Dave Taylor

FIFTH EDITION

Covers OS X, Linux, and Solaris

Sams Teach Yourself

Unix

in 24 Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Dave Taylor

Sams **Teach Yourself**

Unix

in 24 Hours

FIFTH EDITION

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself Unix in 24 Hours, Fifth Edtion

ISBN-13: 978-0-672-33730-7

ISBN-10: 0-672-33730-4

Library of Congress Control Number: 2015913255

Printed in the United States of America

First Printing October 2015

Trademarks

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

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

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

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

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

Acquisitions Editor
Mark Taber

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Kitty Wilson

Indexer
WordWise Publishing Services, LLC

Proofreader
Laura Hernandez

Technical Editors
Siddhartha Singh
Brian Tiemann

Editorial Assistant
Vanessa Evans

Cover Designer
Mark Shirar

Compositor
codeMantra

Contents at a Glance

Introduction 1

HOUR 1 What Is This Unix Stuff? 3
2 Getting onto the System and Using the Command Line 23
3 Moving About the File System 39
4 Listing Files and Managing Disk Usage 59
5 Ownership and Permissions 83
6 Creating, Moving, Renaming, and Deleting Files and Directories 107
7 Looking into Files 123
8 Filters, Pipes, and Wildcards! 141
9 Slicing and Dicing Command-Pipe Data 163
10 An Introduction to the `vi` Editor 177
11 Advanced `vi` Tricks, Tools, and Techniques 209
12 An Overview of the `emacs` Editor 241
13 Introduction to Command Shells 263
14 Advanced Shell Interaction 279
15 Job Control 295
16 Shell Programming Overview 313
17 Advanced Shell Programming 333
18 Printing in the Unix Environment 347
19 Archives and Backups 365
20 Using Email to Communicate 385
21 Connecting to Remote Systems Using SSH and SFTP 403
22 Searching for Information and Files 415
23 Perl Programming in Unix 427
24 GNOME and the GUI Environment 441

Appendix
A Common Unix Questions and Answers 455

Index 463

Table of Contents

Introduction 1

HOUR 1: What Is This Unix Stuff? 3
- What Is Unix? 3
- A Brief History of Unix 5
- What's All This About Multiuser Systems? 7
- Cracking Open the Shell 8
- Getting Help 9

HOUR 2: Getting onto the System and Using the Command Line 23
- Beginning Your Session 23
- Seeing What's Going On Around You 30

HOUR 3: Moving About the File System 39
- What a Hierarchical File System Is All About 39
- Directory Separator Characters 45
- The Difference Between Relative and Absolute Filenames 47

HOUR 4: Listing Files and Managing Disk Usage 59
- The `ls` Command 60
- Special `ls` Command Flags 67
- Permissions Strings 70

HOUR 5: Ownership and Permissions 83
- Working with File Permissions 83

HOUR 6: Creating, Moving, Renaming, and Deleting Files and Directories 107
- Manipulating the Unix File System 107

HOUR 7: Looking into Files 123
- Looking Inside Files 123

HOUR 8: Filters, Pipes, and Wildcards! 141
- Maximizing the Command Line 142

HOUR 9: Slicing and Dicing Command-Pipe Data 163
The awk Programming System 164
How to Use cut in Pipes 169
Inline Editing with sed and tr 171

HOUR 10: An Introduction to the vi Editor 177
Editing the Unix Way 178

HOUR 11: Advanced vi Tricks, Tools, and Techniques 209
Advanced Editing with vi 209
Summary of vi Commands 238

HOUR 12: An Overview of the emacs Editor 241
The Other Popular Editor: emacs 242

HOUR 13: Introduction to Command Shells 263
The (Command) Shell Game 263

HOUR 14: Advanced Shell Interaction 279
Which Shell Is Which? 280

HOUR 15: Job Control 295
Wrestling with Your Jobs 295

HOUR 16: Shell Programming Overview 313
Building Your Own Commands 314

HOUR 17: Advanced Shell Programming 333
Searching a Database of Filenames with mylocate 334

HOUR 18: Printing in the Unix Environment 347
Making a Printed Copy 348

HOUR 19: Archives and Backups 365
The tar Tape Archive Utility 366
The zip Archive Utility 372
Shrinking Your Files with compress 375
Exploring the Unix Tape Command: cpio 377
Personal Backup Solutions 379
Working with Linux Package Managers 381

HOUR 20: Using Email to Communicate . . . 385
Interacting with the World . . . 386

HOUR 21: Connecting to Remote Systems Using SSH and SFTP . . . 403
Stepping Beyond Your Own System . . . 403

HOUR 22: Searching for Information and Files . . . 415
Finding What's Where . . . 415

HOUR 23: Perl Programming in Unix . . . 427
Flexible and Powerful: Perl . . . 428

HOUR 24: GNOME and the GUI Environment . . . 441
Tweaking Your Inner GNOME . . . 442
Working with GNOME Applications . . . 445

APPENDIX A: Common Unix Questions and Answers . . . 455
How do I use `find|xargs` with filenames that contain spaces? . . . 455
How do I find large files on my system? . . . 456
How do I run a program on a schedule? . . . 457
How do I fix file permission problems? . . . 458
How do I list files that don't match a given pattern? . . . 458
How do I view lines *X*–*Y* in a text file? . . . 458
How do I add a new directory to my `PATH`? . . . 459
How do I recover deleted files? . . . 459
How can I set my shell to protect me from accidental deletions? . . . 460
What do the shell errors `arg list too long` and `broken pipe` mean? . . . 460
Why use `ssh` instead of `telnet`? Or `sftp` instead of `ftp`? . . . 461

Index . . . 463

About the Author

Dave Taylor is president of Intuitive Systems, LLC, a consulting firm focused on online communications and marketing strategies. Founder of four Internet startups, he has been involved with Unix and the Internet since 1980, having created the popular Elm Mail System and Embot mail autoresponder. A prolific author, he has been published more than 1,000 times, and his most recent books include the best-selling *Wicked Cool Shell Scripts* and *Learning Unix for Mac OS X*.

A popular columnist for *Linux Journal*, he also writes a tech Q&A column for the *Boulder Colorado Daily Camera* newspaper. Previously, he was a research scientist at HP Palo Alto Laboratories. He has contributed software to the 4.4 release of Berkeley Unix (BSD), and his programs are found in all versions of Linux and other popular Unix variants.

Dave has a bachelor's degree in computer science (University of California at San Diego), a master's degree in educational computing (Purdue University), and an MBA (University of Baltimore), and he is a top-rated public speaker who frequently offers workshops on online marketing, blogging, and various technical topics. His official home page on the Web is http://www.DaveTaylorOnline.com, and his email address is d1taylor@gmail.com.

Dave also maintains three weblogs online, Ask Dave Taylor (at www.askdavetaylor.com), where he fields questions from readers on a wide variety of topics; GoFatherhood (at www.GoFatherhood.com), where he talks about the challenges and joys of parenting; and Dave On Film (www.DaveOnFilm.com), where he shares his reviews of the latest movies. You're invited to get involved at all three!

Dedication

To the lights of my life: Ashley, Gareth, and Kiana.

Acknowledgments

However you slice it, you can't write a book locked in a cave (even if there's a high-speed Internet connection and fancy computer therein), and this book has evolved over many, many years, starting its life as an *Interactive Unix* tutorial I was writing for Sun Microsystems. In the interim, a number of people have added their spices to the stew, most notably my co-author for the first and second editions of *Teach Yourself Unix in 24 Hours*, James C. Armstrong, Jr.

In this new fifth edition, I've been delighted by the cooperative and talented team at Sams Publishing, again, and would like to specifically thank Mark Taber and Tonya Simpson, and my tech editors Brian Tiemann and Siddhartha Singh for all their ideas and commentary on how to make this book really superb. Any technical errors remaining are my own responsibility.

Finally, I would like to acknowledge and thank my kids for letting me focus on updating this book, chapter by chapter, even when there were games and other activities that could have proven more fun. I wouldn't trade them in, even for a 1THz PC! :-)

We Want to Hear from You!

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

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

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

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

Email: feedback@samspublishing.com

Mail: Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

Introduction

Welcome to the fifth edition of *Sams Teach Yourself Unix in 24 Hours*! This book has been designed to be helpful as a guide as well as a tutorial for both beginning users and those with previous Unix or Linux experience. The reader of this book is assumed to be intelligent, but no familiarity with Unix is expected or required.

Does Each Chapter Take an Hour?

You can learn the concepts in each of the 24 lessons in one hour. If you want to experiment with what you learn in each lesson, you might take longer than an hour. However, all the concepts presented here are straightforward. If you are familiar with Windows applications or the Macintosh, you will be able to progress more quickly through the lessons.

What if I Take Longer Than 24 Hours?

Since the publication of the first edition of this book, I've received a considerable amount of praise and positive feedback, but the one message that has always been a surprise is "I finished your book, but it took me a lot longer than 24 hours." Now you can read here, directly from the author: That's okay! Take your time and make sure you try everything as you go along. Learning and remembering are more important than speed. And if you do finish it all in 24 hours, let me know!

How to Use This Book

This book is designed to teach you topics in one-hour lessons. All the books in the *Sams Teach Yourself* series enable you to start working and become productive with a topic as quickly as possible. This book will do that for you!

Each hour, or lesson, starts with an overview of the topic to inform you of what to expect in that lesson. The overview helps you determine the nature of the lesson and whether the lesson is relevant to your needs.

Main Section

Each lesson has a main section that discusses the lesson topic in a clear, concise manner by breaking the topic down into logical components and explaining each component clearly.

Interspersed throughout each lesson are special elements, called tips, notes, and cautions, which provide additional information.

NOTE

Notes are designed to clarify the concept that is being discussed or elaborate on the subject. If you are comfortable with your understanding of the subject, you can bypass them without danger.

TIP

Tips inform you of tricks or elements that are easily missed by most computer users. You can skip them, but often tips show you an easier way to do a task.

CAUTION

A caution deserves at least as much attention as a tip because cautions point out problematic elements of the topic being discussed. Ignoring the information contained in a caution could have adverse effects on the task at hand. These are the most important special elements in this book.

Tasks

This book offers another special element called tasks. These step-by-step exercises are designed to walk you quickly through the most important skills you can learn in Unix.

Workshops

The Workshop section at the end of each lesson provides lists of key terms and exercises that reinforce concepts you learned in the lesson and help you apply them in new situations. You can skip the Workshop section, but we recommend that you go through the exercises to see how the concepts can be applied to other common tasks. The key terms also are compiled in one alphabetized list in the Glossary at the end of the book.

This page intentionally left blank

HOUR 2

Getting onto the System and Using the Command Line

Goals for This Hour

In this hour, you will learn

- How to log in to and log out of the system
- How to change your password with the `passwd` command
- About choosing a memorable and secure password
- How to find out who the computer thinks you are
- How to find out who else is on the system
- How to find out what everyone is doing on the system
- About checking the current date and time

In this second Unix lesson, it's time for you to log in to the system and try some commands. This hour focuses on learning the basics of interacting with your Unix machine.

This hour introduces many commands, so it's very important that you have a Unix system available on which you can work through all the examples. Most examples have been taken from a PC running Solaris 11, a variant of Unix System V Release 4, and have been double-checked on both a BSD-based system and a Mac OS X command line. Any variance between the three is noted. If you have a Unix system available, odds are good that it's based on either AT&T System V or Berkeley Unix.

Beginning Your Session

Before you can start interacting with the Unix command shell of your choice, you need to learn how to log in to your account. The good news is that it's easy! Let's have a look.

Task 2.1: Logging In to and Out of the System

Because Unix is a multiuser system, user authentication is always enforced: You always need to provide credentials (generally a username and a password) to the system so that it knows who you are. Some modern user-friendly flavors of Unix (such as Mac OS X) allow you to bypass this

requirement by always booting into a single user's desktop session, but this is just a convenience feature; under the hood, all Unix flavors are the same, and all require that you authenticate yourself at some stage of the process.

Old-school hardware terminals do still exist, or you might choose to boot a Linux or FreeBSD box directly to the textual console; but if you're new to Unix, you'll most likely need an application known as a terminal to access the command line. Most graphical operating systems include one. I use the Terminal app included with Mac OS X (in the Utilities folder) whether I'm accessing my local system or just opening an environment in which to connect to a remote system via `ssh`.

TIP

Most Linux flavors have a prominently available terminal program for your use; on a Windows PC, your best bet is the freeware program PuTTY, available at http://www.putty.org.

If you need to actually log in, the first thing you'll see on the screen will look something like this:

```
GNU/Linux ado.aplonis.net 5:38pm on Tue, 8 Jul 2014
login:
```

The first line of this challenge prompt indicates what variant of Unix the system is running (GNU/Linux in this case), the hostname of the computer system, and the current time and date. The second line asks for your login, also known as your username or account name.

NOTE

If you connect to a Unix server via the network, using either `telnet` or `ssh`, you'll see the same login prompt, though I strongly recommend that you always use `ssh` for security reasons. If you use a terminal program within a graphical environment, you won't need to log in because you've already logged in to your GUI session.

1. Know your account name. It would be nice if computers could keep track of users by simply using full names so that I could enter `Dave Taylor` at the login prompt. Alas, like the Internal Revenue Service, the Department of Motor Vehicles, and many other agencies, Unix does not use names but instead assigns each user a unique identifier. This identifier, called an *account name*, has eight characters or fewer and is usually based on the user's first or last name, although it can be any combination of letters and numbers. I have two account names, or logins, on the systems I use: `taylor` and, on another machine where someone already had that account name, `d1taylor`.
2. Know your password. Perhaps your account name is on a piece of paper with your initial password, both assigned by the Unix system administrator. If you do not have this information, you need to track it down before you can go further. Some accounts might not have an initial password; in that case, you won't have to enter one the first time you log in

to the system. If that's the case, *create a password* for your own security. In a few minutes, you will learn how you can give yourself the password of your choice by using the Unix command `passwd`.

Note that a lot of systems are accessible only through the `ssh` function, and so a common way to connect to a modern system is to open up a local terminal app on your Mac or PC and type in something like:

```
$ ssh taylor@intuitive.com
```

where `taylor` is the account name and `intuitive.com` is the name of the remote host. If that's how you need to access your Unix system remotely, it's actually easier than using the login/password sequence; you just need to make extra sure that you type in everything exactly as prompted.

3. At the login prompt, enter your account name if needed:

```
login: taylor
Password:
```

Be particularly careful to use exactly what your administrator tells you to use (for example, the accounts `taylor`, `Taylor`, and `TAYLOR` are all different to Unix). After you've entered your account name, the system moves the cursor to the next line and prompts you for your password. If you're using the `ssh` sequence, then the prompt will include your account name, as shown here:

```
taylor@intuitive.com's password:
```

Either way, when you enter your password, the system won't echo it (that is, won't display it) on the screen. That's okay. Lack of an echo doesn't mean anything is broken; instead, this is a security measure to ensure that even if people are looking over your shoulder, they can't learn your secret password by watching your screen. Be certain to type your password correctly because you won't see what you've typed and have a chance to correct it.

NOTE

If you enter either your login or your password incorrectly, the system complains with an error message:

```
login: taylor
Password:
Login incorrect
login:
```

Most systems give you three or four attempts to get both your login and password correct, so try again. Don't forget to enter your account name at the login prompt each time, as required. Be careful, though: Too many failed login attempts, and you might lock out your account and have to contact the administrator for help.

4. After you've successfully entered your account name and password, you are shown some information about the system, some news for users, perhaps a fortune, and an indication of whether you have electronic mail. The specifics will vary, but here's an example of what I see when I log in to my account:

```
login: taylor
Password:
Last login: Thu Jul 7 17:00:23 on ttyAe
You have mail.
$
```

NOTE

The dollar sign prompt is Unix's way of telling you that it's ready for you to enter some commands. It is the equivalent of a soldier saluting and saying, "Ready for duty!" or an employee saying, "What shall I do now, boss?"

Your system might be configured so that you have a slightly different prompt here. The possibilities include a `%` for the C shell, your current location in the file system, the current time, the command-index number (which you'll learn about when you learn how to teach the Unix command-line interpreter to adapt to your work style rather than vice versa), and the name of the computer system itself. Here are some examples:

```
[/users/taylor] :
(mentor) 33 :
taylor@mentor %
```

Your prompt might not look exactly like any of these, but you know you're looking at a prompt because it's at the beginning of the line on which your cursor sits, and it reappears each time you've completed working with any Unix program. That's how you know the program has completed its task.

5. At this point, you're ready to enter your first Unix command, `exit`, to sign off from the computer system. Try it. On my system, entering `exit` shuts down all my programs and quits the terminal app. On other systems, it returns you to the login prompt. Many Unix systems offer a pithy quote as you leave, too.

```
% exit
He who hesitates is lost.
login:
```

NOTE

You might be able to end your session by pressing Ctrl-D. Some shells will catch this and prompt you to determine whether you want to end your session; others will exit. Ctrl-D is actually an end-of-file character; it may be different on your system.

6. If you have a direct connection to the computer because you're using a shared system in a computer center, library, or similar, odds are very good that logging out causes the system to prompt for another account name, enabling the next person to use the system. If you manually connected to the system via the Internet, you probably will see something more like the following example. After being disconnected from the remote system, you'll then be able to safely shut down your local computer:

```
% exit
Did you lose your keys again?

Connection to 154.23.11.140 closed.
```

NOTE

Unix is *case sensitive*, so the `exit` command is not the same as `EXIT`. If you enter a command all in uppercase, the system won't find any such program or command and instead will respond with the complaint `command not found`. Get in the habit of using all lowercase for commands and Unix input. Lowercase is the natural Unix style.

At this point, you've stepped through the toughest parts of getting started with Unix. You have an account, know the password, have logged in to the system, and have entered a simple command telling the computer what you want to do, and the computer has done it!

Task 2.2: Changing Passwords with `passwd`

Having logged in to a Unix system, you can clearly see that many differences exist between Unix and a PC or Macintosh personal computer. Certainly the style of interaction is different. With Unix command lines, the keyboard becomes the exclusive method of instructing the computer what to do, and the mouse sits idle. One of the greatest differences is that Unix is a multiuser system, as you learned in the preceding hour. As you learn more about Unix, you'll find that this characteristic has an impact on various tasks and commands. The next Unix command you'll learn about is one that exists because of the multiuser nature of Unix: `passwd`.

With the `passwd` command, you can change the password associated with your individual account name. As with your personal identification number (PIN) for automated-teller machines, the value of your password is directly related to how secret it remains.

NOTE

Unix is careful about the whole process of changing passwords. It requires you to enter your current password to prove you're really you. Imagine that you are at a computer center and have to leave the room to make a quick phone call. Without much effort, a prankster could lean over and quickly change your environment or even delete some critical files! That's why you should log out if you're not going to be near your system, and that's also why passwords are never echoed in Unix.

1. Consider what happens when I use the `passwd` command to change the password associated with my account:

```
% passwd
Changing password for taylor.
Old password:
New passwd:
Retype new passwd:
%
```

2. Notice that I never received any visual confirmation that the password I actually entered was the same as the password I thought I entered. This is not as dangerous as it seems, though, because if I had made any typographical errors, the password I entered the second time (when the system said `Retype new passwd:`) wouldn't have matched the first. In a no-match situation, the system would have warned me that the information I supplied was inconsistent:

```
% passwd
Changing password for taylor.
Old password:
New passwd:
Retype new passwd:
Mismatch - password unchanged.
%
```

3. Smart systems will complain if you pick a really bad password or one that's just obviously too short. I tried `cat` on my Oracle Solaris system, and the `passwd` command complained:

```
passwd: Password too short - must be at least 6 characters.
```

   Oops. In the next section you'll learn about how to pick good, hard-to-guess but easy-to-remember passwords.

After you change the password, don't forget it. Resetting it to a known value if you don't know the current password requires the assistance of a system administrator or other operator. Using a trick to remember your password can be a Catch-22, though: You don't want to write down the password because that reduces its secrecy and you don't want to make it too easy to remember because someone else can then guess it, but you don't want to forget it, because that can be all sorts of hassle. You want to be sure that you pick a good password, too, as described in Task 2.3.

Task 2.3: Picking a Secure Password

If you're an aficionado of old movies, you are familiar with the thrillers in which the hoods break into an office and spin the dial on the safe a few times, snicker a bit about how the boss shouldn't have chosen his daughter's birthday as the combination, and crank open the safe. (If you're really familiar with the genre, you recall films in which the criminals rifle the desk drawers and find the combination of the safe taped to the underside of a drawer as a fail-safe,

or a failed safe, as the case may be. Hitchcock's great film *Marnie* has just such a scene.) The moral is that even the best secret password is useful only if you keep it secret.

For computers, security is tougher because a fast computer system can test all the words in an English dictionary against your account password faster than you can say "don't hack me, bro." If your password is *kitten* or, worse yet, your account name, any semicompetent bad guy could be in your account and messing with your files in no time. This is called a *dictionary attack*.

Most modern Unix systems have some *heuristics*, or smarts, built in to the `passwd` command; the heuristics check to determine whether what you've entered is reasonably secure.

The tests performed typically answer these questions:

- Is the proposed password at least six characters long? (A longer password is more secure.)
- Does it have both digits and letters? (A mix of both is best.)
- Does it mix upper- and lowercase letters? (A mix is best.)
- Does it include at least one punctuation character? (adding a `%`, `!`, `@`, or even `.` is best)
- Is it in the online dictionary? (You should avoid common words.)
- Is it a name or word associated with the account? (`Dave` would be a bad password for my account `taylor` because my full name on the system is Dave Taylor).

Some versions of the `passwd` program are more sophisticated, and some less, but generally the following are good guidelines for picking a secure password:

1. An easy way to choose memorable and secure passwords is to think of them as small sentences rather than as a single word with some characters surrounding it. If you're a fan of Alexander Dumas and *The Three Musketeers*, then "All for one and one for all!" is a familiar cry, but it's also the basis for a couple of great passwords. Easily remembered derivations might be the punnish `awl4ONE?` or `a41&14A!`.
2. If you've been in the service, you might have the old U.S. Army jingle stuck in your head: "Be All You Can Be." Try thinking of that phrase as a series of abbreviations and letters: `ballucanb`. Turn that into a good password with a few additional tweaks: 4ballu@canb. You might have a self-referential password: `account4me` or `MySekrit` would work. If you're ex-Vice President Dan Quayle, `1Potatoe` could be a memorable choice. (`potatoe` by itself wouldn't be particularly secure because it lacks digits and lacks uppercase letters and because it's a simple variation on a word in the online dictionary.)
3. Another way to choose passwords is to find acronyms that have special meaning to you. Don't choose simple ones. Remember, short ones aren't going to be secure. But if you have always heard that "Real programmers don't eat quiche!" then `Rpdeq!` could be a complex password that you'll easily remember.

4. Many systems you use every day require numeric passwords to verify your identity, including the automated-teller machine (with its PIN), government agencies (with the Social Security number), and the Department of Motor Vehicles (your driver's license number or vehicle license). Each of these actually is a poor Unix password because it's too easy for someone to find out your license number or Social Security number. And a series of nothing but numbers is a terrible password anyway!

NOTE

The important thing is to come up with a strategy of your own for choosing a password that is both memorable and secure. Then keep the password in your head rather than write it down.

Why be so paranoid? For a small Unix system that will sit on your desk and won't have any other users, a high level of concern for security is, to be honest, unnecessary. As with driving a car, though, it's never too early to learn good habits. Any system that has Internet access means that it's probably accessible *from* the Internet, too, and that means it's at risk of hackers trying to break in, a target for delinquents who relish the intellectual challenge of breaking into an account and then altering and destroying files and programs purely for amusement.

The best way to avoid trouble is to develop good security habits now, when you're first learning about Unix. Learn how to recognize what makes a good, secure password, pick one for your account, and keep it a secret. Don't write it down, or, if you must, keep that note secure too and notify your admin if it gets lost. A little prevention can be a lot easier than mopping up after a security breech.

With that in mind, log in again to your Unix system and try changing your password. First, change it to `easy` and see whether the program warns you that `easy` is too short or otherwise a poor choice. Then try entering two different secret passwords to see whether the program notices the difference. Finally, pick a good password, using the preceding guidelines and suggestions, and change your account password to be more secure.

Seeing What's Going On Around You

You're logged in, looking at the command prompt, and ready to delve into this Unix thing. Great! Let's have a look.

Task 2.4: Who Are You?

While you're logged in to the system, you can learn a few more Unix commands, including a couple that can answer a philosophical conundrum that has bothered men and women of thought for thousands of years: Who am I?

1. The easiest way to find out "who you are" is to enter the `whoami` command:

```
% whoami
taylor
%
```

   Try it on your system. The command lists the account name associated with the current login.

2. Ninety-nine percent of the commands you type with Unix have a single specific spelling and will fail if you get creative. With `whoami`, however, adding spaces to transform the statement into proper English—that is, entering `who am I`—dramatically changes the result. On my system, I get the following results:

```
% who am i
taylor     pts/2        Oct 27 10:11    (:0.0)
%
```

   On a Mac system, it doesn't show (:0.0) otherwise things work well.

   This tells me quite a bit about my identity on the computer, including my account name and where and when I logged in. Try the command on your system to see what results you get.

   In this example, my account name is `taylor`. The `pts/2` is the current communication line I'm using to access the system, and you can see that I logged in at 10:11 using a regular communications socket. (The :0.0 is relevant under the X Window System, something we won't cover for quite a while in this book.)

NOTE

Unix is full of oddities that are based on historical precedent. One is `tty` or `pty` to describe a computer or terminal line. This comes from the earliest Unix systems, in which Digital Equipment Corporation teletypewriters were hooked up as interactive devices. The teletypewriters quickly received the nickname tty, and all these years later, when people wouldn't dream of hooking up a teletypewriter, the line is still known as a `tty` (or `pty`, for "pseudo terminal") line.

3. One of the most dramatic influences Unix systems have had on the computing community is the propensity for users to work together on a network, hooked up by telephone lines and modems (the predominant method until the middle to late 1980s) or by high-speed network connections to the Internet (a more common type of connection today). Regardless of the connection, however, you can see that each computer needs a unique identifier to distinguish it from others on the network. In the early days of Unix, systems had unique hostnames, but as hundreds of systems have grown into millions, this has proved to be an unworkable solution.

4. The alternative was what's called a domain-based naming scheme, where systems are assigned unique names within specific subsets of the overall network. Here's an example:

   ```
   mentor.utech.edu
   ```

   The computer I use is within the `.edu` domain (read the hostname and domain—`mentor.utech.edu`—from right to left), meaning that the computer is located at an educational institution. Then, within the educational institution subset of the network, `utech` is a unique descriptor, and, therefore, if other UTech universities existed, they couldn't use the same top-level domain name. Finally, `mentor` is the name of the computer itself.

5. As with learning to read addresses on envelopes, learning to read domain names can unlock much information about a computer and its location. For example, `lib.stanford.edu` is the library computer at Stanford University, and `ccgate.infoworld.com` tells you that the computer is at InfoWorld, a commercial computer site, and that its hostname is `ccgate`. You'll learn more about this later when you learn how to use electronic mail to communicate with people throughout the Internet.

6. Another way to find out who you are in Unix is to use the `id` command. The purpose of this command is to tell you what group or groups you're in and the numeric identifier for your account name (known as your *user ID number* or *user ID*). Enter `id` and see what you get. I get the following result:

   ```
   % id
   uid=100(taylor) gid=10(staff)
   %
   ```

NOTE

If you enter `id` and the computer returns a different result or indicates that you need to specify a filename, don't panic. On many Berkeley-derived systems, the `id` command is used to obtain low-level information about files.

7. In this example, you can see that my account name is `taylor` and that the numeric equivalent, the user ID, is `100`. (Here it's abbreviated as `uid`—pronounce it "you-eye-dee" to sound like a Unix expert). Just as the account name is unique on a system, so also is the user ID. Fortunately, you rarely, if ever, need to know these numbers since they're used by the OS internally, so focus on the account name and group name.

8. Next, you can see that my group ID (or `gid`) is `10` and that group number `10` is known as the `staff` group. It's the only group to which I belong.

   On another system, I am a member of two different groups:

   ```
   % id
   uid=103(taylor) gid=10(staff) groups=10(staff),44(ftp)
   %
   ```

Although I have the same account name on this system (`taylor`), you can see that my user ID and group ID are both different from those in the earlier example. Note also that I'm a member of two groups: the `staff` group, with a group ID of `10`, and the `ftp` group, with a group ID of `44`.

You've now learned a couple different ways to have Unix give you some information about your account. Later, you'll learn how to set protection modes on your files so that people in your group can read your files but so those not in your group are barred from access.

Task 2.5: Finding Out What Other Users Are Logged In to the System

The next philosophical puzzle that you can solve with Unix is "Who else is there?" The answer, however, is rather restricted, limited to only those people currently logged in to the same computer at the same time. Three commands are available to get you this information, and the one you choose depends on how much you'd like to learn about the other users: `users`, `who`, and `w`.

1. The simplest of the commands is the `users` command, which lists the account names of all people using the system:

```
% users
david mark taylor
%
```

   In this example, `david` and `mark` are also logged in to the system with me. Try this on your computer and see what other users—if any—are logged in to your computer system.

2. A command that you've encountered earlier in this hour can be used to find out who is logged on to the system, what line they're on, and how long they've been logged in. That command is `who`:

```
% who
taylor     vt/7         Oct 27 14:10    (:0)
david      pts/1        Dec 27 15:54    (:0.0)
mark       pts/2        Oct 27 11:51    (:0.0)
%
```

   Here, you can see that three people are logged in: `taylor` (me), `david`, and `mark`. Furthermore, you can now see that `david` is logged in by connection `pts/1` and has been connected since December 27 at 3:54 p.m. You can see that `mark` has been connected since just before noon on October 27 on line `pts/2`. Note that I have been logged in since 14:10, which is 24-hour time for 2:10 p.m. Unix doesn't always indicate a.m. or p.m.

The `user` and `who` commands can tell you who is using the system at any particular moment, but how do you find out what they're doing?

Task 2.6: What Is Everyone Doing on the Computer?

To find out what everyone else is doing, there's a third command, `w`, that serves as a combination of "Who are they?" and "What are they doing?"

1. Consider the following output from the `w` command:

```
% w
2:12pm  up 7 days,  5:28,  3 users, load average: 0.33, 0.33, 0.02
User     tty           login@  idle   JCPU   PCPU  what
taylor   vt/7         27Oct14          2:35   2:07  python2.6 /usr/lib/
                                                   ➥ system-config
david    pts/1         3:54pm   2:04    15     33  bash
mark     pts/2        27Oct14     43                -csh
%
```

   This is a much more complex command than `users` or `who`, and it offers more information. Notice that the output is broken into different areas. The first line summarizes the status of the system and, rather cryptically, the number of programs that the computer is running at one time. Finally, for each user, the output indicates the username, the tty, when the user logged in to the system, how long it's been since the user has done anything (in minutes and seconds), the combined CPU time of all jobs the user has run, and the amount of CPU time taken by the current job. The last field tells you what you wanted to know in the first place: What are the users doing?

   In this example, the current time is 2:12 p.m., and the system has been up for 7 days, 5 hours, and 28 minutes. Currently three users are logged in, and the system is very quiet, with an average of 0.33 jobs submitted (or programs started) in the last minute; 0.33 jobs, on average, in the last 5 minutes; and 0.02 jobs in the last 15 minutes.

   `taylor` is the only user actively using the computer (that is, who has no idle time) and is using the `python` command. User `david` is sitting in the bash shell, which has gone for quite awhile without any input from the user (2 hours and 11 minutes of idle time). The program already has used 15 seconds of CPU time and, overall, `david` has used 33 seconds of CPU time. User `mark` has a C shell running, as indicated by `-csh`. (The leading dash indicates that this is the program that the computer launched automatically when `mark` logged in. This is akin to how the system automatically launches the Finder on a Macintosh.) User `mark` hasn't actually done anything yet: Notice that there is no accumulated computer time for that account.

2. Now it's your turn. Try the `w` command on your system and see what kind of output you get. Try to interpret all the information based on the explanation here. One thing is certain: Your account should have the `w` command listed as what you're doing.

On a multiuser Unix system, the `w` command gives you a quick and easy way to see what's going on.

Task 2.7: Checking the Current Date and Time

You've learned how to orient yourself on a Unix system, and you are now able to figure out who you are, who else is on the system, and what everyone is doing. What about the current time and date?

1. Logic suggests that `time` shows the current time and `date` the current date; but this is Unix, and logic doesn't always apply. In fact, consider what happens when I enter `time` on my system:

   ```
   % time

   real    0m0.000s
   user    0m0.000s
   sys     0m0.000s
   %
   ```

   The output is cryptic to the extreme and definitely not what you're interested in finding out. The program is showing how much user time, system time, and CPU time has been used by the command interpreter itself, broken down by input/output operations and more. (The `time` command is more useful than it looks, particularly if you're a programmer.)

   On other Unixes, you might find `time` to be a missing command, a built-in shell function, or something completely different. In all cases, it won't tell you the current time.

2. Well, `time` didn't work, so what about `date`?

   ```
   % date
   Sat Jun  617:05:32 MST 2015
   %
   ```

   That's more like it!

   Try the `date` command on your computer and see whether the output agrees with your watch.

How do you think `date` keeps track of the time and date when you've turned off the computer? Does the computer know the correct time if you unplug it for a few hours? (I hope so. Almost all computers today have little batteries inside for just this situation.)

Summary

This hour focuses on giving you the skills required to log in to a Unix system, figure out who you are and what groups you're in, change your password, and log out again. You also learned how to list the other users of the system, find out what Unix commands they're using, and check the date and time.

Workshop

The Workshop summarizes the key terms you've learned and poses some questions about the topics presented in this lesson. It also provides you with a preview of what you will learn in the next hour.

Key Terms

account name This is the official one-word name by which the Unix system knows you; mine is `taylor`. (*See also* **account** in Hour 1, "What Is This Unix Stuff?")

domain name Unix systems on the Internet, or any other network, are assigned a domain within which they exist. This is typically the company (for example, `microsoft.com` for Microsoft Corporation) or institution (for example, `lsu.edu` for Louisiana State University). The domain name is always the entire host address, except the hostname itself. (*See also* **hostname**.)

heuristic An approach or a procedure for accomplishing a specific task, not guaranteed of success but widely accepted as providing good results for relatively little effort. Think "rule of thumb."

hostname Unix computers all have unique names assigned by the local administration team. The computers I use are `limbo`, `well`, `netcom`, and `mentor`, for example. Enter `hostname` to see what your system is called.

login A synonym for account name, this also can be a verb (when it's two words: log in) that refers to the process of connecting to the Unix system and entering your account name and password for your account.

user ID (uid) This is the numeric equivalent of the account name, which the system uses for internal bookkeeping.

Exercises

1. Why can't you have the same account name as another user? How about user ID? Can you have the same `uid` as someone else on the system?

2. Which of the following are good passwords, based on the guidelines you've learned in this hour?

```
foobar      4myMUM      Blk&Blu
234334      Laurie      Hi!
2cool.      rolyat      j j kim
```

3. Are the results of the two commands `who am i` and `whoami` different? If so, explain how. Which do you think you'd rather use when you're on a new computer?

4. List the three Unix commands for finding out who is logged in to the system. Describe about the differences between the commands.

5. One of the commands in the answer to question 4 indicates how long the system has been running. (In the example, it had been running for seven days.) What value do you think there is for keeping track of this information?

6. If you can figure out what other people are doing on the computer, they can figure out what you're doing, too. Does that bother you? Why or why not?

Preview of the Next Hour

The next hour focuses on the Unix hierarchical file system. You'll learn about how the system is organized, how it differs from Windows and Macintosh hierarchical file systems, the difference between relative and absolute filenames, and the mysterious `.` and `..` directories. You'll also learn about the `env`, `pwd`, and `cd` commands, as well as the `HOME` and `PATH` environment variables.

This page intentionally left blank

This page intentionally left blank

Index

SYMBOLS

[ ] (square brackets), 150
(command, 238
) command, 238
: ! command, 238
! command, 255
!! command, 238, 286
! $ command, 286
! * command, 286
! } command, 238
! escape-to-Unix command, 232-237
! n command, 286
! ptrn command, 286
$ (dollar sign), 314
$ command, 205
* (asterisk), 125
+ (plus) sign, 220
, (comma), 222
. (dot), 49
/ (slash), 42, 46
: (colons), 46, 170, 214-219
:ab a bcd command, 238
:ab command, 238
:map a bcd command, 238
:map command, 238
:set nonumber command, 238
:set number command, 238
:s/old/new/ command, 238
:s/old/new/g command, 238
@ (at sign), 42
\{ command, 238
^ (carat), 159
^a^b command, 286
^b command, 205
^d command, 205
^f command, 206
^g command, 238
^u command, 206
^v command, 238
| (pipe), 129
0 command, 206
-1 flag, 64, 70, 72

A

A command, 257
a command, 205
-A flag, 379
-a flag, 64, 303, 379
absolute filenames, 47-56
access
- ! escape-to-Unix, 232-237
- concentric access models, 94
- CUPS (Common Unix Printing System), 349

adding. *See* inserting
Aho, Alfred, 169
aliases
- command shells, 286-290
- commands, 264, 327
- creating, 63

Alt key, 242
American Telephone and Telegraph (AT&T), 5
anonymous archives, FTP, 410-413
applications, GNOME, 445-452
applying
- cut command, 169-171
- find command, 420-422
- head program, 128-130
- PATH variables, 337-339
- permissions, 83-104

archives
- anonymous, FTP, 410-413
- compress command, 375-376
- cpio command, 377-379
- Linux package managers, 381-382
- tar commands, 366-372
- zip command, 372-375

arg list too long error, 460
arithmetic, programming shells, 316-317
asterisk (*), 125
at command, 457
at sign (@), 42
availability
- of command shells, 264-266
- disk space, checking, 77-79

The AWK Programming Language, 169
awk programs, 163-169
- commands, 304, 341
- navigating, 164

B

B command, 205, 257
b command, 205
background command. *See* bg (background) command
Backspace key, 182, 205
backups, 379-381. *See also* archives
bash (Bourne Again shell), 263, 265
- configuration files, navigating, 274-277
- history command, navigating, 280-281
- shells, programming, 326-331

Bash shells, 4
.bashrc file, 274
Bell Labs, 5
Berkley Mail, 386. *See also* mailx command
bg (background) command, 299-302
bin directory, 42
Bourne, Steven, 264
Bourne Again shell. *See* bash
Bourne shell. *See* sh
broken pipe error, 460
building mylocate scripts, 334-337

C

C, 6
C command, 225, 238, 257
c command, 225, 238
C shell. *See* csh
-C flag, 64
-c flag, 171, 371
C-a command, 248
calculating strings, numeric permissions, 98-100
cancel command, 360
carat (^), 159
case command, 321
cat program, 131-133
c-b command, 248
cd command, 54-56
C-d command, 252, 257
C-e command, 248
c-f command, 248
CGI (Common Gateway Interface), 438

change command, 225-232
characters
emacs editors, deleting, 249-252
separator (directories), 45-46
sets, 172
chmod command, 458
directories, modifying, 92-94
files, modifying, 94-98
numeric mode, 98
C-k command, 252
clients, PuTTY, 406
closing vi editors, 181
C-n command, 248, 257
col command, 355-359
colons (:), 46, 170, 214-219
com domain, 395
combining flags, 64-65
comma (,), 222
command line
email, sending from, 391-395
pipes, 163
cut command, 169-171
inline editing, 171-175
sed command, 172-175
tr command, 172-175
user interfaces, 4. *See also* shells
command prompts, navigating, 30-35
commands, 4, 10
(, 238
), 238
: !, 238
: ab a bcd, 238
!, 255
!!, 238, 286
! $, 286
! *, 286
! }, 238
! escape-to-Unix, 232-237
! n, 286
! ptrn, 286
$, 314
:ab, 238
:map, 238
:map a bcd, 238
:s/old/new/, 238
:s/old/new/g, 238
:set nonumber, 238
:set number, 238
^a^b, 286
^b, 205
^d, 205
^f, 206
^g, 238
^u, 206
^v, 238
0, 206
A, 257
a, 205
at, 457
aliases, 264, 327
awk, 304, 341
B, 205, 257
b, 205
bg (background), 299-302
C, 225, 238, 257
c, 225, 238
C-a, 248
cancel, 360
case, 321
c-b, 248
cd, 54-56
C-d, 252, 257
C-e, 248
c-f, 248
change, 225-232
chmod, 458
modifying directories, 92-94
numeric mode, 98
C-k, 252
C-n, 248, 257
col, 355-359
colon (:), 214-219
compress, 375-376
cp, 110-112
C-p, 248
cpio, 377-379
cron, 457
curl, 422-425
cut, 169-171, 268
C-v, 248
C-w, 257
C-x [, 248
C-x], 248
C-x b, 257
C-x Delete, 252
C-x u, 252
D, 205
d, 197, 205
date, 35, 380
Delete, 252
delete *msgs*, 386
df, 77-79, 340
DIR, 60
du (disk usage), 75-77

e, 238
echo, 148, 284
egrep, 158-160
emacs editors, 258-261
env, 52-53, 271
-exec, 419
exit, 26
export, 275
expr, 316, 318
F, 257
fg (foreground), 295, 299-302
file, 123-126, 323
  identifying file types, 123-126
  navigating Unix directories, 126-128
find, 334, 455-456
  applying, 420-422
  navigating, 415-420
flags, 73
for, 324
ftp, 461
G, 206
GET, 422-425
grep, 151-154, 334, 420, 458
h, 206
headers, 386, 389
help, 387
history
  navigating, 280-281
  shortcuts, 281-286
I, 257
i, 189, 206
if, 321
j, 206
K, 257
k, 206
kill, 307-310
L, 257
l, 206
lp, 351
lpadmin, 348
lpinfo, 355
lpr, 351
lprm, 360
lpstat, 348-349, 360
ls, 41, 52, 60-67
  file type indicators, 84
  flags, 67-70
  listing directory trees, 68-69
  long listing formats, 70-74
  modifying sorting, 67-68
ls -1, 104
lynx, 422-425
M, 257
M-<, 248
M->, 248
M-a, 248
mail *address*, 387
mailx, 386-391
M-b, 248
M-d, 252
M-Delete, 252
M-e, 248
M-f, 248
M-k, 252
mkdir, 108-110
more program, 136
motion. *See* motion commands
mv
  moving files, 112-113
  renaming files, 113-114
M-v, 248
N, 257
n, 206, 255
nG, 206
noclobber, 460
O, 192, 206
o, 192, 206
passwd, 27
pattern, 210
Perl programming, 438
pr, 355-359
print *msgs*, 387
printenv, 271
ps, 302-307
pwd, 54
python, 34
q, 255
query-replace, 256
quit, 181, 387
R, 225, 238
r, 225, 238
read, 324
:redo, 198
replace, 225-232
reply, 387
rm, 116, 118-120
rmdir, 114-115
S, 257
save, 389
save *folder*, 387
sed, 172-175, 459
sftp, 407, 461
sleep, 305

sort, 15
spell, 160
ssh, 404, 461
T, 257
tar, 366-372, 380
telnet, 403, 461
test, 318, 338
time, 35
touch, 74-75
tr, 172-175
trap, 342
tree, 69
U, 206
u, 197, 206
umask, 100-103
undelete *msgs*, 387
unzip, 373
users, 33
V, 257
vi editors, 205, 238. *See also* vi editors
W, 206, 257
w, 34-35, 206
whatis, 14
who, 185, 356
whoami, 31-33
x, 206
xargs, 420, 455-456
y, 255
zip, 372-375

command shells, 263
aliases, 286-288
availability of, 264-266
bash (Bourne Again shell) configuration files, 274-277
custom prompts, configuring, 290-292
history command
navigating, 280-281
shortcuts, 281-286
identifying, 267-268
navigating, 271-274
power aliases, 288-290
selecting, 269-271

C shell. *See* csh

Common Gateway Interface. *See* CGI

Common Unix Printing System. *See* CUPS

comparison functions, programming shells, 318-321

compress command, 375-376

compression, 79-81
unzip command, 373
zip command, 372-375

Computer Science Research Group, 5

concentric access models, 94

conditional expressions, programming shells, 321-324

configuring
backups, 379-381
configuration files, 274-277
CUPS (Common Unix Printing System), 349
custom prompts, command shells, 290-292
default variables at login, 271
GNOME (GNU Network Object Model Environment), 442-445
permissions
directories, 88-91
files, 84-88

connecting
remote Internet connections, 403-405
third-party SSH connections, 405-406

control
flow, 326
jobs, 295. *See also* jobs

Control key, 185

copying. *See also* moving
directories, 111
files, 110-112
SFTP (Secure FTP), 407-410

counting
files, 337-339
words, 143-144

cp command, 110-112

C-p command, 248

cpio command, 377-379

cron command, 457

csh (C shell), 4, 263, 280

CUPS (Common Unix Printing System), 349-351

curl command, 422-425

curses package, 181

cursor-control keys (vi editors), 182-185

custom prompts, configuring command shells, 290-292

cut command, 169-171, 268

C-v command, 248

C-w command, 257

C-x b command, 257

C-x [command, 248

C-x] command, 248

C-x Delete command, 252

C-x u command, 252
cycles, edit-compile-run, 283

D

D command, 205
d command, 197, 205
-d flag, 319, 379
-d (debug) flag, 438
date command, 35, 380
default variables, configuring login, 271
defaults, formatting permissions, 100-103, 109
Delete command, 252
delete *msgs* command, 386
deleting. *See also* removing
characters, emacs editors, 249-252
files, 114-115
recovering files, 459
text, vi editors, 197-205
df command, 77-79, 340
dicing with cut commands, 169-171
Digital Equipment Corporation (DEC), 5
DIR command, 60
directories, 42. *See also* files; folders
adding, 459
bin, 42
copying, 111
lib, 43
lists, 65-67
lost+found, 43
mkdir command, formatting, 108-110
mnt, 44
modifying, 107
moving, 54-56
navigating, 126-128
net, 45
permissions
configuring, 88-91
formatting defaults, 100-103
modifying with chmod command, 92-94
present working, 54
removing, 115
renaming, 113
separator characters, 45-46
special, 51-52
sys, 44
tmp, 44
trees, 68-69
usr, 44
disk space
checking available, 77-79
scripts, 342
utilization, 339-342
disk usage, managing, 59
documentation, Perl programming, 435-437
dollar sign ($), 314
domains, 395
dot (.), 49
dot-dot notation, 51, 66
-dprinter flag, 352
du (disk usage) command, 75-77

E

e command, 238
-E flag, 379
-e flag, 303, 320
echo command, 148, 284
echo statements, 54
edit-compile-run cycle, 283
editing
command-line pipes, 171-175
emacs editors, 178, 241
deleting characters, 249-252
file commands, 258-261
Help System commands, 257
inserting text, 242-244
navigating files, 244-248
searching/replacing, 253-256
starting, 242
file permissions, 85
vi editors, 177
! escape-to-Unix command, 232-237
change command, 225-232
colon commands, 214-219
commands, 205
deleting text, 197-205
inserting text, 188-196
moving pages/words, 185-188
navigating, 182-185
optimizing, 209

quitting, 181
replace command, 225-232
searching files, 210-214
searching/replacing, 222-225
starting, 178-182, 219-222
edu domain, 395
-ef flag, 320
egrep command, 158-160
elements
of directory permissions, 72
of permission strings, 71
elif keyword, 322
else keyword, 321
emacs editors, 178, 241
characters, deleting, 249-252
files
commands, 258-261
navigating, 244-248
Help System commands, 257
motion commands, 248
searching/replacing, 253-256
starting, 242
text, inserting, 242-244
email, 385
command-line, sending from, 391-395
globally, sending, 395-400
reading, 386-391
Thunderbird (GNOME), 449-452
entering
commands, 136
passwords, 25
env command, 52-53, 271
environments
commands shells, 271-274
GNOME (GNU Network Object Model Environment), 441-442
applications, 445-452
configuring, 442-445
PRINTER, 352
variables, 53
viewing, 52-53
-eq flag, 318
errors. *See also* troubleshooting
arg list too long, 460
broken pipe, 460
Escape (Esc) key, 178, 206
Exchange program, 428-432
-exec command, 419
execute permissions
directories, 89
files, 84. *See also* permissions
exit command, 26
EXIT signal, 342
export command, 275
expr command, 316, 318
expressions
conditional, programming shells, 321-324
looping, 324-326
regular
egrep command, 158
formatting, 154-157

F

F command, 257
-F flag, 64
-f flag, 320, 355, 371
fg (foreground) command, 295, 299-302
File Browser (GNOME), 446-447
file systems
/ (slash), 42
bin directory, 42
cd command, 54-56
directory separator characters, 45-46
filenames, 47-56
hidden files in Unix, 48-51
hierarchies, 39-45
HOME variable, 53
lib directory, 43
lost+found directory, 43
mnt directory, 44
navigating, 39
PATH variable, 53
pwd command, 54
special directories, 51-52
sys directory, 44
test flags, 319
Thompson, 6
tmp directory, 44
Unix, 41-42
usr directory, 44
viewing, 60-67
filenames, 47-56
mylocate script, 334
suffixes, 63
wildcards, 148-151

files, 123
. (dot), 49
.bashrc, 274
commands, 123-126, 323
emacs editors, 258-261
file types, identifying, 123-126
Unix directories, navigating, 126-128
compress command, 375-376
compression, 79-81
copying, 110-112
counting, 337-339
emacs editors, navigating, 244-248
grep command, 151-154
hidden files in Unix, 48-51
hierarchies, 47
Internet, searching, 422-425
lines, viewing, 458-459
LISTS, 70
lists, 59, 458
modifying, 107
moving, 112-113
naming, 50
overwriting, 113
permissions, 83
applying, 83-104
configuring, 84-88
directory settings, 88-91
formatting defaults, 100-103
identifying owners, 103-104
modifying with chmod command, 94-98
troubleshooting, 458
printing, 351-355
.profile, 274
recovering, 459
redirecting, 142-143
removing, 114-115
renaming, 113-114
searching, 456-457
SFTP (Secure FTP), 407-410
sorting, 67-68
touch command, formatting, 74-75
types, indicators, 84
Unix, 44
vi editors, searching, 210-214
viewing, 60-67, 123-138
applying head program, 128-130
cat program, 131-133
identifying file types, 123-126
more program, 133-138
navigating directories, 126-128
tail program, 130-131
vmunix, 44
filters, 141, 144-148
find command, 334, 455-456
applying, 420-422
navigating, 415-420
Firefox (GNOME), 448-449
flags
-A, 379
-a, 303, 379
-c, 171, 371
combining, 64-65
commands, 73
-d, 319, 379
-d (debug), 438
-dprinter, 352
-E, 379
-e, 303, 320
-ef, 320
-eq, 318
-f, 320, 355, 371
-g, 320
-ge, 318
grep command, 152
-gt, 318
-H, 371
-h, 352, 371
-hheader, 355
-I, 379
-i, 352, 379
-j, 371
-L, 319, 352, 379
-l, 303
-le, 318
lp command, 352
lpr command, 352
ls command, 63, 67-70
-lt, 318
-m, 355, 371
+n, 355
-n, 355
-ne, 318
-nt, 320
-O, 379
-o, 379
-ot, 320
-p, 371
-Pn, 352
-Pprinter, 352

pr command, 355
-R, 215, 352, 379
-r, 320
-s, 320
sort command, 145
-t, 371, 379
-ttitle, 352
-t *xx*, 303
-u, 303
-v, 371, 379
-w, 303, 306
-wn, 355
-X, 371
-x, 303, 306
-Z, 371
-z, 371
flow control, 326
folders. *See also* files
/ (slash), 42
opening, 8
for command, 324
foreground command. *See* fg (foreground) command
formatting
aliases, 63
default permissions, 100-103, 109
directories, mkdir command, 108-110
files, touch command, 74-75
jobs, printing, 355-359
long listing formats, 70-74
ls commands, long listing formats, 70
passwords, 28-30
regular expressions, 154-157
Free Software Foundation, 265
FreeBSD, 265
FTP (File transfer Protocol), 403, 410-413
ftp command, 461
functions
bash (Bourne Again shell), 326-331
comparison, programming shells, 318-321
showdirectory, 330

G

G command, 206
-g flag, 320
games, hi-low, 342-345
-ge flag, 318
General Electric (GE), 5
GET command, 422-425
GNOME (GNU Network Object Model Environment), 4, 8, 225, 441-442
applications, 445-452
configuring, 442-445
GNU Network Object Model Environment. *See* GNOME
GNU Project, 265
graphical interfaces, 4, 8
graphical user interfaces. *See* GUIs
grep command, 151-154, 334, 420, 458
groups, permissions, 103-104
-gt flag, 318
GUIs (graphical user interfaces), 441-442
gzip program, 79-81

H

h command, 206
-H flag, 371
-h flag, 352, 371
head program, 128-130
headers command, 386, 389
help, 9
commands, 386
man pages, 9-16
Unix online reference, 9-16
Help System commands, emacs editors, 257
-hheader flag, 355
hidden files in Unix, 48-51
hierarchies
file systems, 39-45
files, 47
hi-low game, 342-345
history command
navigating, 280-281
shortcuts, 281-286
history of Unix, 5-7
HOME variable, 53, 271

I

I command, 257
i command, 189, 206
-I flag, 379

-i flag, 352, 379
identifying command shells, 267-268
if command, 321
if-then statements, 322
if-then-else statements, 323
indicators, file types, 84
inline editing, command-line pipes, 171-175
inserting
directories, 459
text
emacs editors, 242-244
vi editors, 188-196
insertion commands (vi editors), 195-196
installing CUPS (Common Unix Printing System), 349
interfaces, 4. *See also* shells
CGI (Common Gateway Interface), 438
CUPS (Common Unix Printing System), 349-351
File Browser (GNOME), 446-447
Firefox (GNOME), 448-449
Gnome, 225
Thunderbird (GNOME) for email, 449-452
Internet, searching files, 422-425

J

j command, 206
-j flag, 371
jobs
control, 295
foregrounds/backgrounds, 299-302
printing, formatting, 355-359
processes
stopping, 307-310
viewing, 302-307
stopping, 295-298
Joy, Bill, 178, 264

K

K command, 257
k command, 206
KDE, 4, 8
Kernighan, Brian, 5, 169
keys
Alt, 242
Backspace, 182, 205
Control, 185
cursor-control keys (vi editors), 182-185
Escape (Ecs), 178, 206
Meta, 242
Return, 182, 206
keywords
elif, 322
else, 321
kill command, 307-310
Korn, David, 264
Korn shells. *See* ksh
ksh (Korn shells), 264, 280-281

L

L command, 257
l command, 206
-L flag, 319, 352, 379
-l flag, 64, 303
languages, C, 6
-le flag, 318
lib directory, 43
lines
counting, 143-144
searching/replacing, 222
viewing, 130-131, 458-459
links, symbolic, 63
Linux, 4
package managers, 381-382
lists
directories, 65-67
files, 59, 458
ordered, 52
trees, 68-69
LISTS file, 70
local printers, searching, 348-349
locations, listing, 65-67
logging in/out, 23-27, 271
LOGNAME variable, 53, 273
long listing formats, 70, 71-74
loops
expressions, 324-326
while loops, 326, 344
lost+found directory, 43
lp command, 351
lpadmin command, 348
lpinfo command, 355
lpr command, 351
lprm command, 360

lpstat command, 348-349, 360
ls -1 command, 104
ls command, 41, 52, 60-67
directory trees, listing, 68-69
file type indicators, 84
long listing formats, 70-74
sorting, modifying, 67-68
special flags, 67-70
-lt flag, 318
lynx command, 422-425

M

M command, 257
-m flag, 64, 355, 371
M-a command, 248
Macintosh files
copying, 110
removing, 118
renaming, 114
viewing file types, 123
mail *address* command, 387
MAIL variable, 53, 273
mailx command, 386-391
man pages, 9-16
mkdir, 11-13
navigating, 10
managing
disk usage, 59
Linux package managers, 381-382
M-b command, 248
M-< command, 248
M-> command, 248
M-d command, 252
M-Delete command, 252
M-e command, 248
messages, viewing, 387. *See also* email
Meta key, 242
M-f command, 248
mil domain, 395
M-k command, 252
mkdir command, 108-110
mkdir man page, 11-13
mnt directory, 44
models, concentric access, 94
modifying
chmod command
directories, 92-94
files, 94-98
directories, 107
files, 107
passwords, 27-28
shells, variables, 314
sorting, 67-68
more program, 133-138, 178
motion commands
emacs editors, 248
vi editors, 195-196
moving
directories, 54-56
files, 112-113
pages/words, 185-188
multiuser systems, 7-8
M-v command, 248
mv command, files
moving, 112-113
renaming, 113-114
mylocate script, 334
building, 334-337
disk space utilization, 339-342
hi-low game, 342-345
PATH variables, applying, 337-339

N

N command, 257
n command, 206, 255
+n flag, 355
-n flag, 355
+n notation, 134
naming
directories, 113
file systems, 48-51
filenames, 47-56
files, 50
wildcards, 148-151
navigating
awk programs, 164
bash (Bourne Again shell) configuration files, 274-277
commands
compress, 375-376
cpio, 377-379
prompts, 30-35
shells, 271-274
directories, 126-128
file systems, 39
bin directory, 42
cd command, 54-56
directory separator characters, 45-46
env command, 52-53

filenames, 47-56
hierarchies, 39-45
HOME variable, 53
lib directory, 43
lost+found directory, 43
mnt directory, 44
PATH variable, 53
pwd command, 54
special directories, 51-52
sys directory, 44
tmp directory, 44
Unix, 41-42
usr directory, 44
files, emacs editors, 244-248
find command, 415-420
history command, 280-281
man pages, 10
tar commands, 366-372
unzip command, 373
vi editors, 182-185
zip command, 372-375
-ne flag, 318
net directory, 45
net domain, 395
nG command, 206
noclobber command, 460
notation
dot-dot, 51
egrep command, 158
+n, 134
regular expressions, 154
symbolic, 92
-nt flag, 320
number of lines, viewing, 130-131
numeric comparisons, 318
numeric permissions, calculating strings, 98-100

O

O command, 192, 206
o command, 192, 206
-O flag, 379
-o flag, 379
online documentation, Perl programming, 435-437
open statements, 435
opening folders, 8
OpenWindows, 8
operating systems, history of, 5-7
operators, test, 318
optimizing
rm command, 118-120
vi editors, 209
! escape-to-Unix command, 232-237
change command, 225-232
colon commands, 214-219
replace command, 225-232
searching/replacing, 210-214, 222-225
starting, 219-222
ordered lists, 52
org domain, 395
-ot flag, 320
overwriting files, 113
ownership, permissions, 83, 103-104

P

-p flag, 371
packages
curses, 181
Linux package managers, 381-382
pages, moving, 185-188
passwd command, 27
passwords
entering, 25
formatting, 28-30
modifying, 27-28
PATH variables, 53, 271, 337-339, 459
pattern command, 210
patterns
egrep command, 158
files, listing, 458
searching, 210
performance
disk usage, managing, 59
du (disk usage) command, 75-77
Perl, 169, 427
commands, 438
Exchange program, 428-432
online documentation, 435-437
-w (warnings), 433-435
permissions
defaults, formatting, 100-103, 109
directories
configuring, 88-91
modifying with chmod command, 92-94
removing, 115

files
applying, 83-104
configuring, 84-88
directory settings, 88-91
modifying with chmod command, 94-98
numeric, calculating strings, 98-100
ownership, 83, 103-104
strings, 70-81
troubleshooting, 458
phrases, searching/replacing, 222
pipes (|), 129, 141, 163. *See also* command line, pipes
cut command, 169-171
inline editing, 171-175
sed command, 172-175
tr command, 172-175
plus (+) sign, 220
-Pn flag, 352
power aliases, 288-290
-Pprinter flag, 352
pr command, 355-359
present working directories, 54
print *msgs* command, 387
printenv command, 271
PRINTER environment, 352
printing, 347
CUPS (Common Unix Printing System), 349-351
files, 351-355
jobs, formatting, 355-359
local printers, searching, 348-349
queues, 359-362
processes, jobs
stopping, 307-310
viewing, 302-307
.profile file, 274
programming
C, 6
logging in/out, 23-27
overview of, 3-4
Perl, 169, 427
commands, 438
Exchange program, 428-432
online documentation, 435-437
-w (warnings) flag, 433-435
schedules, 457-458
shells, 313, 333
applying PATH variables, 337-339
arithmetic, 316-317
bash (Bourne Again shell), 326-331
building mylocate scripts, 334-337
comparison functions, 318-321
conditional expressions, 321-324
disk space utilization, 339-342
hi-low game, 342-345
looping expressions, 324-326
variables, 314-316
programs. *See also* commands
awk, 163-169
cat, 131-133
Exchange, 428-432
gzip, 79-81
head, 128-130
more, 133-138, 178
sort, 144-148
suspending, 295-298
tail, 130-131
wc, 143-144
prompts, command
configuring custom, 290-292
navigating, 30-35
protecting shells, 460
protocols
FTP (File transfer Protocol), 403
SFTP (Secure FTP), 407-410
SSH. *See* SSH (secure shell)
ps command, 302-307
PuTTY client, 406
pwd command, 54
python command, 34

Q

q command, 255
query-replace command, 256
queues, printing, 359-362
quit command, 181, 387
quitting vi editors, 181

R

R command, 225, 238
r command, 225, 238

-R flag, 215, 352, 379
-r flag, 320
rc (resource config), 50
read command, 324
read permissions
directories, 89
files, 84. *See also* permissions
reading email, 386-391
recovering files, 459
redirecting files, 142-143
:redo command, 198
regular expressions
egrep command, 158
formatting, 154-157
relative filenames, 47-56
remote Internet connections, 403-405
removing
directories, 115
files, 114-115
renaming files, 113-114
replace command, 225-232
replacing. *See also* searching
emacs editors, 253-256
vi editors, 222-225
reply command, 387
resource config (rc), 50
Return key, 182, 206
revising. *See* editing
Ritchie, Dennis, 5
rm command, 116, 118-120
rmdir command, removing files, 114-115

S

S command, 257
-s flag, 64, 320
save command, 389
save *folder* command, 387
schedules, programming, 457-458
scripts
diskspace, 342
mylocate, 334
applying PATH variables, 337-339
building, 334-337
disk space utilization, 339-342
hi-low game, 342-345
searching, 9. *See also* find command emacs editors, 253-256
files, 456-457
grep command, 151-154
Internet, 422-425
vi editors, 210-214
local printers, 348-349
mylocate script, 334
patterns, 210
vi editors, 222-225
Secure FTP. *See* SFTP
secure shell. *See* SSH
security, 24, 70. *See also* permissions
sed command, 172-175, 459
selecting
command shells, 269-271
passwords, 28-30
sending email
from command-line, 391-395
globally, 395-400
separator characters (directories), 45-46
sets, characters, 172
SFTP (Secure FTP), 407-410
sftp command, 407, 461
sh (Bourne shell), 263
SHELL variable, 53, 271
shells, 4
command, 263. *See also* command shells
jobs, stopping, 295-298
overview of, 8-9
programming, 313, 333
applying PATH variables, 337-339
arithmetic, 316-317
bash (Bourne Again shell), 326-331
building mylocate scripts, 334-337
comparison functions, 318-321
conditional expressions, 321-324
disk space utilization, 339-342
hi-low game, 342-345
looping expressions, 324-326
variables, 314-316
protecting, 460
troubleshooting, 460
shortcuts, history command, 281-286

showdirectory function, 330
shrinking files, compress command, 375-376
SIGHUP signal, 308
SIGINT signal, 308
SIGKILL signal, 308
signals
EXIT, 342
SIGHUP, 308
SIGINT, 308
SIGKILL, 308
SIGTERM, 308
SIGTERM, 308
slash (/), 42, 46
sleep command, 305
slicing with cut command, 169-171
snippets, 14
sort command, 15
sort program, 144-148
sorting, modifying, 67-68
special directories, 51-52
special flags, ls command, 67-70
spell command, 160
square brackets ([ ]), 150
SSH (secure shell)
remote Internet connections, 403-405
third-party SSH connections, 405-406
ssh command, 404, 461
starting
emacs editors, 242
vi editors, 178-182, 219-222
statements
echo, 54
if, 322
if-then, 322
if-then-else, 323
open, 435
status, process values, 305
stopping
jobs, 295-298
processes, 307-310
strings
numeric permissions, calculating, 98-100
permissions, 70-81
suffixes, filenames, 63
suspending programs, 295-298
symbolic links, 63
symbolic notation, 92
sys directory, 44

T

T command, 257
-t flag, 371, 379
-t *xx* flag, 303
tail program, 130-131
tar commands, 366-372, 380
telnet command, 403, 461
TERM variable, 53, 271
test command, 318, 338
test operators, 318
text
emacs editors, inserting, 242-244
lines, viewing, 458-459
vi editors
deleting, 197-205
inserting, 188-196
themes, GNOME, 443
third-party SSH connections, 405-406
Thompson, Ken, 5, 264
Thompson file system, 6
Thunderbird (GNOME), email, 449-452
time command, 35
TLD (top-level domain), 395
tmp directory, 44
top-level domain. *See* TLD
touch command, 74-75
tr command, 172-175
trap command, 342
tree command, 69
trees, listing directories, 68-69
troubleshooting
disk usage, managing, 59
du (disk usage) command, 75-77
Perl code, 433-435
permissions, 458
rm command, 118-120
shells, 460
-ttitle flag, 352
-type file types, 418
types, files
identifying, 123-126
indicators, 84

U

U command, 206
u command, 197, 206
-u flag, 303

umask command, 100-103
undelete *msgs* command, 387
Unity, 4, 8
Unix
file systems, 41-42
hidden files in, 48-51
history of, 5-7
multiuser systems, 7-8
online reference, 9-16
overview of, 3-4
unix file, 44
unzip command, 373
us domain, 395
user environments, 52-53. *See also* environments
USER variable, 271
users command, 33
usr directory, 44
utilities, tar commands, 366-372

V

V command, 257
-v flag, 371, 379
values. *See also* variables
process status, 305
special for system prompts, 290
variables
default, configuring at login, 271
environments, 52-53
HOME, 53, 271
LOGNAME, 53, 273
MAIL, 53, 273
PATH, 53, 271
adding directories, 459
applying, 337-339
SHELL, 53, 271
shells, programming, 314-316
TERM, 53, 271
USER, 271
versions of file command, 124
vi editors, 177
! escape-to-Unix command, 232-237
change command, 225-232
colon commands, 214-219
commands, 205, 238
files, searching, 210-214
insertion commands, 195-196
motion commands, 195-196
navigating, 182-185
optimizing, 209
pages/words, moving, 185-188
quitting, 181
replace command, 225-232
searching/replacing, 222-225
starting, 178-182, 219-222
text
deleting, 197-205
inserting, 188-196
viewing
environments, 52-53
files, 60-67, 123-138
applying head program, 128-130
cat program, 131-133
identifying file types, 123-126
more program, 133-138
navigating directories, 126-128
tail program, 130-131
hidden files, 49
lines, 458-459
messages, 387
processes, 302-307
vmunix file, 44

W

W command, 206, 257
w command, 34-35, 206
-w flag, 303, 306
-w (warnings) flag, Perl programming, 433-435
wc program, 143-144
Weinberger, Peter, 169
whatis command, 14
while loops, 326, 344
who command, 185, 356
whoami command, 31-33
wildcards, 141, 148-151
Windows, files
copying, 110
removing, 118
renaming, 114
viewing file types, 123
-wn flag, 355
words
counting, 143-144
deleting, 249-252

moving, 185-188
searching/replacing, 222
write permissions
directories, 90
files, 84. *See also* permissions

X

x command, 206
-X flag, 371
-x flag, 67, 303, 306
xargs command, 420, 455-456

Y

y command, 255

Z

-Z flag, 371
-z flag, 371
zip command, 372-375