facility that enables you to move commands from the foreground to the background and vice versa. Job control enables you to stop commands temporarily. journaling filesystem A filesystem that maintains a noncached log file, or journal, which records all transactions involving the filesystem. When a transaction is complete, it is marked as complete in the log file. The log file results in greatly reduced time spent recovering a filesystem after a crash, making it particularly valuable in systems where high availability is an issue. JPEG Joint Photographic Experts Group. This committee designed the standard imagecompression algorithm. JPEG is intended for compressing either full-color or gray-scale digital images of natural, real-world scenes and does not work as well on nonrealistic images, such as cartoons or line drawings. Filename extensions: .jpg, .jpeg.FOLDOC justify To expand a line of type in the process of formatting text. A justified line has even margins. A line is justified by increasing the space between words and sometimes between letters on the line. 962 Glossary Kerberos An MIT-developed security system that authenticates users and machines. It does not provide authorization to services or databases; it establishes identity at logon, which is used throughout the session. Once you are authenticated, you can open as many terminals, windows, services, or other network accesses as you like until your session expires. kernel The part of the operating system that allocates machine resources, including memory, disk space, and CPU (page 949) cycles, to all other programs that run on a computer. The kernel includes the low-level hardware interfaces (drivers) and manages processes (page 972), the means by which Linux executes programs. The kernel is the part of the Linux system that Linus Torvalds originally wrote (see the beginning of Chapter 1). kernelspace The part of memory (RAM) where the kernel resides. Code running in kernelspace has full access to hardware and all other processes in memory. See the KernelAnalysis-HOWTO. key binding A keyboard key is said to be bound to the action that results from pressing it. Typically keys are bound to the letters that appear on the keycaps: When you press A, an A appears on the screen. Key binding usually refers to what happens when you press a combination of keys, one of which is CONTROL, ALT, META, or SHIFT, or when you press a series of keys, the first of which is typically ESCAPE. keyboard A hardware input device consisting of a number of mechanical buttons (keys) that the user presses to input characters to a computer. By default a keyboard is connected to standard input of a shell.FOLDOC kilo- In the binary system, the prefix kilo- multiplies by 210 (i.e., 1,024). Kilobit and kilobyte are common uses of this prefix. Abbreviated as k. Korn Shell ksh. A command processor, developed by David Korn at AT&T Bell Laboratories, that is compatible with the Bourne Shell but includes many extensions. See also shell on page 978. LAN Local area network. A network that connects computers within a localized area (such as a single site, building, or department). large number Visit mathworld.wolfram.com/LargeNumber.html for a comprehensive list. LDAP Lightweight Directory Access Protocol. A simple protocol for accessing online directory services. LDAP is a lightweight alternative to the X.500 Directory Access Protocol (DAP). It can be used to access information about people, system users, network devices, email directories, and systems. In some cases, it can be used as an alternative for services such as NIS. Given a name, many mail clients can use LDAP to discover the corresponding email address. See directory service on page 951. leaf In a tree structure, the end of a branch that cannot support other branches. When the Linux filesystem hierarchy is conceptualized as a tree, files that are not directories are leaves. See node on page 969. Glossary 963 least privilege, concept of Mistakes made by a user working with root privileges can be much more devastating than those made by an ordinary user. When you are working on the computer, especially when you are working as the system administrator, always perform any task using the least privilege possible. If you can perform a task logged in as an ordinary user, do so. If you must work with root privileges, do as much as you can as an ordinary user, log in as root or give an su or sudo command so you are working with root privileges, do as much of the task that has to be done with root privileges, and revert to being an ordinary user as soon as you can. Because you are more likely to make a mistake when you are rushing, this concept becomes more important when you have less time to apply it. Lightweight Directory Access Protocol See LDAP. link A pointer to a file. Two kinds of links exist: hard links (page 956) and symbolic links (page 982) also called soft links. A hard link associates a filename with a place on the disk where the contents of the file is located. A symbolic link associates a filename with the pathname of a hard link to a file. Linux-PAM See PAM on page 970. LinuxSee PAM on page 970. Pluggable Authentication Modules list box A widget (page 987) that displays a static list for a user to choose from. The list appears as multiple lines with a scrollbar (page 977) if needed. The user can scroll the list and select an entry. Different from a drop-down list (page 952). loadable kernel module See loadable module. loadable module A portion of the operating system that controls a special device and that can be loaded automatically into a running kernel as needed to access that device. local area network See LAN. locale The language; date, time, and currency formats; character sets; and so forth that pertain to a geopolitical place or area. For example, en_US specifies English as spoken in the United States and dollars; en_UK specifies English as spoken in the United Kingdom and pounds. See the locale man page in section 5 of the system manual for more information. Also the locale utility. 964 Glossary log in To gain access to a computer system by responding correctly to the login: and Password: prompts. Also log on, login. log out To end your session by exiting from your login shell. Also log off. logical expression A collection of strings separated by logical operators (>, >=, =, !=,