

Check for Updates

Make sure you have the latest information!



TidBITS Publishing Inc.

Take Control of

v1.0.2

The Mac
Command
Line
with
Terminal

Joe Kissell

\$10

[Help](#)

[Catalog](#)

[Feedback](#)

[Order Print Copy](#)

Table of Contents

Skip to next section.

READ ME FIRST 5

Updates	5
Basics	5

INTRODUCTION 8

MAC OS X COMMAND LINE QUICK START 11

UNDERSTAND BASIC COMMAND-LINE CONCEPTS 12

What's Unix?	12
What's a Command Line?	13
What's a Shell?	14
What's Terminal?	15
What Are Commands, Arguments, and Flags?	16

GET TO KNOW (AND CUSTOMIZE) TERMINAL 20

Learn the Basics of Terminal	20
Modify the Window	22
Open Multiple Sessions	22
Change the Window's Attributes	23
Set a Default Shell	26

LOOK AROUND 29

Discover Where You Are	29
See What's Here	29
Repeat a Command	31
Cancel a Command	33
Move into Another Directory	33
Jump Home	35
Understand How Paths Work	36
Understand Mac OS X's File System	38
Use Tab Completion	40
Find a File	42
View a Text File	44
Get Help	45
Clear the Screen	47
End a Shell Session	47

WORK WITH FILES AND DIRECTORIES 48

Create a File.....	48
Create a Directory.....	49
Copy a File or Directory	49
Move or Rename a File or Directory	51
Delete a File.....	53
Delete a Directory.....	54

WORK WITH PROGRAMS 55

Learn Command-Line Program Basics	55
Run a Program or Script	57
Run a Program in the Background	60
See What Programs Are Running.....	61
Stop a Program	65
Edit a Text File	66
Create Your Own Shell Script.....	68

CUSTOMIZE YOUR PROFILE 71

How Profiles Work.....	71
Edit .bash_profile.....	72
Create Aliases	72
Modify Your PATH	73
Change Your Prompt	74

BRING THE COMMAND LINE INTO THE REAL WORLD 75

Get the Path of a File or Folder	75
Open the Current Directory in the Finder.....	76
Open a Hidden Directory Without Using Terminal	77
Open the Current Folder in Terminal.....	77
Open a Mac OS X Application.....	79
Open a File in Mac OS X.....	79

LOG IN TO ANOTHER COMPUTER 80

Start an SSH Session	80
Run Commands on Another Computer	82
End an SSH Session	82

VENTURE A LITTLE DEEPER 83

Understand Permission Basics.....	83
Change an Item's Permissions	86
Change an Item's Owner or Group	87
Perform Actions as the Root User	88

COMMAND-LINE RECIPES 91

Change Defaults	91
Perform Administrative Actions	94
Modify Files	95
Work with Information on the Web	97
Manage Network Activities	98
Work with Remote Macs	101
Troubleshoot and Repair Problems	102
Get Help in Style	104
Do Other Random Tricks	105

ABOUT THIS BOOK 108

About the Author	108
Author's Acknowledgments	108
Shameless Plug	109
About the Publisher	109
Production Credits	109

COPYRIGHT AND FINE PRINT 110

Read Me First

Welcome to *Take Control of the Mac Command Line with Terminal*, version 1.0.2, published in April 2009 by TidBITS Publishing Inc. This book was written by Joe Kissell and edited by Geoff Duncan.

This book introduces you to Mac OS X's command line environment, teaching you how to use the Terminal utility to accomplish useful, interesting tasks that are either difficult or impossible to perform in the graphical interface.

Copyright © 2009, Joe Kissell. All rights reserved.

If you have the PDF version of this title, please note that if you want to share it with a friend, we ask that you do so as you would a physical book: “lend” it for a quick look, but ask your friend to buy a new copy to read it more carefully or to keep it for reference. You can click [here](#) to give your friend a discount coupon. Discounted [classroom and Mac user group copies](#) are also available.

UPDATES

We may offer free minor updates to this book. To read any available new information, click the Check for Updates link on the [cover](#) or click [here](#). On the resulting Web page, you can also sign up to be notified of major updates via email. If you own only the print version of the book or have some other version where the Check for Updates link doesn't work, contact us at tc-comments@tidbits.com to obtain the PDF.

BASICS

In reading this book, you may get stuck if you don't know certain fundamental facts about your Mac or if you don't understand Take Control syntax for things like working with menus or finding items in the Finder.

Please note the following:

- **Menus:** Where I describe choosing a command from a menu in the menu bar, I use an abbreviated description. For example, the abbreviated description for the menu command that selects everything in the frontmost window is “Edit > Select All.”
- **Finding System Preferences:** I sometimes refer to settings in System Preferences that you may want to adjust. To open System Preferences, click its icon in the Dock or choose  > System Preferences. When the System Preferences window opens, click an icon to display a corresponding settings pane. I refer to these panes using an abbreviated notation such as “the Sharing preference pane.”
- **Path syntax:** This book often uses a *path* to show the location of a file or folder in your file system. For example, Leopard stores most utilities, such as Terminal, in the Utilities folder. The path to Terminal is: `/Applications/Utilities/Terminal`.

The slash at the beginning of the path tells you to start at the root level of the disk. You will also encounter paths that begin with `~` (tilde), which is a shortcut for the user’s home folder. For example, if a person with the user name `joe` wants to install fonts that only he can access, he would install the fonts in his `~/Library/Fonts` folder, which is just another way of writing `/Users/joe/Library/Fonts`.

- **Folders and directories:** In the Finder, you organize files into *folders*, but the term *directory* is more common in the command-line world. They have more or less equivalent meanings, except that folders are visible in the Finder and have icons that *look* like folders, while directories may not appear in the Finder at all. In this book, I say “folder” when talking about the Finder, and “directory” when talking about the command line. When we’re in a Terminal window, I may refer to your “home directory,” but in the context of the Finder I would call the same location your “home folder.”
- **Line breaks:** This book contains many examples of text that you must type into a Terminal window; these appear in a special font. If you’re viewing this book on an iPhone or other device with a narrow screen, this text may wrap oddly. Common sense is the best policy: if something looks like it should all be on one line, it probably should. Don’t add extra line breaks to match the book’s display.

- **Entering commands:** I frequently tell you to “enter” commands in a Terminal window. This means you should type the command and then press Return or Enter. Typing a command without pressing Return or Enter has no effect.
- **Getting commands into Terminal:** When you see commands that are to be entered into a Terminal window, you can type them manually. If you’re reading this on a Mac, you can copy the command from the PDF and paste it into Terminal (which is handy, especially for longer and more complex commands). Whichever method you use, keep these tips in mind:
 - **When typing:** Every character counts, so watch carefully. The font that represents text you should type is *monospaced*, meaning every character has the same width. So, if it looks like there’s a space between two characters, there is—and you should be sure to type it. Similarly, be sure to type all punctuation—such as hyphens and quotation marks—exactly as it appears in the book, even if it seems odd. If you type the wrong thing, the command probably won’t work.
 - **When copying and pasting:** If you select a line of text to copy and paste into Terminal, be sure that your selection begins with the first character and ends with the last. If you accidentally leave out characters, the command probably won’t work, and if you select too much (for example, extending your selection to the next line), you may see unexpected results, such as the command executing before you’re ready.

WHAT’S NEW IN VERSION 1.0.2

This version fixes two typos—the keystrokes to move backward by a screen in `less` ([More or Less](#), p. 44) and uncut a line in `nano` (“[Uncut](#)” tip, p. 68). It also includes the changes in version 1.0.1—correcting problems with copying and pasting from the ebook into Terminal, printing on some printers, and minor errors on pages 41, 54, and 96.

Does your printer prefer wide margins? *When printing, to prevent page numbers or right margins from cutting off (a few pages, such as page 99, have skinny margins), try printing with a reduction, perhaps at 90 percent, instead of full size.*

Introduction

Back when I began using computers, in the early 1980s, user interfaces were pretty primitive. A computer usually came with only a keyboard for input—mice were a novelty that hadn't caught on yet. To get your computer to do something, you typed a command, waited for some result, and then typed another command. There simply was no concept of pointing and clicking to make things happen.

When I finally switched from DOS to the Mac (without ever going through a Windows phase, I should mention!), I was thrilled that I could get my work done without having to memorize lists of commands, consult manuals constantly, or guess at how to accomplish something. Everything was right there on the screen, just a click away. It was simpler—not in the sense of being less powerful, but in the sense of requiring less effort to access the same amount of power. Like most everyone else, I fell instantly in love with graphical interfaces.

Fast forward a couple of decades, and I find myself faced with some mundane task, such as renaming all 500 files in a folder to use a different extension, deleting a file that refuses to disappear from the Trash, or changing an obscure system preference. After wasting some time puzzling over how to accomplish my task—and perhaps doing some Web searches—I finally discover that Mac OS X's graphical interface does not, in fact, offer any built-in way to do what I want. So I have to hunt on the Internet for an application that seems to do what I want, download it, install it, and run it (and perhaps pay for it, too), all so that I can accomplish a task with my mouse that would have taken me 5 seconds in DOS 25 years ago.

That's not simple.

I'm a Mac user because I don't have time to waste. I don't want my computer to put barriers between me and my work. I want easier ways to do things instead of harder ways. Ironically, Mac OS X's beautiful Aqua graphical interface, with all its menus, icons, and buttons, doesn't always provide the easiest way to do something, and in some cases it doesn't even provide a hard way. The cost of elegance and simplicity is sometimes a lack of flexibility.

Luckily, Mac OS X isn't restricted to the graphical realm of windows and icons. It has another whole interface that lets you accomplish many tasks that would otherwise be difficult, or even impossible. This other way of using Mac OS X looks strikingly like those DOS screens from the 1980s: it's a command-line interface, in which input is done with the keyboard, and the output is sent to the screen in plain text.

The usual way of getting to this alternative interface (though there are others) is to use a program called Terminal, located in the Utilities folder inside your Applications folder. It's a simple program that doesn't appear to do much at first glance—it displays a window with a little bit of text in it. But Terminal is in fact the gateway to vast power.

If you read *TidBITS*, *Take Control* books, *Macworld*, or any of the numerous other publications about the Mac, you've undoubtedly seen tips and tricks from time to time that begin, "Open Terminal and type in the following...". Many Mac users—especially those without prior experience in DOS or Unix—find that sort of thing intimidating. What do I click on? How do I find my way around? How do I stop something I've started? Without the visual cues of a graphical interface, lots of people get stuck staring at that blank window, frustrated that they can't accomplish whatever task they're trying to perform.

If you're one of those people, this book is for you. It's also for people who know a little bit about the command line—perhaps just enough to be dangerous—but don't fully understand what they can do, how to get around, and how to stay out of trouble. By the time you're finished reading this book and trying out the examples I give, you should be comfortable interacting with your Mac by way of the command line, ready to confidently use Terminal whenever the need arises.

It's not scary. It's not hard. It's just different. And don't worry—I'll be with you every step of the way!

Much of this book is concerned with teaching you the skills and basic commands you must know in order to accomplish genuinely useful things later on. If you feel that it's a bit boring or irrelevant to learn how to list files or change directories, remember: it's all about the end result. You learn the fundamentals of baking not because measuring flour or preheating an oven is intrinsically interesting, but because you need to know how to do those things in order to end up with cookies. And let me tell you, the cookies make it all worthwhile!

Speaking of food—my all-purpose metaphor—this book doesn't *only* provide information on individual ingredients and techniques. The last section is full of terrific, simple command-line recipes that put all this power to good use while giving you a taste of some advanced capabilities I don't explore in detail. Among other things, you'll learn:

- How to figure out what's preventing a disk from disconnecting (unmounting or ejecting)
- How to tell which applications are currently accessing the Internet
- How to rename lots of files at once
- How to change a number of hidden preferences
- How to understand and change file permissions
- How to automate command-line activities with scripts

Astute readers may note that some of these tasks can be accomplished with third-party utilities (most of which simply carry out command-line tasks in response to a mouse click). That's true, but the command line is infinitely more flexible—and Terminal is free! It's like the difference between buying supermarket cookies and being able to bake your own—in any variety, and in any quantity. Sure, there's a place for prepackaged solutions, but it's often quicker, easier, and more effective just to type a command into Terminal.

I should be clear, however, that this book won't turn you into a command-line expert. I would need thousands of pages to list everything you can accomplish using the command line. Instead, my goal is to cover the basics and get you up to a moderate level of familiarity and competence. I may not answer every question you have, but you should get a solid foundation and be able to figure out how to learn more. I'll take your feedback into account, too: if there's sufficient interest, I may expand on this information in a future version of this book (or another Take Control title).

Most of the examples in this book work with any version of Mac OS X, but a few of them require Mac OS X 10.5 Leopard or newer. If you're following along in Mac OS X 10.4 Tiger or earlier, you'll notice that the Terminal application isn't identical—it omits tabs and some other customization options—but mostly works the same.

Mac OS X Command Line Quick Start

This book is almost entirely linear—later sections build on earlier sections. I strongly recommend starting from the beginning and working through the book in order (perhaps skimming lightly over any sections that explain already-familiar concepts). You can use the items in the final section, [Command-Line Recipes](#), at any time, but they'll make more sense if you understand all the basics presented earlier in the book.

Find your bearings:

- Learn about the command line and its terminology; see [Understand Basic Command-Line Concepts](#) (p. 12).
- Become familiar with the most common tool for accessing the command line; see [Get to Know \(and Customize\) Terminal](#) (p. 20).
- Navigate using the command line; see [Look Around](#) (p. 29).

Learn basic skills:

- Create, delete, and modify files and directories; see [Work with Files and Directories](#) (p. 48).
- Run or stop programs and scripts; see [Work with Programs](#) (p. 55).
- Make your command-line environment work more efficiently; see [Customize Your Profile](#) (p. 71).

Go beyond the fundamentals:

- Integrate the command line and Mac OS X's graphical interface; see [Bring the Command Line into the Real World](#) (p. 75).
- Use the command line to control another Mac; see [Log In to Another Computer](#) (p. 80).
- Learn some slightly advanced techniques; see [Venture a Little Deeper](#) (p. 83).
- Do cool stuff; see [Command-Line Recipes](#) (p. 91).

Understand Basic Command-Line Concepts

In order to make sense of what you read about the command line, you should know a bit of background material. This section explains the ideas and terminology I use throughout the book, providing context for everything I discuss later in the book.

WHAT'S UNIX?

Unix is a computer operating system with roots going back to 1969. Back then, Unix referred to one specific operating system running on certain expensive minicomputers (which weren't "mini" at all: they were enormous!). Over time, quite a few companies, educational institutions, and other groups have developed their own variants of Unix—some were offshoots from the original version and others were built from scratch.

After many branches, splits, mergers, and parallel projects, there are now more than a dozen distinct families of Unix and Unix-like operating systems. Within each family, such as Linux (a Unix-like system), there may be many individual variants, or distributions.

Note: A Unix-like system is one that looks and acts like Unix, but doesn't adhere completely to a list of standards known as the Single UNIX Specification, or SUS. Mac OS X 10.5 Leopard and newer (including the Server versions) are true Unix operating systems when running on Intel-based Macs. Earlier versions of Mac OS X, and current versions running on PowerPC-based Macs, are technically Unix-like.

Mac OS X is a version of Unix that nicely illustrates this process of branching and merging. On the one hand, you had the classic Macintosh OS, which developed on its own path between 1984 and 2002. On the other hand, you had NeXTSTEP, an operating system

based on a variety of Unix called BSD (Berkeley Software Distribution). NeXT was the company Steve Jobs founded after leaving Apple in 1985.

When Apple bought NeXT in 1996, it began building a new operating system that extended and enhanced NeXTSTEP while layering on capabilities (and some of the user interface) of the classic Mac OS. The result was Mac OS X: it's Unix underneath, but with a considerable amount of extra stuff that's not in other versions of Unix. If you took Mac OS X and stripped off the graphical interface, the programming interfaces (Cocoa, Carbon, and Java), and all the built-in applications such as Mail and Safari, you'd get the Unix core of Mac OS X. This core has its own name: Darwin. When you work in the command-line environment, you'll encounter this term from time to time.

Darwin is itself a complete operating system, and although Apple doesn't sell computers that run only Darwin, it is available as open source so anyone with sufficient technical skill can download, compile, and run Darwin as an operating system on their own computer—for free. Darwin is, in a sense, OS X without the Mac part.

WHAT'S A COMMAND LINE?

A command-line interface is a way of giving instructions to a computer and getting results back. You type a *command* (a word or other sequence of characters) and press Return or Enter. The computer then processes that command and displays the result (often in a list or other chunk of text). In most cases, all your input and output remains on the screen, scrolling up as more appears. But only one line—usually the last line of text in the window, and usually designated by a blinking cursor—is the actual *command line*, the one where commands appear when you type them.

Note: Although Darwin (which has only a command-line interface) is part of Mac OS X, it isn't quite correct to say that you're working in Darwin when you're using the Mac OS X command line. In fact, the command line gives you a way of interacting with *all* of Mac OS X, only part of which is Darwin.

WHAT'S A SHELL?

A *shell* is a program that creates a user interface of one kind or another, enabling you to interact with a computer. In Mac OS X, the Finder is a type of shell—a graphical shell—and there are still other varieties with other interfaces. But for the purposes of this book, I use the term “shell” to refer only to programs that create a command-line interface.

Mac OS X includes six different shells, which means that your Mac has not just one command-line interface, but six! These shells share many attributes—in fact, they’re more alike than different. Most commands work the same way in all the shells, and produce similar results. The shells in Mac OS X are all standard Unix shells, and at least one of them is on pretty much any computer running any Unix or Unix-like operating system.

The original Unix shell was called the Bourne shell (after its creator, Stephen Bourne). The actual program that runs the Bourne shell has a much shorter name: `sh`. The other Unix shells included with Mac OS X are:

- **`cs`h**: the C shell, named for similarities to the C programming language (Unix folks love names with puns, too, as you’ll see)
- **`tc`sh**: the Tenex C shell, which adds features to `cs`h
- **`k`sh**: the Korn shell, a variant of `sh` (with some `cs`h features) developed by David Korn
- **`ba`sh**: the Bourne-again shell (yet another superset of `sh`)
- **`z`sh**: the Z shell, an advanced shell named after Yale professor Zhong Shao that incorporates features from `tc`sh, `k`sh, and `ba`sh, plus other capabilities

In Mac OS X 10.2 Jaguar and earlier versions, `tc`sh was the default shell. Starting with Mac OS X 10.3 Panther, `ba`sh became the new default. Even if you’re running a later version of Mac OS X, though, your account may still be configured to use `tc`sh if you upgraded (or migrated) from Mac OS X 10.2 or older.

In this book, I discuss only the `bash` shell. Some may argue `zsh` has a superior feature set or `tcsh` is more universal—and I can't particularly disagree—but because `bash` is the current default and can easily handle everything I want to show you about the command line, that's what we'll be sticking with here.

A bit later in the book, in the section [Set a Default Shell](#), I show you how to confirm that you're using the `bash` shell and how to change your default, if you like.

WHAT'S TERMINAL?

So, how do you run a shell in order to use a command-line interface on your Mac? You use an application called a *terminal emulator*. As the name suggests, a terminal emulator simulates a *terminal*—the devices people used to interact with computers back in the days of monolithic mainframes. A terminal consisted of little more than a display (or, even earlier, a printer), a keyboard, and a network connection of sorts. Terminals may have looked like computers, but all they did was receive input from users, send it along to the actual computer (which was likely in a different room or even a different building), and display any results that came back. A modern terminal emulator program provides a terminal-like connection to a shell running either on the same computer or on a different computer over a network.

Quite a few terminal emulators run on Mac OS X, but the one you're most likely to use is called—you guessed it—Terminal, and it's included as part of Mac OS X. Although you're welcome to find and use a different terminal emulator if that's your preference, in this book I discuss only Terminal.

Note: At the risk of redundancy, I want to emphasize where Terminal fits in the scheme of things. A common misconception is that Terminal *is* the Mac OS X command-line interface. You'll hear people talk about entering "Terminal commands" and things of that sort. (Even I have said things like that from time to time.) But that's incorrect. Terminal is just a program—one of numerous similar programs—that gives you access to Mac OS X's command-line interface. When you run a command-line program, you're running it in a shell, which in turn runs in Terminal.

So, to summarize: you use Terminal to run a shell, which provides a command-line interface to Mac OS X—a variety of Unix (of which the non-graphical portion is known as Darwin). You can use the Mac OS X command line successfully without having all those facts entirely clear in your mind, but a rough grasp of the hierarchy makes the process a bit more comprehensible.

WHAT ARE COMMANDS, ARGUMENTS, AND FLAGS?

The last piece of background information I want to provide has to do with the kinds of things you type into a Terminal window. I provide extensive examples of all these items ahead, but I want to give you an introduction to three important terms: *commands*, *arguments*, and *flags*. If you don't fully understand this stuff right now, don't worry: it will become clearer after some examples.

Commands

Commands are straightforward; they're the verbs of the command line (even though they may look nothing like English verbs). When you enter a command, you tell the computer to do something, such as run a program. Very often, entering a command—a single word or abbreviation—is sufficient to get something done.

You may enter: *As a reminder, in a command-line interface, merely typing a command does nothing; the command doesn't execute until you press Return or Enter. So, throughout this book, when I say to "enter" a command, what I mean is: Type this, and then press Return or Enter.*

For example—not to get ahead of myself but just to illustrate—if you enter the command `date`, your Terminal window shows the current date and time.

Note: Many commands are abbreviations or shortened forms of longer terms—for example, the command `pwd` stands for Print Working Directory.

Arguments

Along with commands (verbs), we have arguments, which you can think of as nouns—or, in grammatical terms, direct objects. For example, I could say to you, “Eat!,” and you could follow that command by consuming any food at hand. However, if I want you to eat something in particular, I might say, “Eat cereal!” Here, *cereal* is the direct object, or what we’d call an argument in a command-line interface.

On the command line, you must frequently specify the file, directory, or other item to which you want a command applied. In general, you simply type the command, a space, and then the argument. For example, the command `nano`, by itself, opens a text editor called `nano`. (In other words, entering `nano` means “*run nano*”—you tell the shell to execute a command simply by entering its name.) But enter `nano file1` and the command instead opens the file `file1` using the `nano` text editor. Here, `file1` is the argument to the command `nano`.

Space ranger: *Always be sure to type a space after the command and before any arguments.*

Some commands accept no arguments. Some take optional arguments. And some commands require one or even several arguments. For example, to change the modification date of three files—`file1`, `file2`, and `file3`—I can enter `touch file1 file2 file3`. But other commands require multiple arguments that have different meanings (as in “Process `file1` with the information found in `file2` and store the output in `file3`”). In these cases, the order in which the arguments appear is critical. I detail which commands in this book take arguments, the order of those arguments, and the circumstances when you need to use those arguments.

Flags

Besides verbs and nouns, we also have adverbs! In English, I could say, “Eat cereal *quickly*!” or “Watch TV *quietly*.” The adverbs *quickly* and *quietly* don’t tell you what to do, but rather how to do it. By analogy, an expression in a command-line statement that specifies how a command should be accomplished is called a flag, though you may also hear it referred to as an *option* or *switch*. (Some people consider a flag to be a type of argument, but I’m going to ignore that technicality.)

Suppose I want to list the files in a directory. I could enter the `ls` (list) command, which would do just that. But if I want to list the files in a particular way—say, in a way that included their sizes and modification dates—I could add a flag to the `ls` command. The flag that `ls` uses to indicate a “long” listing (including sizes and dates) is `-l`. So if I enter `ls -l` (note the space before the flag), I get the kind of listing I want.

Flagging Enthusiasm

I should mention a couple of irritations with flags:

- First, you’ll notice in this example that the flag was preceded by a hyphen: `-l`. That’s common, and it enables the command to distinguish a flag (which has a hyphen) from an argument (which doesn’t).

Unfortunately, Unix commands aren’t entirely consistent: you’ll sometimes see commands that require flags with *no* hyphen, commands that require flags with *two* hyphens, and commands with flags that can appear in either a “short” form (one hyphen, usually followed by a single letter) or a “long” form (two hyphens, usually followed by a complete word).

- Second, a command may take more than one flag. (“Eat quickly and quietly!”) For example, you might want to tell the `ls` command not only to use the long format (`-l`) but also to show all files, including any hidden ones (`-a`). Here you get two choices. You can either combine the flags (`ls -la` or `ls -al`) or keep them separate (`ls -l -a` or `ls -a -l`). In this example, both ways work just fine, and the flags work in any order. But that isn’t always the case; some commands are picky and require you to list flags one way or the other.

Don’t worry about these differences; just be aware that they may come up from time to time. For now, assume that most flags will start with a single hyphen, and that the safest way to express most flags is to keep them separate.

Some commands require both arguments and flags. In general, the order is `command flag(s) argument(s)`, which is unlike usual English word order—it would be comparable to saying, “Eat quickly cereal!” For example, if you want to use the `ls` (list) command to show you only the names of files beginning with the letter `r` (`r*`), in long (`-l`) format, you’d put it like this: `ls -l r*`.

Sin Tax?

As you read about the command line, you'll sometimes see the word *syntax*, which is a compact way of saying, "which arguments and flags are required for a given command, which are optional, and what order they should all go in." When I say that the usual order is `command flag(s) argument(s)`, I'm making a general statement about syntax, though there are plenty of exceptions.

One place you see a command's syntax spelled out is in the `man` (manual) pages for Unix programs (see [Get Help](#)), at the top under the heading "Synopsis." For example, the `man` page for the `mkdir` (make directory) command (see [Create a Directory](#)) gives the following:

```
mkdir [-pv] [-m mode] directory_name ...
```

Here's how to read this command's syntax, one item at a time (don't worry about exactly what each item does; this is just for illustration):

- `mkdir`: First is the command itself.
- `[-pv]`: Anything in brackets is optional, and flags are run together in the syntax if they can be run together when using the command. So we know that the `-p` flag and the `-v` flag are both optional, and if you want to use them both they can optionally be written as `-pv`.
- `[-m mode]`: Another optional flag is `-m`, and it's listed separately because if you do use it, it requires its own argument (another string of characters, described in the `man` page). The underline beneath `mode` means it's a variable; you have to fill in the mode you want.
- `directory_name`: This argument is not optional (because it's not in brackets), and it's also a variable, meaning you supply your own value.
- `...`: Finally, we have an underlined ellipsis, which simply means you can add on more arguments like the last one. In this case, it would mean you could list additional directories to be created.

So the final command could look like, for example:

```
mkdir teas (all optional items omitted), or
```

```
mkdir -pv -m 777 a/b/teas a/b/nuts (all optional items included).
```

Get to Know (and Customize) Terminal

As I mentioned in [What's Terminal?](#), the application you're most likely to use for accessing the command line in Mac OS X is Terminal. Since you'll be spending so much time in this application, a brief tour is in order. In addition, you may want to adjust a few settings, such as window size, color, and font, to whatever you find most comfortable and easy to read.

LEARN THE BASICS OF TERMINAL

The moment has arrived. Find the Terminal application (inside [/Applications/Utilities](#)), double-click it, and take a Zen moment to contemplate the emptiness (**Figure 1**).

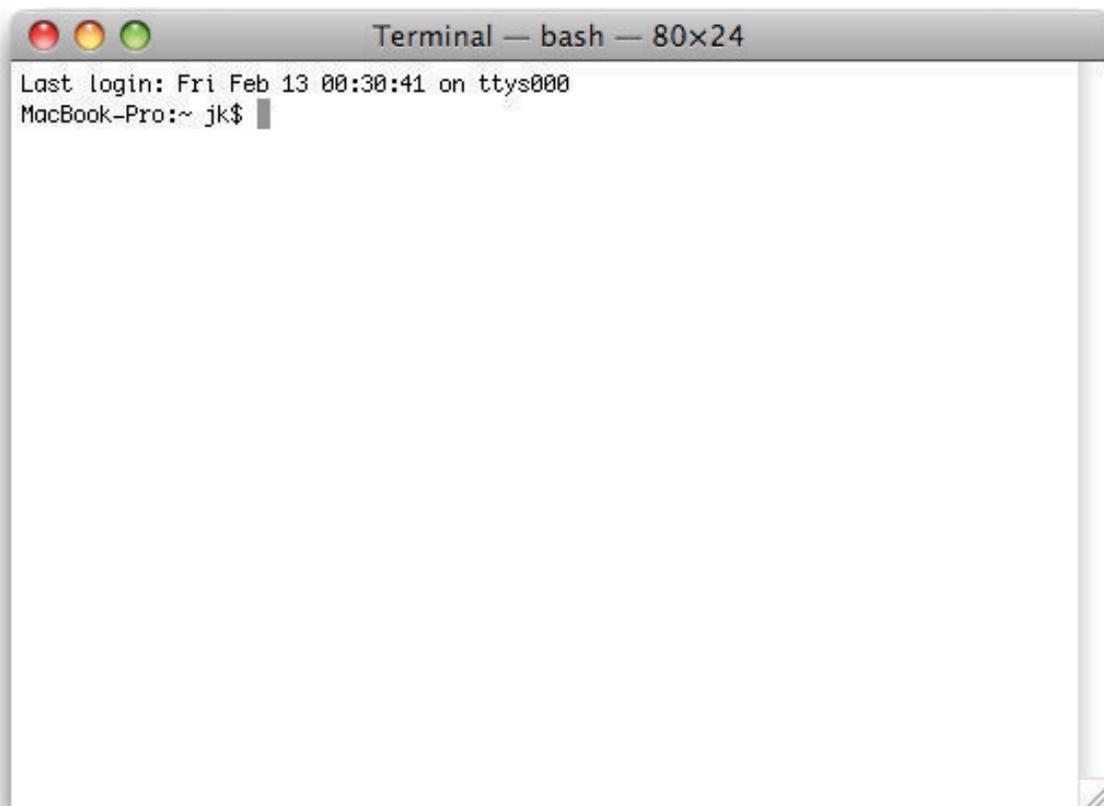


Figure 1: The Terminal window harks back to pre-graphical days.

To state the obvious, it's a (mostly) empty window. A Terminal window simply shows a command-line interface generated by a shell (in this case, the `bash` shell). As long as you're in this window, you can forget about your mouse: with a couple of notable exceptions that I cover later, everything you do here uses the keyboard only.

Of course, the window isn't *completely* empty. The first line lists, by default, the date and time of your last login. In this example, it's:

```
Last login: Fri Feb 13 00:30:41 on ttys000
```

That last part, `on ttys000`, is a bit of esoteric information that signifies the terminal interface with which you logged in the last time. It might say something different (such as `on console`) or nothing at all—for all practical purposes, you can safely ignore this line.

The second line is the actual command line (the line on which you type commands):

```
MacBook-Pro:~ jk$ █
```

The rectangular box at the end (which may instead appear as a vertical line or an underscore, any of which may or may not blink) is the *cursor* (not to be confused with your *pointer*, which reflects mouse movement). Everything before the cursor is known as the *prompt*, which is to say it's prompting you to type something.

The first part of the prompt, `MacBook-Pro`, is the name of my computer (spaces are replaced with hyphens, and punctuation, if any, usually disappears). The colon (`:`) is simply a visual separator. Next is the tilde (`~`), which signifies that I'm currently in my home directory (which, for me, is `/Users/jk`). The `jk` is the short user name of the account under which I'm logged in. And finally, the `$` signifies that I'm logged in as an ordinary (non-root) user. (I say more about the `$` in the sidebar [The \\$, #, and Other Strange Things on My Command Line](#), ahead.) If your short user name is `cindy` and your computer's name (as shown in the Sharing pane of System Preferences) is `Cindy's Groovy iMac`, your command line may look something like this:

```
Cindys-Groovy-iMac:~ cindy$ █
```

All these things are customizable; see [Customize Your Profile](#).

MODIFY THE WINDOW

The window you're looking at is just like any other Mac OS X window. You can move it, minimize it, resize it, zoom it, scroll through its contents, and hide it using the usual controls. So please do adjust it to your liking. However, I want to make two important points about window modification:

- First, resizing isn't only a good idea, it's practically mandatory. Some commands you run in this window will generate a lot of text, including some large tables, and you'll find it much easier to work in the command line if your Terminal window is a bit bigger. Go ahead and make the window as large as you want—but do leave at least a bit of space so that you can see some parts of other windows on your screen.
- Second, any changes you make to the window ordinarily last only until you close it. If you open a new window—or quit Terminal and launch it again later—you're returned to the defaults. So, once you get your Terminal window to a size, shape, and position you like, choose Shell > Use Settings as Default. Thereafter, all new Terminal windows you open use your preferred characteristics. (I say more about customizing windows ahead, in [Change the Window's Attributes.](#))

OPEN MULTIPLE SESSIONS

Most applications can have multiple windows open at once—think of your word processor, your Web browser, or your email program, for example. The same is true of Terminal—you can have as many windows open as you need, each with its own command line. To open a new window, press Command-N.

When you open a new window in Terminal, you begin a new *session*. That means another copy of the shell runs, separate from the first copy. You can run a program or navigate to a location in the first session, and run a completely different program or navigate to another location in the second. The two sessions don't normally interact at all; it's as though you're using two different computers at once that happen to share the same set of files.

Why would you want to do this? Perhaps you want to refer to a program's man (manual) page in one window, while trying out the command in a second. Perhaps one shell is busy performing some lengthy task and you want to do something else at the same time. Or perhaps you want to compare the contents of two directories side by side. Whatever the case, remember: you're not limited to using one window—or one session—at a time.

But wait, there's more! As long as you're running Mac OS X 10.5 Leopard or newer, every window in Terminal also supports multiple tabs—just like most Web browsers (**Figure 2**). So if you want to have multiple sessions open without the screen clutter of multiple windows, you can do so easily. Create a new tab by pressing Command-T. Exactly as in a browser, you can drag tabs to rearrange them, close them individually, and even drag a tab from one window to another.

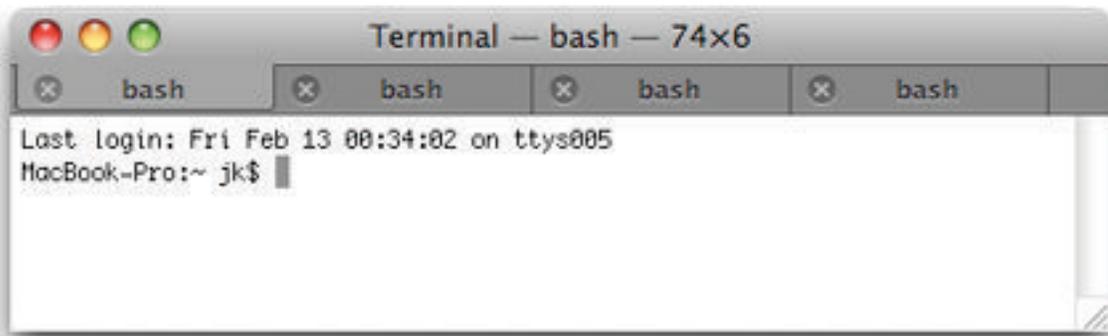


Figure 2: Terminal windows can have multiple tabs, which can be moved and closed individually just like those in most Web browsers.

CHANGE THE WINDOW'S ATTRIBUTES

Moving and resizing windows is one thing, but Terminal lets you go much further. You can change the background color (and transparency), font (typeface and size), text color, cursor type, and numerous other settings. In fact, you can change far more attributes than I care to describe here, so I want to explain just a few of the basics.

For starters—just to get a feel for what's possible—choose Shell > New Window (or New Tab) and try some of the prebuilt themes. For example, choose Shell > New Window > Homebrew for a display

with bright green text in 12-point Andale Mono against a slightly transparent black background. Or choose Shell > New Window > Grass for pale yellow text, in bold 12-point Courier, on an opaque green background, with a red cursor. **Figure 3** shows several examples.

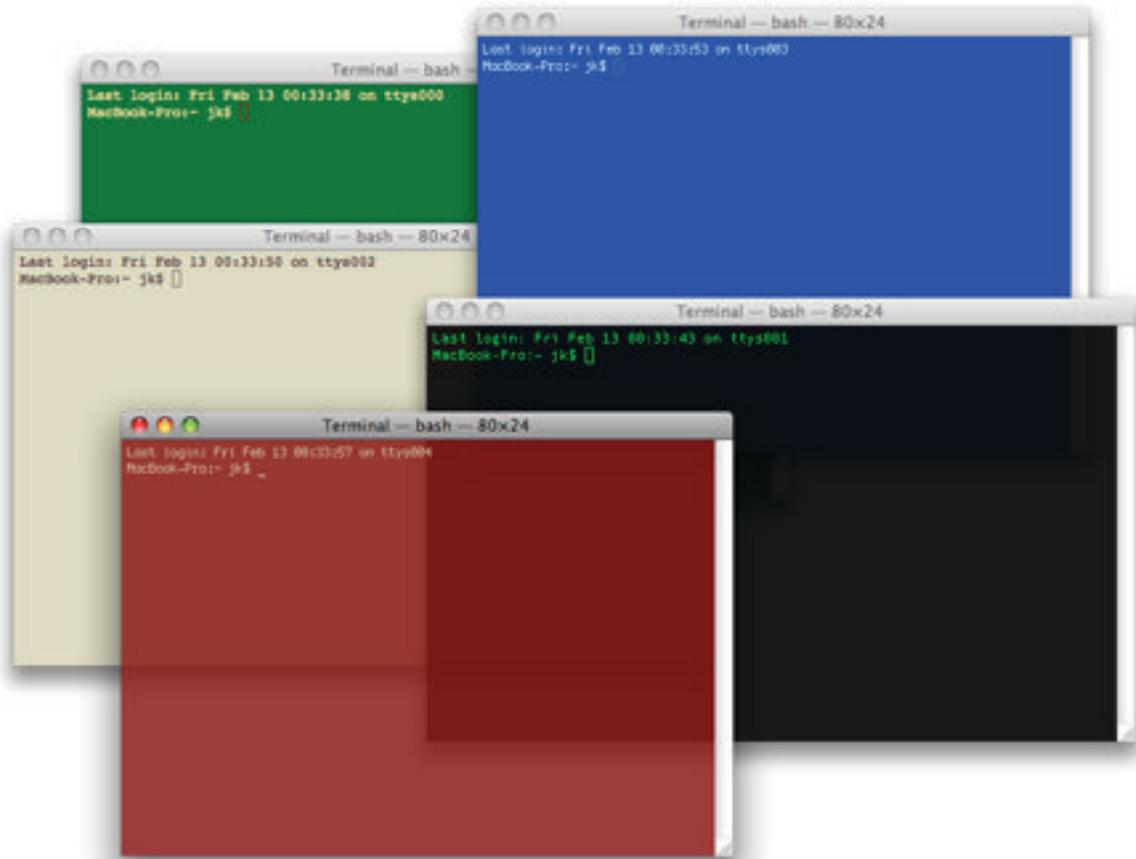


Figure 3: Terminal windows can take on many themes; this image shows several of the stock themes.

If you prefer to use one of these other themes as your default, open a new window with that theme and choose Shell > Use Settings as Default. But you can also modify these themes or create your own.

To modify your window's appearance, follow these steps:

1. Choose Terminal > Preferences and click Settings on the toolbar.
2. Select a theme in the list to modify it. Or, to create your own new theme based on an existing one, select a theme and choose Duplicate Settings from the pop-up action menu (ⓘ) menu at the bottom of the list—or click the plus (+) button to add your own theme from scratch.

3. To modify the text that appears in the window of the currently selected theme, click Text. A few of the more useful options in this view are the following:
 - **Font:** To change the typeface or size, click the Change button, select a new font, size, and style from the Fonts palette, and close the palette. For best results, I strongly recommend choosing a fixed-width (monospaced) font, such as Courier, Monaco, or Lucida Console.
 - **Text color:** To change the color of the font, click the color button to the left of the word Text and chose a color using the Colors palette. You can pick a separate color for boldface text and for text you've selected with the mouse by clicking the color buttons next to Bold Text and Selection, respectively.
 - **Cursor attributes:** To change the shape of the cursor, select the Block, Underline, or Vertical Bar radio button. Check Blink Cursor if you want it to blink, and if you want to change the cursor's color, click the color button next to the word Cursor.
4. To modify the window itself, click Window. Some options you can change here include:
 - **Title bar elements:** To change the name of the window ("Terminal" by default), type new text into the Title field. You can also select any or all of the checkboxes beneath to display other information in the title bar, such as the name of the active process or the dimensions of the window. Terminal windows express their size in terms of rows and columns of text rather than in pixels. By default, Terminal windows are 24 rows by 80 columns, a size that harks back to old-style text-only terminals.
 - **Background color:** Click the color button under Background to display the Colors palette, in which you can choose a background color for the window. You can also adjust the opacity of the window's background color. Why would you want a window that's partially transparent? I like transparency because I can put a Terminal window directly above, say, a Web page and read instructions *through* the window as I type in Terminal! To adjust the opacity, move the Opacity slider at the bottom of the Colors palette.

- **Window size:** You can change the default window size for the current theme by typing numbers into the Columns and Rows fields, or you can simply resize the window to your liking later by dragging the resize control at the window's lower right corner.

Ocean blue: *My personal preference for window appearance is based on the Ocean theme (white text on a blue background) but with a larger window (160 columns by 50 rows) and background transparency set to 80%.*

5. To make a particular theme the default (which means it's used automatically when you launch Terminal, and when you press Command-N), select it and click the Default button beneath the list of themes. When you're finished adjusting window settings, close the Settings window.

All the settings you change here take effect immediately for existing windows using the selected theme, and for the next new window or tab opened using that theme.

SET A DEFAULT SHELL

As I explained in the introduction, this book covers only the `bash` shell, which has been the default since Mac OS X 10.3 Panther, though your account may have a different default if you upgraded your Mac from an earlier version of Mac OS X or migrated your account forward from an older system (even if you've gone through several upgrades since then). So you may want to confirm that you're running `bash`, or switch to `bash` if not.

Find Out Which Shell You're Using

To find out which shell is currently running, enter this:

```
echo $0
```

The shell replies with its own name, sometimes preceded by a hyphen:

```
-bash
```

Change Your Default Shell

If you want to change the default shell *only for yourself*, leaving other users' defaults intact, follow these steps (requires Mac OS X Leopard or newer):

1. Open the Accounts pane of System Preferences.
2. If the lock icon in the lower left corner of the window is closed, click it and enter your administrator's credentials to authenticate.
3. Control-click (right-click) on your name in the list on the left and choose Advanced Options from the contextual menu.
4. In the dialog that appears, choose a different shell from the Login Shell pop-up menu.
5. Click OK, and then close System Preferences.

Although the Advanced Options pane warns that you need to restart your computer to apply changes, changing the default shell takes effect with the next Terminal session you open.

Change the Default Terminal Shell

To change the default shell Terminal opens *regardless* of which user is logged in or what that user's individual preference is, do the following:

1. Choose Terminal > Preferences and click Startup on the toolbar.
2. Next to Shells Open With, select Command (Complete Path) and make sure the path to `bash (/bin/bash)` is filled in. (To use a different shell, such as `zsh`, substitute that shell's name for `bash`.)

The setting applies starting with the next session you open.

The \$, #, and Other Strange Things on My Command Line

By default, when you open a Terminal window, you see a prompt that ends in a `$` (followed by the cursor), like this:

```
Joes-MacBook-Pro:~ jk$ █
```

If you log in as the root user (see [Perform Actions as the Root User](#)), the prompt ends instead in a `#` character:

```
bash-3.2# █
```

Other shells have different default characters. For example, in the `zsh` shell, the prompt normally ends with a `%`. As a result, when you're reading articles and Web sites listing commands you might enter in Terminal, you might run across examples like these:

```
$ open -e file1
# chown www file1
% top
```

The `$`, `#`, or `%` at the beginning merely signifies that what follows is a command to be typed and, in the case of `#`, that it's supposed to be typed by the root user. You wouldn't actually type `$`, `#`, or `%`.

I don't use that convention in this book; whatever you need to type on the command line simply appears in a special font, usually on a line by itself. I find those extra characters distracting.

In any case, you can easily change the prompt so that it shows something else entirely. If you want your prompt to look like this...

```
Joe rocks +> █
```

...you can make that happen. See [Change Your Prompt](#) for details.

Look Around

In this section, I help you find your way around your Mac’s disk from the command line and, at the same time, teach you some of the most common navigational commands and conventions. For right now, you’re going to look, but not touch—that is, nothing you do here can change any files or cause any damage, as long as you follow my instructions.

DISCOVER WHERE YOU ARE

Ready to start learning some commands? Here we go. Open a Terminal window and enter this:

```
pwd
```

Reminder: To enter something on the command line, type it and press Return or Enter afterwards.

The `pwd` command stands for “print working directory,” and it gives you the complete path to the directory you’re currently using. If you haven’t done anything else since opening a Terminal window, that’s your home directory, so you’ll see something like this:

```
/Users/jk
```

That’s not exciting, but it’s extremely important. As you navigate through the file system, it’s easy to get lost, and ordinarily your prompt only tells you the name of your current directory, not where it’s located on your disk. When you’re deep in the file system, being able to tell exactly where you are can be a huge help.

SEE WHAT'S HERE

If you were in the Finder, you’d know exactly what’s in the current folder just by looking. Not so on the command line; you must ask explicitly. To get a list, you use the “list” command:

```
ls
```

What you get by default is a list along the lines of the following:

```
Applications      Library           Pictures
Desktop           Movies            Public
Documents         Music             Sites
Downloads
```

Items are listed alphabetically, top to bottom and then left to right. But as you can see, this doesn't tell you whether these items are files or directories, how large they are, or anything else about them. So most people prefer to use the more-helpful long format by adding the `-l` flag:

```
ls -l
```

This produces a result something like:

```
drwxr-xr-x    2 jk    admin    68 Aug 27 13:11 Applications
drwxr-xr-x   18 jk    admin   612 Feb 12 09:42 Desktop
drwxr--r--@ 108 jk    admin  3672 Feb  9 14:35 Documents
drwx-----   15 jk    admin   510 Feb 12 11:17 Downloads
drwx-----   94 jk    admin  3196 Feb 11 22:40 Library
drwx-----   13 jk    admin   442 Dec 30 15:34 Movies
drwxr--r--   15 jk    admin   510 Aug 27 15:02 Music
drwxr--r--   14 jk    admin   476 Jan 26 19:40 Pictures
drwxr-xr-x    7 jk    admin   238 Jan 22 23:13 Public
drwxr-xr-x    7 jk    admin   238 Jun  6  2007 Sites
```

Reading from right to left, notice that each line ends with the item's name. To the left of the name is a date and time showing when that item was most recently modified. To the left of the date is another number showing the item's size in bytes. (In the case of directories, that number doesn't tell you the total size of the directory's *contents*, only the size of the information stored *about* the directory.) See the sidebar on the next page, [Making Output \(More\) Human-Readable](#), to find out how to turn that number into a nicer format.

Later in this book, in [Understand Permission Basics](#), I go into more detail about all those characters that occupy the first half of each line, such as `drwxr-xr-x 7 jk admin`; those characters describe the item's permissions, owner, and group. For the moment, just notice the very first letter—it's `d` in every item of this list. The `d` stands for "directory," meaning these are all directories. If the item were a file, the `d` would be replaced with a hyphen (`-`), for example: `-rwxr-xr-x`.

Finally, look at one other number, between the permissions and owner (in `drwxr--r-- 14 jk` the number is 14). That's the number of *links* to the item, and though links are too advanced to explain in detail here, the number serves one practical purpose: it gives you an approximation of the number of items in a directory. In fact, it will always be at least two higher than the number of visible files or directories in the directory (for complicated reasons). For now, just know that the number can tell you, at a glance, if a directory has only a few items or many.

Making Output (More) Human-Readable

I've shown the `-l` (long format) flag, which provides much more detail than the `ls` command alone. But it shows the file size in *bytes*, which isn't a convenient way to tell the size of large files. For example, an `ls -l` listing might include the following:

```
-rw-r--r--@ 1 jk admin 169552353 Jan 13 17:07 image.dmg
```

Really—169552353 bytes? Wait a minute, let me do some math...how large is that exactly?

Luckily, you can improve on this by adding the `-h` flag, which stands for "human-readable." (In fact, `-h` works with many commands, not just `ls`.) You can enter either `ls -lh` or `ls -l -h`. Either way, you get something like this:

```
-rw-r--r--@ 1 jk admin 162M Jan 13 17:07 image.dmg
```

Aha! The file is 162 megabytes (M) in size. That I understand!

I don't want to belabor the `ls` command, but it will without question be one of the top two or three things you type on the command line—you'll use it constantly. So it pays to start getting `ls` (along with a flag or two) into your muscle memory. For a way to display even more information with `ls`, see the recipe [List More Directory Information](#).

Note: You can also list the contents of a directory *other than* your current one like this: `ls /some/other/path`.

REPEAT A COMMAND

If you've just entered a two-character command, it's no big deal to enter it again. But sometimes commands are quite complex, wrapping over several lines, and retyping all that is a pain. So I want to tell you about two ways of repeating commands you've previously entered.

Arrow Keys

First, you can use the Up and Down arrow keys to move backward and forward through the list of commands you've recently typed. For example, if the last command you typed was `ls -lh`, simply pressing the Up arrow once puts that on the command line. (Then, to execute it, you would press Return or Enter.) Keep pressing the Up arrow, and you'll step backward through even more commands. You can even scroll through commands you entered *in previous sessions*. The Down arrow works the same way—it progresses forward in time from your current location in the list of previous commands.

The !! Command

Another handy way of repeating a command is to enter `!!` (that's right—just two exclamation points). This repeats your previous command. Try it now. Enter, say, `pwd`, and get the path of your current directory. Then enter `!!` and you'll get the same output.

Again, this isn't terribly interesting when you're talking about short commands, but it can save time and effort with long commands.

!! Plus

The `!!` need not stand alone on the command line—you can add stuff before or after in order to expand the previous command. For example, if you previously entered `ls -l` and you now want to enter `ls -l -h`, you could repeat the previous command and add an extra flag like so:

```
!! -h
```

Or, if you enter a command like `rm file1` (remove the file `file1`) and get an error message telling you that you don't have permission, you can repeat it preceded by the `sudo` command (described in [Perform Actions as the Root User](#)):

```
sudo !!
```

In this example, the result would be exactly the same as entering:

```
sudo rm file1
```

CANCEL A COMMAND

What if you type some stuff on the command line and realize you don't want to enter the command after all? Well, you *could* backspace over it, but that could take a while if there's a lot of text on the line. An easier way to back out of a command without executing it is to press either Control-C or Command-. (period). The shell creates a new, blank command line, leaving your partially typed line visible but unused.

MOVE INTO ANOTHER DIRECTORY

This has been a lovely visit in your home directory, but now it's time to explore. To change directories, you use the `cd` command. As you saw a moment ago, one of the directories inside your home directory is called Library. Let's move there now, like so:

```
cd Library
```

Just in case: Notice in this example that `Library` is capitalized. Sometimes case isn't important on the command line (as I explain ahead in [Case Sensitivity](#)), but you can't go wrong if you always use the correct case.

When you put a directory name after the `cd` command, it assumes you want to move into that directory *in your current location*. If there doesn't happen to be a directory called Library in your current directory, you see an error message like this:

```
-bash: cd: Library: No such file or directory
```

As a reminder, the command line environment doesn't list the contents of a directory unless you ask it to (using `ls`), so using `cd` doesn't automatically show what's in your new location. You know the command succeeded if you don't see an error message, and by default your prompt will include the name of your current directory.

Move Up or Down

Now that you're in the `Library` directory inside your home directory (`~/Library`), you can use `ls` to look around; you'll see that one of the directories inside the current one is `Preferences`. To move down a level into preferences, you'd enter `cd Preferences`. And so on.

To go up a level, you use the `..` convention, which means “the directory that encloses this one.” For example, if you're currently in `/Users/jk/Library/Preferences` then the directory that encloses `Preferences` is `/Users/jk/Library`, so in this particular location two periods (`..`) means `/Users/jk/Library`.

To get there, you enter:

```
cd ..
```

That translates as “change directories to the one that encloses this one.” You can keep going up and down with `cd ..` and `cd directory` (fill in the name of any directory) as much as you like.

Spaced out: *Moving into directories with spaces in their names requires extra effort; read [Understand How Paths Work](#), ahead.*

Move More Than One Level

Nothing says you have to move up or down just one level at a time. If you're currently in `/Users/jk` and you know that there's a `Library` directory inside it, and inside that there's a `Preferences` directory, you can jump directly to `Preferences` like so:

```
cd Library/Preferences
```

The slash (`/`) simply denotes that the term to its right is a directory inside the term on its left: `Preferences` is a directory inside `Library`. You can add on as many of these as you need:

```
cd Library/Logs/Adobe/Installers
```

The same thing works in the other direction. If you're currently in `/Users/jk/Library/Preferences`, you could enter `cd ..` to move into `Library`. Or, you could enter `cd ../..` to move directly into `jk`, or `cd ../../..` to move into `Users`.

Move to an Exact Location

So far, we've been moving using relative locations—a directory inside the current one, or a directory that encloses the current one. But if you know exactly where you're going, you can jump directly to any location on your disk. Just specify the full path, beginning with a slash (/), which represents the root level of your disk. For example, enter this:

```
cd /private/var/tmp
```

That takes you directly to `/private/var/tmp` (a rather boring directory full of caches and temporary files, and one that's normally invisible in the Finder) without having to navigate all the way up to the root level of your drive and then back down.

Speaking of the root level: If you want to go to the very top of your hard disk hierarchy, just enter this:

```
cd /
```

Move Between Two Directories

Another handy shortcut, which lets you go back to the last directory you were in, is this:

```
cd -
```

For example, suppose I start in my home directory and then enter `cd /Users/Shared`. I do some things in that directory, and then I enter `cd ~/Library/Preferences` to look at some files there. If I then enter `cd -` I jump back to `/Users/Shared` (the last directory I was in), without having to type or even remember its path.

JUMP HOME

Once you've changed directories a few times, you may want to get back to your home directory. Of course, you could keep navigating up or down, one directory at a time, until you got there, or you could enter the complete path to your home directory (`cd /Users/jk`, for example). But as you may have read (in [Basics](#) at the beginning of the book), Mac OS X has another shortcut (along the lines of `..`) that means “the current user's home directory”: the tilde (`~`). So one way to jump home, from any location on your disk, is to enter:

```
cd ~
```

But in fact, it can be even easier. If you enter `cd` alone, with nothing after it, the command assumes you want to go home, so `cd` by itself does the same thing as `cd ~`.

Just as you can enter the full path after `cd` to jump to any spot on your disk, you can substitute `~` whenever you'd otherwise use the full path to your home directory. So, even if you're in `/private/var/tmp`, you can go directly to the Library directory inside your home directory with this command:

```
cd ~/Library
```

Not my type: This might be a good time to remind you that the command line can be unforgiving. If you type an extra period, leave out a space, or make some other similarly tiny error, your command might not work at all—or it might do something entirely unexpected. That need not frighten you, but just be aware that you should be deliberate and careful when typing on the command line.

UNDERSTAND HOW PATHS WORK

You've already seen both relative paths (such as `Library/Preferences`, which means the `Preferences` directory inside the `Library` directory inside my *current* directory) and absolute paths, which begin with a slash (such as `/Library/Preferences`, which means the `Preferences` directory inside the `Library` directory at the *top* level of your disk). But there are a few other things you should understand about paths.

Spaces in Paths

Mac OS X lets you put almost any character in a file or folder name, including spaces. But space characters can get you in trouble in the command-line environment, because normally a space separates commands, flags, and arguments. Suppose you were to enter this:

```
cd My Folder
```

Even if there were a folder named `My Folder` in the current directory, the command would produce an error message, because the `cd` command would assume that both `My` and `Folder` were intended to be separate arguments.

You can deal with spaces in either of two ways:

- **Quotation marks:** One way is to put the entire path in quotation marks. For example, entering `cd "My Folder"` would work fine.
- **Escape the space:** The other way is to put a backslash (\) before the space—this *escapes* the space character, making the shell treat it literally rather than as a separator between arguments. So this would also work: `cd My\ Folder`.

Back to basics: *To be crystal clear, the backslash (\) is normally located on a key just to the right of the] key. It has a completely different meaning from the ordinary (forward) slash (/), located on the same key as the question mark. Don't mix them up!*

Wildcards

You can use wildcards when working on the command line; these can save you a lot of typing and make certain operations considerably easier. The two wildcards you're most likely to use are these:

- *** (asterisk):** This means “zero or more characters.” For example, if you want to switch to a directory called Applications, you could enter `cd App*` and, as long as there was no other directory there that started with those three letters, you'd go directly to the Applications folder. (I talk about another way of doing something similar ahead a few pages in [Use Tab Completion](#).)

You can use this wildcard with almost any command. For instance, if you're in your home directory, you could type `ls D*` to list all and only the items that begin with “D” ([Desktop](#), [Documents](#), [Downloads](#)).

- **? (question mark):** This means “any single character.” That means `?at` could match `bat`, `cat`, `fat`, `rat`, `sat`, and so on. If you have many files with similar names—say, sequentially numbered photos—you could limit the ones listed with something like `ls 01?? .jpeg`.

Case Sensitivity

Here's a trick question: is the Mac OS X command line case-sensitive? The answer is yes—and no! Suppose you're in `~`. There's a directory in there called `Pictures`, and you could move into it in any of these ways (among others):

```
cd Pictures
cd pictures
cd Pic*
cd pic*
```

That certainly seems to suggest that the command line is *not* case-sensitive, because using either `p` or `P` has the same effect. But if you're a MobileMe member and your iDisk is mounted in the Finder, you may notice that it also contains a folder called `Pictures`. So navigate to your iDisk on the command line by entering `cd /Volumes/iDisk`, and now try all those commands again. You'll discover that *only* the ones beginning with a capital `P` work! What's going on here?

The difference between the two locations is that your iDisk is formatted using a case-sensitive version of the Mac OS Extended (HFS+) file system, whereas your main disk (assuming you haven't manually changed it for some reason) uses a case-insensitive version of the file system.

You won't see any visual cue to let you know whether a particular volume uses a case-sensitive format. So the safest assumption is to always use the correct case: that always works.

UNDERSTAND MAC OS X'S FILE SYSTEM

You surely know from day-to-day use that your Mac has a bunch of standard folders at the top level of your hard disk—`Applications`, `Library`, `System`, and `Users`, at minimum. You may have also noticed that each user's home folder has its own `Library` folder (not to mention a `Desktop` folder, a `Documents` folder, and several others). In addition to these and the numerous other folders you can see in the Finder, Mac OS X has a long list of directories that are normally invisible (because most users never need to interact with them directly), but you can see them from the command line.

I could explain what every single (visible) folder and (hidden) directory is for, and how to make sense of the elaborate hierarchy in which Mac OS X stores all its files. But that would take many pages and, honestly, it would be mighty boring. So I'm going to let you in on a little secret: *you don't need to know*.

I mean it: you don't need to know why one program is stored in `/bin` while others are in `/usr/bin`, `/usr/local/bin`, or any of numerous other places. You don't need to know why you have a `/dev` directory or what goes in `/private/var`. Seriously. Knowing all those things might be useful if you're a programmer or a system administrator, but it's absolutely irrelevant for ordinary folks who want to do the kinds of things discussed in this book. True, I may direct you to use a program in `/usr/sbin` or modify a file in `/private/etc` (or whatever), but as long as you can follow the instructions to do these things, you truly don't need to know all the details about these directories.

So, instead, I want to provide a very short list of the key things you should understand about Mac OS X's file system:

- **The invisible world of Unix:** If you enter `ls -l /` (go ahead and do that), you get a list of all the files and directories at the root level of your disk. You'll see familiar names such as `Applications` and `Users`, and some less-familiar ones, such as `bin` and `usr`. Here at the root level of your disk, directories that begin with a lowercase letter and aren't shown in the Finder (such as `bin`, `private`, `usr`, and `var`), plus a few files that are also normally invisible (such as `mach_kernel` and `mach_kernel.ctfsys`), make up Darwin, the Unix core of Mac OS X. Similar directories appear in other Unix and Unix-like operating systems.
- **Recursion, repetition, and recursion:** If you were to work your way from the root of your disk down through all its directories and subdirectories, you'd notice a lot of names that appear over and over again. For example, there's a top-level `/Library` directory, another inside `/System`, and yet another inside each user's home directory (`~/Library`). Similarly, there are top-level `/bin` and `/sbin` directories, but also `/usr/bin` and `/usr/sbin`. The reasons for all these copies of similar-looking directories are sometimes practical, sometimes purely historical. But everything has its place.

You don't need to grasp all the logic behind what goes where, but you do need to be sure you're in the right place when you work on the command line. For instance, if an example in this book tells you to do something in `~/Library`, be absolutely sure that's where you are, as opposed to, say, `/Library`. The smallest characters—in particular, the period (`.`), tilde (`~`), slash (`/`), backslash (`\`), and space (), have the utmost significance on the command line, so always pay strict attention to them!

- **The bandbox rule:** My grandfather had a curious and oft-repeated expression: “Don't monkey with the bandbox.” He (and, subsequently, my mother) used this to mean, approximately, “Don't mess with something if you could break it and not be able to put it back together.” (As a child, I had quite a propensity for disassembling things and then getting stuck!) On the command line, this means don't go deleting, moving, or changing files if you don't know what they are or what the consequences could be. Something that seems insignificant or useless to you could be crucial to the functioning of your Macintosh. (As a corollary, it should go without saying that you back up your Mac thoroughly and regularly; read my book *Take Control of Mac OS X Backups* if you need help or advice with backups).

USE TAB COMPLETION

Because everything you do on the command line involves typing, it can get kind of tedious spelling out file and directory names over and over again—especially since even the slightest typo will make a command fail! So the bash shell includes a number of handy features to reduce the amount of typing you have to do. Earlier I explained how to use the arrow keys and the `!!` command to repeat previous commands ([Repeat a Command](#)). Now I want to tell you about a different keystroke-saving technique: tab completion.

Here's the basic idea. You start typing a file or directory name, and then you press the Tab key. If only one item in the current directory starts with the letter(s) you typed, the `bash` shell fills in the rest of that item's name. If there's more than one match, you'll hear a beep; press Tab again to see a list of all the matches.

For example, try this:

```
cd
```

Now that you're in your home directory, type `cd De` (*without* pressing Return) and press Tab. Your command line should look like this:

```
cd Desktop/
```

If you do in fact want to change to your Desktop directory, you can simply press Return. Or, you can type something more on the line if need be. For now, let's stay where we are—press Control-C to cancel the command.

Next, try typing `cd D` (again, without pressing Return) and press Tab. You should hear a beep—signifying that there was more than one match—but nothing else should happen. Press Tab again. Now you'll see something like this:

```
Desktop/  Documents/ Downloads/
```

And, on the next line, your command-in-progress appears again exactly as you left it off:

```
cd D
```

In this way, tab completion lets you know what your options are; you can type more letters (say, `oc`) and press Tab again to have it fill in Documents for you.

Tab completion isn't limited to just the current directory. For example, enter `cd ~/Lib` and press Tab. The bash shell fills in the following:

```
cd ~/Library/
```

Now type `Favorites` and press Tab. You should see `Favorites` filled in, like this:

```
cd ~/Library/Favorites/
```

In this way, you can keep going as many levels deep as you need to.

Case by case: *Tab completion in `bash` is always case-sensitive, even on a volume that doesn't use case-sensitive formatting. If a directory is named `Widgets`, typing `wi` and pressing Tab produces no matches.*

FIND A FILE

In the command-line environment, as in the Finder, you may not know where to find a particular file or directory. Two commands can supply that information readily: `find` and `locate`. Each has its pros and cons.

Find

To use the `find` command, you give it a name (or partial name) to look for and tell it where to start looking; the command then traverses every directory in the area you specify, looking at every single file until it finds a match. That makes it slow but thorough.

For example, suppose I want to find all the files anywhere in my home directory with names that contain the string “keychain.” I can do this:

```
find ~ -name "*keychain*"
```

After the command `find`, the `~` tells the command to begin looking in my home directory (and work its way through all its subdirectories). The `-name` flag says to look for patterns in the pathname (which may include the names of directories, not necessarily file names). I put the search string inside quotation marks, with an asterisk (*) wildcard at the beginning and end to signify that there may be other letters before or after “keychain.”

Even a simple search such as this one can take several minutes, because it must look at every single file, starting at the path I specified. To make it go quicker, I could specify a narrower search range. For example, to have it look only in my `~/Library` directory, I’d enter:

```
find ~/Library -name "*keychain*"
```

Let me offer a few other tips for using `find`:

- To search in the current directory (and all subdirectories), use a period (.) as the location: `find . -name "*keychain*"`.

- To search your entire disk, use a slash (/) as the location:

```
find / -name "*keychain*"
```

- Normally, `find` is case-sensitive, so a search for `"*keychain*"` would *not* match a file named `Keychain`. To make a search case-insensitive, replace `-name` with `-iname`, as in `find ~ -iname "*user data*"`.

- During a search, if `find` encounters any directories you don't have permission to search, it displays the path of the directory with the message "Permission denied." To search these paths, use `sudo` before `find`, as described in [Perform Actions as the Root User](#).

Tip: You can also use the `find` command to search the contents of files, though that process takes even longer. See the recipe in [Find a File by Content](#).

Locate

The other way to find files is to use the `locate` command. Unlike `find`, `locate` doesn't traverse every file to find what you're looking for. Instead, it relies on a database (index) of file and path names. The benefit of using the index is that `locate` is lightning fast. The downside is, the database is normally updated only once a week, so `locate` usually can't find files you've added or renamed recently.

To use `locate`, just type that command followed by any portion of the filename you want to look for (no wildcards required). For example:

```
locate keychain
```

Like `find`, `locate` performs case-sensitive searches by default. To make a search case-insensitive, add the `-i` flag:

```
locate -i keychain
```

If you enter `locate` and get an error stating that no database exists—or if it exists but is outdated—you can create or update it by entering this:

```
/usr/libexec/locate.updatedb
```

The command may take some time to complete, because it does have to look at every file on your disk—or nearly so.

I've skipped over one important detail: by default, `locate` only indexes (and finds) files you own (mostly the contents of your home directory). However, if you run the database updating script using `sudo` (see [Perform Actions as the Root User](#)), it indexes every file on your disk, and `locate` can therefore find every file. The benefit of this is being able to find more files with `locate`, but if you attempt to do this, a security warning appears informing you that once you've indexed all your files, any user of your Mac can discover the name and location

(though not the contents) of any file on your disk. Moreover, the next time the `locate` database updates on its weekly schedule, your system-wide index of files will be replaced with a version that contains only those you have permission to read.

VIEW A TEXT FILE

You may not read a lot of plain text files in the Finder, but the need to do so comes up more frequently in the command-line environment—reading documentation, examining programs’ configurations, viewing shell scripts, inspecting logs, and numerous other situations. You can use many tools to read a file, of which I cover just a few here. (If you want to modify a text file, see [Edit a Text File](#), later.)

You can use these commands with any text file on your Mac, but in these examples I use a file every Mac user should have: the license for the postfix email server, located at `/private/etc/postfix/LICENSE`.

More or Less

An early Unix program for reading text files was called `more`. It was pretty primitive and wouldn’t let you move backward to see earlier text. So a new program came along that was supposed to be the opposite of `more`: `less`. In Mac OS X, both names still exist, but they point to the same program; whether you enter `more` or `less`, you’re actually running `less`. (There are some subtle differences depending on which command you use, but they’re not worth mentioning.)

You can use `less` to read a text file like this:

```
less /private/etc/postfix/LICENSE
```

You see the top portion of the file initially. You can scroll down a line at a time using the Down arrow key (and back up using the Up arrow key), scroll ahead a screen at a time by pressing the Space bar, or backward a screen at a time by pressing the `B` key (all by itself). To quit `less`, simply press the `Q` key (all by itself).

Cat

The Unix `cat` command (short for “concatenate”) combines files, but you can also use it to display a text file on your screen. Unlike `less`, it doesn’t give you a paged view, it simply pours the entire contents of the file, regardless of length, onto your screen. You can then

scroll the Terminal window up and down, as necessary, to view the contents. To use `cat`, follow this pattern:

```
cat /private/etc/postfix/LICENSE
```

Tail

If you open a long text file with `less`, it can take quite a bit of tapping on the Space bar to reach the end, which is awkward if the information you want happens to be at the end—as is the case with most logs. And if you use `cat`, it can clutter your Terminal window with lots of information you don't need. To jump to the end of a text file, use a different program: `tail`, which displays the *tail end* of a file.

If you enter `tail` followed by the filename, it displays the last ten lines of the file:

```
tail /private/etc/postfix/LICENSE
```

The `tail` command has flags that enable you to control how much of the file is shown and in what way, but for the sake of brevity I want to mention just one: `-n` (number of lines). Type `tail` followed by the `-n` flag, a space, and a number to set the output to that number of lines from the end of the file:

```
tail -n 50 /private/etc/postfix/LICENSE
```

GET HELP

Almost every program and command you use on the command line has documentation that explains its syntax and options, and in many cases includes examples of how to use the command. This documentation isn't always clear or helpful, but it's worth consulting when you have a question. You can get at these manual pages in several ways.

In a Terminal Window

When you're on the command line, the quickest way to get information about a command is to use the `man` (manual) command. Simply enter `man` followed by the command you want to learn about. For example:

```
man ls  
man cp  
man locate
```

The results appear in a viewer that works like `less`. For best results, I suggest opening a new Terminal window to display `man` pages, so you can see the documentation and your active window side-by-side.

Note: You can, of course, get instructions for using the `man` application itself by entering—you guessed it—`man man`.

In a Mac OS X Application

If you want to learn about a command-line program when Terminal isn't running—or you prefer to read about it in a more user-friendly environment—you can download any of numerous free (donations accepted) applications that give you access to the same information. Some examples include:

- Man Handler: http://geeksuit.com/software/77_0_1_0_M/
- ManOpen: <http://www.clindberg.org/projects/ManOpen.html>
- Man Viewer: http://www.kendallp.net/at_PAK/ManViewer/

Tip: If you want to read `man` pages as nicely formatted PDF files, try the recipe [Read man Pages in Preview](#). Or, if you prefer to view them in BBEdit, try [Read man Pages in BBEdit](#).

On the Web

Another way to view `man` pages for command-line programs is to consult a Web site where they're available in convenient HTML form:

- Apple's official repository of manual pages for Darwin is located at <http://developer.apple.com/documentation/Darwin/Reference/ManPages/>.
- OSXFAQ.com also has a complete set of manual pages online at <http://www.osxfaq.com/MAN/index/A.ws>.

Tip: You can also read your Mac's `man` pages in your Web browser using the free Bwana application (<http://www.bruji.com/bwana/>).

CLEAR THE SCREEN

As you work in Terminal, your window may fill up with commands and their output. The command line itself will, as usual, always be the last line, but the rest of the window can become cluttered with the residue of earlier commands. If you find all that text distracting and want to clear the window (so that it looks much like it did when you started the session), enter `clear` or press Control-L. Note that this actually just moves your command line up to the top of the window with empty space below it; you can still scroll up to see what was on the screen earlier.

END A SHELL SESSION

When you're finished working on the command line for a while, you *could* simply close the Terminal window, or even quit Terminal, but you *shouldn't*. That would be a bit like turning off your Mac by flipping the switch on the power strip instead of choosing  > Shut Down. The proper way to end a shell session in Terminal is to enter `exit`, which gracefully stops any programs you are running in the shell, and then quits the shell program itself.

By default, your Terminal window remains open after you've done this. If you want it to close when you exit, choose Terminal > Preferences, click the Settings button on the toolbar, and then click Shell. From the When the Shell Exits pop-up menu, choose Close the Window.

Work with Files and Directories

Much of what you'll need to do on the command line involves working with files in some way—creating, deleting, copying, renaming, and moving them. This section covers the essentials of interacting with files and directories.

CREATE A FILE

I want to mention a curious command called `touch` that serves two interesting functions:

- When applied to a nonexistent file, `touch` creates an empty file.
- When applied to an existing file or folder, `touch` updates its modification date to the current date and time, marking it as modified.

So, try entering the following command:

```
touch file1
```

Now use `ls -l` to list the contents of your current directory. You'll see `file1` in the list. This file that you've just created is completely empty. It doesn't have an extension, or a type, or any contents. It's just a marker, though you could use a text editor, for example, to add to it. Why would you do this? There are occasionally situations in which a program behaves differently based solely on the existence of a file with a certain name in a certain place. What's in the file doesn't matter—just that it's there. Using `touch` is the quickest way to create such a file. But for the purposes of this book, the reason to know about `touch` is so you can create files for your own experiments. Since you're creating the files, you can rename, move, copy, and delete them without worrying about causing damage. So try creating a few files right now with `touch`.

Don't space out: Remember, if you want to create a file with a space in the name, put it in quotation marks (`touch "my file"`) or escape the space character (`touch my\ file`).

As for the other use of `touch`—marking a file as modified—you might do this if, for example, the program that saved it failed to update its modification date for some reason and you want to make sure your backup software notices the new version. You use exactly the same syntax, supplying the name of the existing file:

```
touch file1
```

When applied to an existing file, `touch` doesn't affect its contents at all, only its modification date.

CREATE A DIRECTORY

To create a directory (which, of course, appears in the Finder as a folder), use the `mkdir` (make directory) command. To make a directory called `apples`, you'd enter the following:

```
mkdir apples
```

That's it! Other than the fact that you can create a new directory in some other location than your current one (for example, `mkdir ~/Documents/apples`), and the fact that spaces, apostrophes, or quotation marks in directory names must be escaped (see [Spaces in Paths](#)), there's nothing else you need to know about `mkdir` at this point.

COPY A FILE OR DIRECTORY

To duplicate a file (in the same location or another location), use the `cp` (copy) command. It takes two arguments: the first is the file you want to copy, and the second is the destination for the copy. For example, if you're in your home directory (`~`) and you want to make a copy of the file `file1` and put it in the `Documents` directory, you can do it like this:

```
cp file1 Documents
```

The location of the file you're copying, and the location you're copying it to, can be expressed as relative or absolute paths. For instance:

```
cp file1 /Users/Shared
cp /Users/jk/Documents/file1 /Users/Shared
cp file1 ..
cp ../../file1 /Users/Shared
```

If you want to duplicate a file and keep the duplicate in the same directory, enter the name you want the duplicate to have:

```
cp file1 file2
```

Likewise, if you want to copy the file to another location and give the copy a new name, specify the new name in addition to the destination:

```
cp file1 Documents/file2
```

Avoid Overwriting Files

Look back at the first example:

```
cp file1 Documents
```

Anything strike you as suspicious about that? We know that there's a file called `file1` and a directory called `Documents` in the current directory, so will this command copy `file1` into `Documents` or make a copy in the *current* directory and name the copy `Documents` (potentially overwriting the existing directory)? The answer is: `cp` is smart. The command assumes that if the second argument is the name of an existing directory, you want to copy the file to that directory; otherwise, it copies the file in the current directory, giving it the name of the second argument. It won't overwrite a directory with a file.

But, in fact, `cp` is not *quite* as smart as you might like. Say there's *already* a file in `Documents` called `file1`. When you enter `cp file1 Documents`, the command happily overwrites the file already in `Documents` without any warning! The same goes for duplicating files in the same directory. If the current directory contains files `file1` and `file2`, entering `cp file1 file2` *overwrites* the old `file2` file with a copy of `file1`!

Fortunately, you can turn on an optional warning that appears whenever you're about to overwrite an existing file, using the `-i` flag. So if you enter `cp -i file1 Documents` and there's already a `file1` in `Documents`, you'll see this:

```
overwrite Documents/file1? (y/n [n])
```

Then enter `y` or `n` to allow or disallow the move. "No" is the default.

Because the `-i` flag can keep you out of trouble, I suggest you always use it with the `cp` command. Or, for an easier approach, set up an alias that does this for you automatically; see [Create Aliases](#).

Copy Multiple Files

You can copy more than one file at a time, simply by listing all the files you want to copy, followed by the (single) destination where all the copies will go. For example, to copy files named `file1`, `file2`, and `file3` into `/Users/Shared`, enter this:

```
cp file1 file2 file3 /Users/Shared
```

Copy a Directory

You can use the `cp` command to copy a directory, but you must add the `-r` (recursive) flag. For instance, given a directory named `apples`, this command would produce an error message:

```
cp apples ~/Documents
```

The correct way to enter the command is as follows:

```
cp -r apples ~/Documents
```

Slashes away: *Avoid putting a slash at the end of the source directory when using `cp -r`. That slash causes the command to copy every item within the directory to the destination. For example, `cp -r apples/ ~/Documents` wouldn't copy the `apples` directory itself to your `~/Documents` directory, but rather all the contents of the `apples` directory to your `~/Documents` directory—probably not what you want.*

If you use tab completion with the `cp` command, be extra careful, because tab completion adds trailing slashes automatically.

MOVE OR RENAME A FILE OR DIRECTORY

If you want to move a file from one location to another, you use the `mv` (move) command. This command takes two arguments: the first is what you want to move, and the second is where you want to move it. For example, if you're in `~` and you want to move `file1` from the current directory to the `Documents` directory, you can do it like this:

```
mv file1 Documents
```

As with `cp`, the location of the file you're moving, and the location you're moving it to, can be relative or absolute paths. Some examples:

```
mv file1 /Users/Shared
mv /Users/jk/Documents/file1 /Users/Shared
mv file1 ..
mv ../../file1 /Users/Shared
```

If you want to rename a file, you *also* use the `mv` (move) command. Weird as it may sound, `mv` does double duty. When you're renaming a file, the second argument is the new name. For example, to rename the file `file1` to `file2`, leaving it in the same location, enter this:

```
mv file1 file2
```

Tip: Want to move a file from somewhere else to your current directory, without having to figure out and type a long path? You can represent your current location with a period (`.`), preceded by a space. So, to move `file1` from `~/Documents` to your current directory, enter `mv ~/Documents/file1 .` on the command line.

Avoid Overwriting Files

The `mv` command works the same way as `cp` when it comes to overwriting files: it won't overwrite a directory with a file of the same name, but it will happily overwrite files unless you tell it not to do so.

Fortunately, `mv` supports the same optional `-i` flag as `cp` to warn you when you're about to overwrite a file. So if you enter `mv -i file1 Documents` and there's already a `file1` in `Documents`, you'll see this:

```
overwrite Documents/file1? (y/n [n])
```

You can then enter `y` or `n` to allow or disallow the move. Again, "no" is the default.

As with `cp`, the `-i` flag is such a good idea that I suggest you get in the habit of using it every single time you enter `mv`. Alternatively, you can set up an alias that does this for you automatically; see [Create Aliases](#).

Move and Rename in One Go

Since `mv` can move and rename files, you may be wondering if you can do both operations at the same time. Indeed you can. All it takes is entering the new name after the new location. For instance, if you

have a file named `file1` and you want to move it into the `Documents` directory where it will then be called `file2`, you can do it like this:

```
mv file1 Documents/file2
```

Move Multiple Files

You can move several files at once, simply by listing all the files you want to move, followed by the (single) destination to which they'll all go. For example, to move files named `file1`, `file2`, and `file3` into `/Users/Shared`, enter this:

```
mv file1 file2 file3 /Users/Shared
```

Too wild to handle: You can use wildcards like `*` with `mv`—for example, entering `mv *.jpg Pictures` moves all the files from the current directory ending in `.jpg` into the `Pictures` directory. But when using `mv` to rename files, wildcards may not work the way you expect. For example, you cannot enter `mv *.JPG *.jpeg` to rename all files with a `.JPG` extension to instead end in `.jpeg`; for that, you must use a shell script (see [Command-Line Recipes](#) for an example).

DELETE A FILE

To delete a file, use `rm` (remove), followed by the filename:

```
rm file1
```

Tip: To try this out safely, use `touch` to create a few files, enter `ls` to confirm that they're there, then use `rm` to remove them. Then enter `ls` again to see that they've disappeared.

You can delete multiple files at once by listing them each separately:

```
rm file1 file2 file3 file4
```

And, of course, you can use wildcards:

```
rm file*
```

Needless to say, you should be extra careful when using the `*` wildcard with the `rm` command!

Warning! *If you put something in the Mac OS X Trash, you can later drag it back out, up until the moment you choose Finder > Empty Trash. But the `rm` command (and the `rmdir` command, described next) has no such safety net. When you delete files with these commands, they're gone—instantly and completely! If you want to be especially cautious, you can follow `rm` with the `-i` flag, which requires you to confirm (or disallow) each item you're deleting before it disappears forever—for example, `rm -i cup*` prompts you to confirm the deletion of each file that has a name beginning with `cup`.*

DELETE A DIRECTORY

Just as you can delete a folder in the Finder by dragging it to the Trash, you can delete a directory on the command line with the `rmdir` (remove directory) command. To delete a directory named `apples`, you can enter this:

```
rmdir apples
```

As with `rm`, you can delete multiple directories at the same time:

```
rmdir pomegranates pomelos  
rmdir pome*
```

This command only works on empty directories. (A directory can have invisible files created by Mac OS X; don't assume it's empty just because you didn't put anything there.) If you run `rmdir` on a non-empty directory, you get this error message:

```
rmdir: apples: Directory not empty
```

This is a safety feature designed to prevent accidental deletions. If you're sure you want to delete a directory *and* its contents (including subdirectories), use the `rm` command with the `-r` (recursive) flag:

```
rm -r apples
```

Work with Programs

Every command you use on the command line, including merely listing files, involves running a program. (So, in fact, you've been using programs throughout this book!) However, some aspects of using programs on the command line aren't entirely obvious or straightforward. In this section, I explain some of the different types of programs you may encounter and how to run them (and stop them). I show you how to edit files on the command line, and I talk about shell scripts, a special kind of program you can create yourself to automate a sequence of tasks.

LEARN COMMAND-LINE PROGRAM BASICS

If you've been reading this book in order, you already know many basics of running programs on the command line. Each time you enter a command such as `ls` or `cp` or `pwd`, you're running a program—and we saw how to change program options and supply additional parameters with arguments and flags earlier (in [What Are Commands, Arguments, and Flags?](#)). However, I think you should know a few other important facts about running programs.

Command-line programs come in a few varieties, which for the sake of convenience I'll lump together in three broad categories. (These are my own terms, by the way; other people may categorize them differently.) You'll have an easier time using the command line if you're aware of the differences.

Basic Programs

Most of the command-line programs you use simply do their thing and then quit automatically. Enter `ls`, for instance, and you instantly get a list of files, after which point `ls` is no longer running. Some of these single-shot programs produce visible output (`date`, `ls`, `pwd`, etc.); some normally provide no feedback at all unless they encounter an error (`cp`, `mv`, `rm`, etc.). But the point is: they run only as long as is needed to complete their task, without requiring any interaction with you other than the original command, with any flags and arguments.

Interactive Programs

A second type of program asks you for an ongoing series of new commands, and in some cases doesn't quit until you tell it to. For example, the command-line program used to change your password is `passwd`. If you enter `passwd`, you see something like the following:

```
Changing password for jk.  
Old password:█
```

You type your old password and press Return, and then the program gives you another prompt:

```
New password:█
```

Type in a new password and you get yet another prompt:

```
Retype new password:█
```

Reenter your new password, as long as it matches the first one, the program changes your password and exits without any further output.

Real change: Note that this procedure really does change the password for your user account, which applies everywhere on your Mac (not just on the command line).

Programs of this sort include `ssh`, which lets you [Log In to Another Computer](#) and `ftp`, which lets you transfer files between computers, among many others. If you're running an interactive program, want to quit it, and can't find an obvious way to do so, try pressing Control-C (see [Stop a Program](#) for more possibilities).

Full-Window Programs

The third broad category of programs is full-window programs—those that are interactive but, instead of handling input and output on a line-by-line basis, take over the entire window (or tab). You've already tried a few of these—`less` and `man` are examples. Some full-window programs helpfully display hints at the top or bottom of the window showing what commands you can use; others require that you've memorized them (or can look them up in a `man` page, perhaps in another window). As with other interactive programs, pressing Control-C usually lets you exit a full-window program if you can't find another way to do so.

Change Your Terminal Type

A curious feature of full-window programs such as [less](#), [top](#), and [man](#) is that once you quit them, everything they previously displayed on screen disappears; for example, you can't scroll back to see something from a man page once you quit [man](#). This behavior (among others) is determined not by your shell but by the specific kind of terminal that Terminal happens to be emulating at any given time. By default, that terminal type is something called `xterm-color`. Without getting into any tedious details, let's just say that `xterm-color` has many virtues, but some people dislike the way it handles full-window programs. If you're one of those people, you can easily switch to a different terminal type.

Follow these steps:

1. Choose Terminal > Preferences and click the Settings button on the toolbar.
2. Click Advanced, and then choose vt102 from the Declare Terminal As pop-up menu.
3. Close the Preferences window.

The change takes effect beginning with the next shell session you open in Terminal.

RUN A PROGRAM OR SCRIPT

Often, running a program requires nothing more than typing its name and pressing Return. However, the process can be a bit trickier in certain cases. Read on to discover how to run programs located in unusual places, as well as scripts (programs packaged in a somewhat different form).

How Your PATH Works

As you know already (see [Understand How Paths Work](#)), every file on your Mac has a path—a location within the hierarchy of directories. So a path of `/Users/jk/Documents/file1` means that `file1` is inside the `Documents` directory, which is in turn inside the `jk` directory, which is inside `Users`, which is at the top, or root, level of the disk (signified by the initial `/`).

But there's another, specialized use of the term *PATH*: when capitalized like this, it refers to a special variable your shell uses that contains a list of all the default locations in which a shell can look for programs.

To run a program, your shell has to be able to find it. But so far, all the commands you've entered have been "bare" program names without specific paths given. For example, to run `less`, you simply enter `less`, but in reality the program you're running is stored in `/usr/bin`. Looking everywhere on your disk for a program would be extremely time-consuming, so how can your shell find it in order to run it? The answer is that when you enter a command without an explicit path, the shell automatically looks in several predetermined locations. That list of locations, which happens to include `/usr/bin`, is your *PATH*.

By default, your *PATH* on Mac OS X Leopard or newer includes all of the following directories:

```
/bin
/sbin
/usr/bin
/usr/local/bin
/usr/sbin
/usr/X11/bin
```

A program in any of these locations is said to be "in your *PATH*." So you can run a program in your *PATH*, regardless of your current location in the file system, simply by entering its name. I encourage you to look through these directories now (try `ls -l /bin`, `ls -l /sbin`, and so on) to get an idea of the hundreds of built-in programs and where they're located.

Most of the programs you'll need to run are already in your *PATH*, and if you download or create new programs, you can put them in one of these locations to make sure you can run them just by entering their names. But what about programs that aren't in your *PATH*? You can either enter the program's full or relative path (for example, `/usr/local/bin/stuff` or `../software/myprogram`), or you can expand your *PATH* to include other directories (I explain how in [Modify Your Path](#)).

Run a Program

To summarize, you can run a program in any of three ways, depending on where the program is located, your current position in the file system, and what's in your PATH:

- **By relative or absolute path:** You can *always* run a program by entering its complete path, such as `/usr/bin/less`, or its relative path from your current location, for example `apples/oranges/program`.
- **In the current directory:** If you're in the same directory as the program you want to run, you might think you could just enter the program's name, but that doesn't work. Instead, you must precede the name with `./` (and no space). For example, to run a program named `counter` in the current directory, enter `./counter`.
- **In your PATH:** To run a program anywhere in your PATH, simply enter the program's name—for example, `less`, `mkdir`, or `man`.

Run a Script

In Mac OS X, as in other varieties of Unix, the programs you run are usually compiled binary files. If you were to open them in a text editor, they'd look like nothing but garbage characters, because they've been put into an optimized, self-contained, machine-friendly format for maximum performance. However, there's another whole category of programs consisting of human-readable text that's interpreted by the computer as it runs instead of being compiled in advance. Programs in this broad category are often referred to as *scripts*, and they're often used to automate or simplify repetitive activities. Just as AppleScript provides a way of writing human-readable programs that run in Mac OS X's graphical environment, scripts of various kinds can run from the command line.

A *shell script* is a series of instructions interpreted, or run, by the shell itself. So, a shell script could consist of little more than a list of commands, just as you would type them manually in a Terminal window. Run the script, and the shell executes all those commands one after the other. (In fact, shell scripts can use variables, conditional tests, loops, math, and much more—but I don't explore those things in this book.) I explain the basics of creating a simple script ahead in [Create Your Own Shell Script](#). By convention, shell scripts usually have an extension of `.sh`.

Other kinds of scripts are written in scripting languages such as Perl, Python, and Ruby, and run by the corresponding interpreter. Perl scripts, by convention, end in the `.pl` extension, Python scripts in `.py`, and Ruby scripts in `.rb`.

Regardless of a script's extension, it's considered good programming practice to include the name and location of the interpreter that should process it on the first line of the script. For example, if a shell script is intended to be interpreted by the `sh` shell, the first line should be:

```
#!/bin/sh
```

The `#!` at the beginning of this line, called a “shebang,” is a marker indicating that what follows it is the path to the interpreter. (You can examine a script using, say, `less` or `cat` to see if it has such a line.)

Because the interpreter is spelled out right in the script, you can run the script just as you would any other program, just by entering its name (if it's in your `PATH`) or its path.

However, if a script doesn't include that line, you must tell it explicitly which shell or other interpreter to run it with. You do that by entering the interpreter's name with the path to the script as an argument. For example:

```
sh ~/Documents/my-shell-script.sh
perl ~/Documents/my-perl-script.pl
python ~/Documents/my-python-script.py
ruby ~/Documents/my-ruby-script.rb
```

RUN A PROGRAM IN THE BACKGROUND

Most of the time when you run a program, it does its thing, and then you quit it (or it quits by itself). While a program is running—whether that takes a second or an hour—it takes over your shell and thus the Terminal window or tab in which the shell is running. If you expect a program to take some time to complete its task, or if you want a program to keep running even after you exit the shell, you can run it in the background. Background programs let you do other tasks in the same Terminal window or tab, and, if necessary, they keep going even after you quit Terminal.

To run a program in the background, you simply put a space and an ampersand (&) after the program name (and any flags or arguments). For example, suppose you want to compress a folder containing hundreds of large files. Ordinarily, you might use a command like `zip -r archive.zip apples`. To run that command in the background instead, enter this:

```
zip -r archive.zip apples &
```

While a program is running in the background, you'll see no feedback or output. If it's a program that simply accomplishes a task (such as copying or compressing files) and then quits automatically, then you'll see a message stating that it's finished—not *immediately* afterwards, but the next time you execute a command or even just press Return to create a new command line. The message saying a process is finished looks something like this:

```
[1]+  Done                zip -r archive.zip apples
```

Note: Programs designed to run in the background are called *daemons* (pronounced “demons”). Examples include database and Web servers, firewalls, and some backup programs. You wouldn't use the term “daemon,” however, for an ordinary program you opt to run in the background temporarily.

SEE WHAT PROGRAMS ARE RUNNING

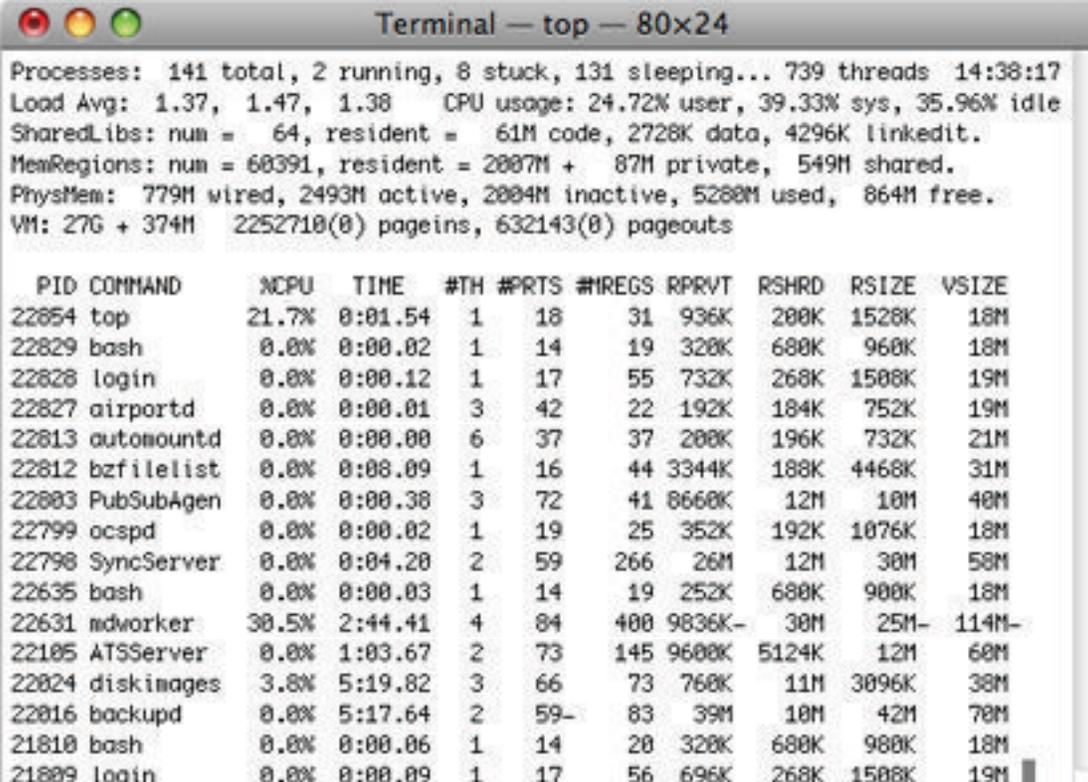
Here's a thought question: How many programs are running on your Mac right this minute? If you glance at the active icons in your Dock and conclude that the number is, say, a dozen, you haven't even scratched the surface. For example, as I type these words, my Dock tells me I have 16 programs running, but in reality the total number is 135! Besides the visible programs like Mail and Safari, that figure includes many background programs that are part of Mac OS X—the Spotlight indexer, the Java virtual machine, Time Machine, and lots of others that perform important but little-noticed functions behind the scenes. It also includes my `bash` shell running in Terminal, and every program currently running in that shell.

Processing: Roughly speaking, the term process is used to describe programs (of any sort) that are actively running, as opposed to those that are merely occupying space on your disk. The commands and procedures I describe in this section are concerned only with active programs, and therefore I use the term “process” to describe them.

You may be aware of Activity Monitor (in `/Applications/Utilities`), which lists all this information and more. In the command-line environment, too, you can list all your Mac’s processes (visible and invisible) and get a variety of useful facts about them. The two most commonly used command-line programs for discovering what’s running on your Mac are `top` and `ps`.

Top

The `top` command is the nearest command-line equivalent to Activity Monitor. Enter `top` and you get a full-window list of all your running processes, updated dynamically. **Figure 4** shows an example.



```
Terminal — top — 80x24
Processes: 141 total, 2 running, 8 stuck, 131 sleeping... 739 threads 14:38:17
Load Avg: 1.37, 1.47, 1.38 CPU usage: 24.72% user, 39.33% sys, 35.96% idle
SharedLibs: num = 64, resident = 61M code, 2728K data, 4296K linkedit.
MemRegions: num = 68391, resident = 2887M + 87M private, 549M shared.
PhysMem: 779M wired, 2493M active, 2884M inactive, 5288M used, 864M free.
VM: 27G + 374M 2252718(0) pageins, 632143(0) pageouts

  PID COMMAND           %CPU   TIME   #TH  #PRTS  #MREGS  RPRVT  RSHRD  RSIZE  VSIZE
22854 top                 21.7%  0:01.54  1    18     31  936K   288K   1528K   18M
22829 bash                 0.0%   0:00.02  1    14     19  328K   688K   960K   18M
22828 login                0.0%   0:00.12  1    17     55  732K   268K   1508K   19M
22827 airportd            0.0%   0:00.01  3    42     22  192K   184K   752K   19M
22813 automountd         0.0%   0:00.00  6    37     37  288K   196K   732K   21M
22812 bzfilelist         0.0%   0:08.09  1    16     44 3344K   188K   4468K   31M
22883 PubSubAgen         0.0%   0:00.38  3    72     41 8668K   12M    10M    48M
22799 ocspd              0.0%   0:00.02  1    19     25  352K   192K   1076K   18M
22798 SyncServer         0.0%   0:04.28  2    59     26  26M    12M    30M    58M
22635 bash                 0.0%   0:00.03  1    14     19  252K   688K   900K   18M
22631 mdworker            38.5%  2:44.41  4    84     48 9836K   38M    25M   114M
22185 ATSServer          0.0%   1:03.67  2    73    145 9688K   5124K  12M    68M
22824 diskimages         3.8%   5:19.82  3    66     73  768K   11M    3896K   38M
22816 backupd            0.0%   5:17.64  2    59     83   39M    18M    42M    78M
21818 bash                 0.0%   0:00.06  1    14     28  328K   688K   988K   18M
21889 login                0.0%   0:00.09  1    17     56  696K   268K   1588K   19M
```

Figure 4: In the `top` window, you get a list of all the processes currently running on your Mac.

By default, the `top` command lists several pieces of information for each process, including the following particularly interesting ones: PID (process ID), COMMAND (the process name), %CPU (how much CPU power the process is using), TIME (how long the process has been running), RSIZE (how much physical RAM the process is using) and VSIZE (how much virtual memory the process is using).

I won't go into great detail about everything you see here (try `man top` to learn more), but I do want to call your attention to a few salient points and offer some `top` tips:

- **Pruning the list:** You almost certainly have many more processes than can fit in your window at one time, even if you make your window very large. So you can restrict the number of items `top` shows at a time using the `-n` (number) flag, followed by the number of items to show (`top -n`).
- **Sorting the list:** By default, `top` lists processes in reverse order of PID, which basically means the processes at the top of the list are the ones launched most recently. You can adjust the sort order with the `-o` (order) flag—for example, enter `top -o cpu` to list processes in order of how much CPU power they're using, or enter `top -o rsize` to list processes in order of how much physical RAM they're using.
- **Top at the top:** Depending on what else is running on your Mac at the moment, `top` itself may be at or near the top of the list, even when sorted by CPU usage. Don't be alarmed: the effect is caused by the way `top` gathers its data.
- **Customizing the list:** You can combine flags to customize your display. For example, enter `top -n 20 -o cpu` to list only the top 20 processes by CPU usage.
- **Quitting:** To quit `top`, just type the `Q` key (by itself).

Ps

Whereas `top` is dynamic, you may want simply to get a static snapshot of the processes running at any moment. For that, the best command is `ps` (process status). If you enter `ps` by itself, you get a list of the processes running in your current shell—which, in all likelihood, is just bash itself.

The list includes the PID, the TTY (or terminal name), time since launch, and command name for each process:

```
PID TTY          TIME CMD
22635 ttys001      0:00.06 -bash
```

You can expand the amount of information that `ps` provides using flags. For example, to include not only processes in the current shell session but also those from other sessions (yours or those belonging to other users), enter `ps -a`. To show processes that aren't running in a shell at all (including regular Mac OS X applications and background processes), enter `ps -x`. Combine the two (`ps -ax`) to show all the processes running on your Mac.

Of course, although `ps -ax` provides lots of information, it might actually be too much to be useful. So you can filter the output from the `ps` command by using a couple of spiffy Unix tricks. First, you add the pipe (`|`) character (which you get by typing Shift-`\`) to channel the output from `ps` into another program. The other program, in this example, is `grep`, a powerful pattern-matching tool. So, enter `ps -ax | grep` followed by a space and some text, and what you get is a list of all and only the running processes whose listing includes that text. For example, to list all the processes that are running from inside your `/Applications` directory, enter this:

```
ps -ax | grep /Applications
```

Get a grep: A curiosity of this command is that the `grep` process itself will appear in the list, because `grep` includes `/Applications` as an argument! If that bothers you and you want to exclude `grep` itself, add the following after `/Applications` and a space: `| grep -v grep`. The same applies for the next example.

Or, to show only the processes whose names include the characters `sys` (in any combination of upper- and lowercase), try this:

```
ps -ax | grep -i sys
```

Just a taste: I can't even begin to do justice to the incredible power that the pipe operator and the `grep` command offer, because they're both far too complex to cover in detail in this book. But you can see additional examples of how to use them in [Command-Line Recipes](#).

STOP A PROGRAM

As we've seen, most command-line programs quit automatically when they finish performing their functions, and full-window programs usually have a fairly obvious way of quitting them (for example, pressing `Q` in the case of `less` or `man`). However, if a program doesn't quit on its own, or if you need to unstick one that's stuck (even if it's a graphical Mac OS X application!), you can use one of several techniques.

Ask Politely

If a command-line program won't quit on its own, the first thing to try is pressing Control-C. In this context, it's approximately like pressing Command-Q in a regular Mac OS X application—it tells the process to quit, but to do so in a controlled way (saving open files and performing any other necessary cleanup operations).

Kill (Humanely)

What if you want to stop a program that's not running in the current shell? If it's a graphical Mac OS X application, or an invisible background process, or a program running in another shell, you can send it a "Quit" signal remotely. The command you use to do this is `kill`. That sounds extreme, but, in fact, when `kill` is used on its own, it sends a program the same sort of polite request to terminate that Control-C does.

***You're killing me:** You can only kill processes you own (that is, ones started under your user account). To kill another user's processes, you must use `sudo` (see [Perform Actions as the Root User](#)).*

The catch is that you have to know how to refer to the program you want to kill. Here there are two options:

- **By PID:** If you can find the process's PID (process ID)—using `top`, `ps`, or even Activity Monitor—you can simply enter `kill` followed by that number. For example: `kill 1342`
- **By name:** If you don't know the process's PID, or can't be bothered to find out—but do know its name—you can quit it by name using a variant of `kill` called `killall`. Simply follow `killall` with the program's name. For example: `killall iTunes`

You must enter the name exactly as it appears in `top`, `ps`, or Activity Monitor. For example, if you want to quit Excel, you must enter `killall "Microsoft Excel"` (quotation marks added because there's a space in the name).

Kill (with Extreme Prejudice)

If a program fails to respond to Control-C or to the standard `kill` or `killall` command, it's time to pull out the big guns. By adding the `-9` flag to the `kill` command, you turn a polite request into a brutal clobbering that can terminate almost any process. When you use the `kill -9` command, you must give it the process's PID; the `-9` flag doesn't work with `killall` to force-quit a process by name. For example:

```
kill -9 1342
```

If even `kill -9` doesn't stop a process—and I've seen that happen more than once—it is likely stuck beyond the power of any software command, and your only choice is to restart the computer.

EDIT A TEXT FILE

Earlier I showed you how to view the contents of text files, but you may also need to modify them. For that, you can use any of several command-line text editors. Using a command-line text editor is often quicker and easier than opening a text file in a program like TextEdit—especially for files that don't appear in the Finder—and is less likely to cause problems with file formats or permissions.

If you ask a hardcore Unix geek what text editor he uses, he (there are far too few female Unix geeks) will probably answer `vi`. (That's "vee-eye," not "vie," by the way.) It's a very powerful text editor that's been around forever, and because a lot of programmers cut their teeth on `vi` and then proselytized future generations, it's become a sort of badge of honor to be skilled in using `vi`.

Mac OS X includes `vi`, but I'm not going to tell you how to use it. As command-line programs go, `vi` has the most opaque user interface I've seen. Until you get used to `vi`'s oddities and memorize its commands, you can't even change a letter in a text document without referring to a manual. Powerful or not, from a usability standpoint, `vi` is hideous. I just want you to know about `vi` so that when someone asks you why you don't use it, you can give the correct response: "Life is too short."

Happily, you can use several other fine text editors. There's the venerable `emacs`, which is less obnoxious than `vi` while still being fabulously flexible. But I'm going to recommend what you might think of as the TextEdit of command-line text editors: a simple, straightforward, and adequately powerful program called `nano`.

Note: The `nano` editor is an "enhanced clone" of an earlier editor called `pico`; they have almost identical interfaces and feature sets. In much the same way as `more` and `less`, Mac OS X includes a program called `pico` and a program called `nano`, but they're the same, and if you try to run `pico`, `nano` is what actually runs.

To edit a text file in `nano`, use a command like the following:

```
nano file1
```

If `file1` is already present, `nano` opens it; otherwise, it opens a blank file that will be called `file1`. **Figure 5** shows a text file open in `nano`.

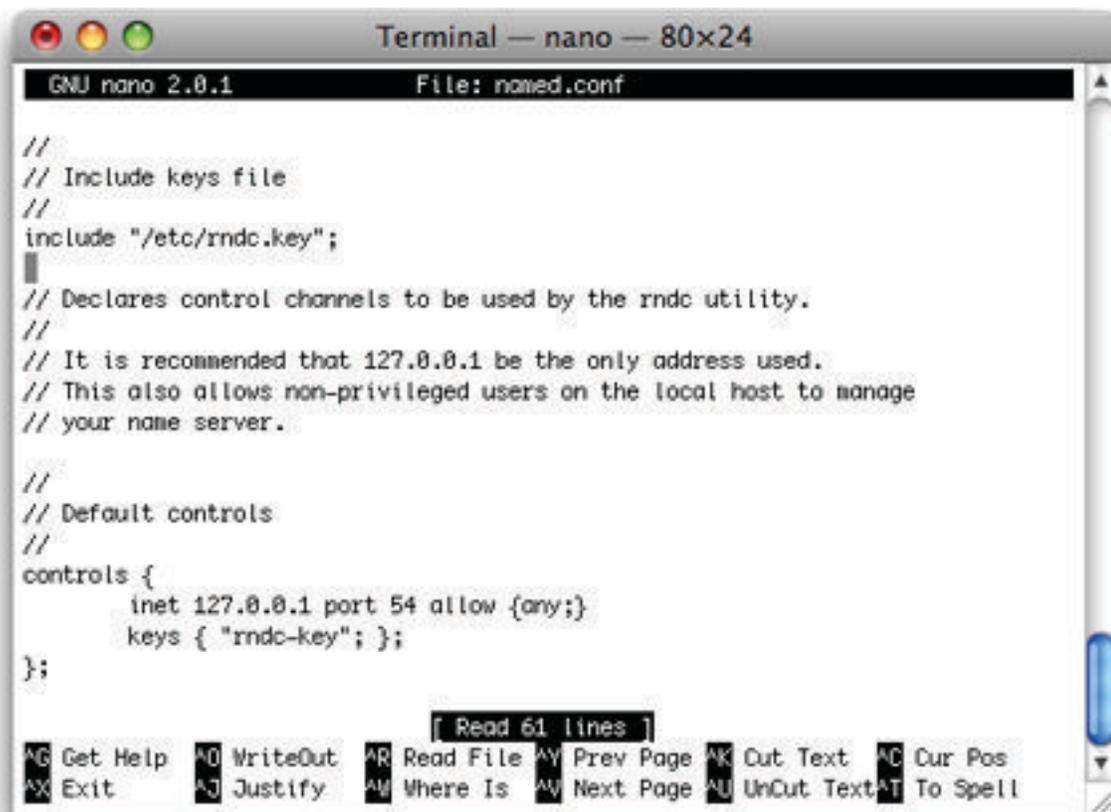


Figure 5: A text file open in the `nano` text editor. The menu of keyboard controls is at the bottom of the window.

Natural selection: You can select text in a `nano` screen using your mouse, and you can even copy it using `Edit > Copy` or `Command-C`. But that's pretty much the only reason to use your mouse in `nano`.

One of the reasons `nano` is easy to use is that editing is straightforward. To insert text at the cursor location, simply type—or paste the contents of your Clipboard by choosing `Edit > Paste` or pressing `Command-V`. To delete the character to the left of the cursor, press the `Delete` key; to delete the character at the cursor, press the `Forward Delete` key (if your keyboard has one). To delete the entire current line, press `Control-K`.

Uncut: The `nano` editor doesn't have an `Undo` command as such, but if you cut a line of text with `Control-K` and want to restore it, you can press `Control-U` to “uncut” it.

Other than those basics, here are the most important things you should know how to do in `nano`:

- **Save:** To save the current file, press `Control-O` (`WriteOut`).
- **Quit:** To quit `nano`, press `Control-X` (`Exit`). If you've made any changes to the document that you haven't yet saved, `nano` prompts you to save the file before exiting. Press `N` to discard changes and quit immediately, `C` to cancel and stay in `nano`, or `Y` to save changes and exit. If you do save changes, `nano` verifies that you want to keep the existing file name (if you don't, you can type a new one). Press `Return` after verifying the file name.
- **Find:** To find text within the file, press `Control-W` (`Where Is`). Type the text you're searching for (case doesn't matter) and press `Return`. The cursor jumps to the next spot in the document where that string appears. Repeat this procedure to do additional searches.

Those commands alone should enable you to do almost everything you need to do in `nano`. To learn about additional `nano` features and short-cuts, press `Control-G` to view its online help.

CREATE YOUR OWN SHELL SCRIPT

Before I wrap up this section on running programs, I want to give you a tiny taste of creating your own shell scripts. Scripting is a bit like learning chess: you can pick up the basics in a few minutes, but it may

take years to master all the subtleties. So I'm not going to teach you anything about *programming* as such, just the mechanics of creating and using a simple script. I want you to have enough familiarity with the process that you can successfully reproduce and run shell scripts you may run across in magazines, on Web sites, or even in this book (see [Command-Line Recipes](#), which includes a couple of shell scripts).

You can create and run a shell script in six easy steps; in fact, you can arguably combine the first four into a single process. But one way or another, you must make sure you've done everything in the list ahead.

Step 1: Start with an Empty Text File

Scripts are plain text files, so you should begin by creating one in a text editor. You can make a shell script in TextEdit, BBEdit, or even Word, but that requires extra steps. So I suggest using `nano`, as described in [Edit a Text File](#). For the purpose of demonstration, name your script `test.sh`. (Remember from [Run a Script](#) that the `.sh` extension isn't mandatory, but it can help you keep track of which files are scripts.)

Before you create this file, I suggest using `cd` (all by itself!) to ensure that you're in your home directory. (You can put scripts anywhere you want, but for now, this is a convenient location.) That done, enter `nano test.sh`. The `nano` text editor opens with a blank file.

Step 2: Insert the Shebang

The first line of your script should include the "shebang" (`#!`) special pointer (see [Run a Script](#)) to the shell it will use. Since this book is all about the `bash` shell, we'll use that one. Type the following line:

```
#!/bin/bash
```

Step 3: Add One or More Commands

Below the shebang line, you enter the commands your script will run, in the order you want them executed. Your script can be anything from a single one-word command to thousands of lines of complex logic. For now, let's keep things simple. Starting on the next line, type this:

```
echo "Hello! The current date and time is:"  
date  
echo "And the current directory is:"  
pwd
```

The `echo` command simply puts text on the screen—and you’ve seen the `date` and `pwd` commands. So, this script displays four lines of text, two of which are static (the `echo` lines) and two of which are variable.

Step 4: Close and Save the File

To save the file, press Control-O and press Return to confirm the file name. Then press Control-X to exit `nano`.

Step 5: Enable Execute Permission

The only slightly tricky thing about running scripts—and the step people forget most often—is adding execute (run) permission to the file. (I say more about this later, in [Understand Permission Basics](#).) To do this, enter `chmod u+x test.sh`.

Step 6: Run the Script

That’s it! To run the script, enter `./test.sh`. It should display something like this:

```
Hello! The current date and time is:  
Mon Feb 16 19:58:21 CET 2009  
And the current directory is:  
/Users/jk
```

For fun, try switching to a different directory (say, `/Library/Preferences`) and then running the script again by entering `~/test.sh`. You’ll see that it shows your new location.

Any time you need to put a new script on your system, follow these same steps. You may want to store the scripts you create somewhere in your PATH (see [How Your PATH Works](#)), or add to your PATH (see [Modify Your Path](#)), to make them easier to run.

Tip: For more on shell scripting, read Apple’s Shell Scripting Primer at <http://developer.apple.com/documentation/OpenSource/Conceptual/ShellScripting/>.

Customize Your Profile

Now that you know the basics of the command line and Terminal, you may find some activities are a bit more complicated than they should be, or feel that you'd like to personalize the way your shell works to suit your needs. One way to exercise more control over the command-line environment is to customize your profile, a special file the `bash` shell reads every time it runs. In this section, I explain how the profile works and how you can use it to save typing, customize your prompt, and more.

HOW PROFILES WORK

A profile is a file your shell reads every time you start a new session that can contain a variety of preferences for how you want the shell to look and behave. Among other things, your profile can customize your `PATH` variable (see [How Your PATH Works](#)), add shortcuts to commands you want to run in a special way, and include instructions for personalizing your prompt. I cover just a few basics here.

What you should understand, though, is that for complicated historical reasons, you may have more than one profile (perhaps as many as four or five!), and certain rules govern which one is used when.

When you start a new shell session, `bash` first reads in the system-wide default profile settings, located at `/etc/profile`. Next, it checks if you have a personal profile. It first looks for a file called `~/.bash_profile`, and if it finds one, it uses that. Otherwise, it moves on to look for `~/.bash_login` and, finally, `~/.profile`. Of these last three files, it loads only the first one it finds, so if you have a `.bash_profile` file, the others, if present, are ignored.

Note: You may also read about a file called `.bashrc`, which `bash` reads in only under certain unusual conditions that you're unlikely to encounter when using Terminal on Mac OS X.

Because `.bash_profile` is the first user-specific profile to be checked, that's the one I suggest you use.

Note: Customizations you make in `.bash_profile` (or any of the other profile files mentioned here) apply only in a shell session; they aren't used by shell scripts (see [Create Your Own Shell Script](#)). As a result, when writing a script, you should always spell out complete paths and assume default values for all variables.

EDIT `.BASH_PROFILE`

To edit `.bash_profile` in `nano`, simply enter the following:

```
nano ~/.bash_profile
```

If the file already exists, `nano` opens it for editing; if not, it prompts you to create the file when you save or quit the program.

This file is a simple text file, and unlike shell scripts, it doesn't use a shebang. Just add one or more lines to specify the changes you want (as described on the following pages). When you're finished editing `.bash_profile`, save it (Control-O) and close it (Control-X). Changes take effect with the next shell session (window or tab) you open.

CREATE ALIASES

In the Finder, an alias is a small file that serves as a pointer to another file. In the command-line environment, however, the word *alias* means a shortcut in which one command substitutes for another.

For example, suppose you're used to command-line conventions from DOS and Windows, in which you enter `dir` (directory) to list your files. If you want to use that same command in Mac OS X, you can make an alias, so that entering `dir` runs the `ls` command. Or, maybe there's a lengthy command you use frequently, and you want to run it with fewer keystrokes. No problem: you can use (for instance) `pp` to mean `cp *.jpg ~/Pictures/MyPhotos`.

To create an alias, put a new line in your `.bash_profile` consisting of the word `alias`, a space, the shortcut you want to use, and `=` with the command you want to run inside the quotation marks. For example, to use the command `dt` as a shortcut for the `date` command, enter this:

```
alias dt="date"
```

Aliases can include flags and arguments, and if you enter a shortcut that's identical to an existing command, your alias takes precedence. For example, if you always want to show file listings in the long format, instead of typing `ls -l` every time, you can create an alias so typing `ls` gives you the same result:

```
alias ls="ls -l"
```

Or, suppose you've taken my advice to heart to always use the `-i` flag with `cp` (copy) and `mv` (move), to display a warning if the command is about to overwrite an existing file. You could add aliases to new, easy-to-remember commands like `copy` and `move`, respectively, with those options pre-configured:

```
alias copy="cp -i"  
alias move="mv -i"
```

Safety net: *You could set up aliases so entering `cp` or `mv` would include the `-i` flag, but I recommend against it because you might get into a habit of using `cp` and `mv` carelessly, assuming you'll be warned of any impending overwrite. That could lead to data loss if you find yourself using the command line on a computer that doesn't have the same aliases configured.*

MODIFY YOUR PATH

As I explained in [How Your PATH Works](#), when you run a program by entering just its name, your shell looks for it in several predetermined directories. You may want to specify additional places where programs or scripts are located, to make it easier to run them. For example, if you're experimenting with your own scripts and you store them all in `~/Documents/scripting`, you should add that directory to your PATH.

To add a directory to your PATH, put this in your `.bash_profile`:

```
export PATH=$PATH:/your/path/here
```

For example, to add the directory `~/Documents/scripting`, enter this:

```
export PATH=$PATH:~/Documents/scripting
```

You can add as many of these `export` statements as you need.

CHANGE YOUR PROMPT

Your command prompt—the string of characters at the beginning of every command line—normally looks something like this:

```
Joes-MacBook-Pro:~ jk$ █
```

You can modify this by adding a line to your `.bash_profile` that begins with `PS1=` and ends with whatever you want your prompt to be. For example, if you enter this:

```
PS1="I love cheese! "
```

then the next time you open a shell, your prompt looks like:

```
I love cheese! █
```

Tip: Always enclose your prompt in quotation marks, and include a space before the closing quotation mark, to make sure you can easily see where the prompt ends and commands begin.

Prompts can include variables. Some common ones are these:

- `\u`: Your short user name
- `\h`: Your computer's name
- `\s`: The name of the current shell
- `\w`: The current directory
- `\d`: The current date, in the format “Mon Feb 16”
- `\@`: The current time, in 12-hour format with AM/PM

So, to make the following prompt:

```
jk 09:08 PM ~ * █
```

Enter this:

```
PS1="\u \@ \w * "
```

Tip: For another example of a profile customization, see the recipe [Read man Pages in BBEdit](#).

Bring the Command Line into the Real World

So far in this book I've largely ignored Mac OS X's graphical interface, treating the command-line environment as a separate world. In fact, because the command-line interface and the graphical interface share the same set of files and many of the same programs, they can interact in numerous ways.

In this section I discuss how your shell and the Finder can share information and complement each others' strengths—giving you the best of both worlds.

GET THE PATH OF A FILE OR FOLDER

Suppose you want to perform some command from the command line on a file or folder you can see in the Finder, but you don't know the exact path of that folder—or even if you do, you don't want to type the whole thing. You're in luck: there's a quick and easy way to get the path of an item from the Finder into a Terminal window.

To get the path of an item in the Finder, do the following:

1. In a Terminal window, type the command you want to use, *followed by a space*. The space is essential!
2. Drag the file or folder from the Finder into the Terminal window.

As soon as you release the mouse button, Terminal copies the path of the file or folder you dragged onto the command line. It even escapes spaces and single quotation marks with backslashes for you automatically! You can then press Return to run the command.

For example, suppose you want to use the `ls -l@` command to list the contents of a folder with their extended attributes, which you can't see in the Finder. You could type this:

```
ls -l@
```

(Don't forget the space after the @!) Then drag a folder into the Terminal window, as shown in **Figure 6**.

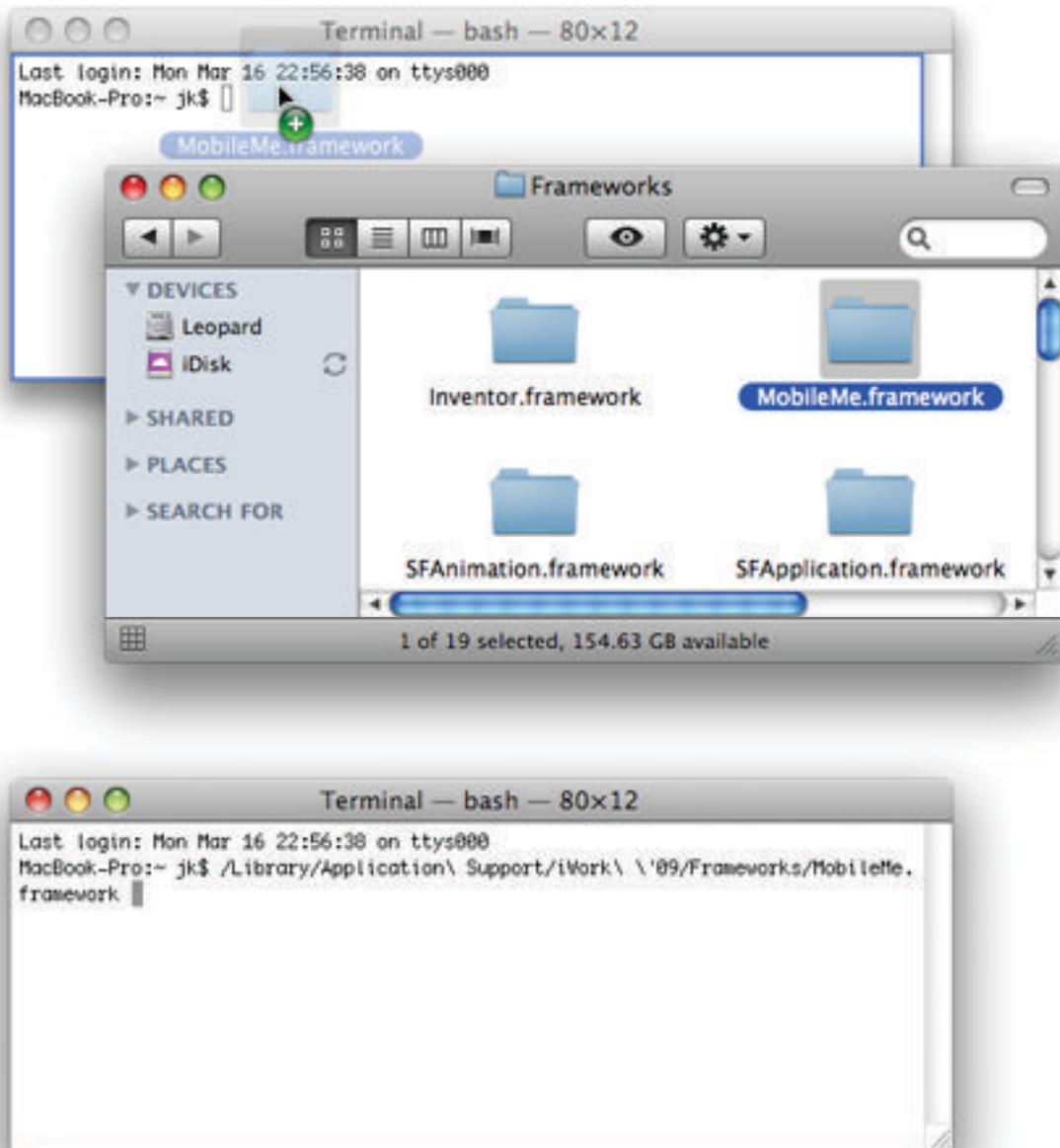


Figure 6: Drag a file or folder into the Terminal window (top); when you release the mouse button, you get that item's full path (bottom).

OPEN THE CURRENT DIRECTORY IN THE FINDER

On occasion you may be using the command line deep in Mac OS X's directory hierarchy (whether or not it's a location that's normally visible in the Finder) and want to open the current directory as a folder in the Finder.

You can do so with one of the simplest commands ever:

`open .`

That's `open` followed by a space and a period. And that's all it takes!

OPEN A HIDDEN DIRECTORY WITHOUT USING TERMINAL

If all you want to do is open a directory that's normally hidden, you need not even open Terminal to do so, as long as you know its location. Just choose `Go > Go to Folder` in the Finder. In the dialog that appears, type the whole path of the directory you want to view (**Figure 7**) and click `Go`. That directory opens as a folder in the current Finder window.

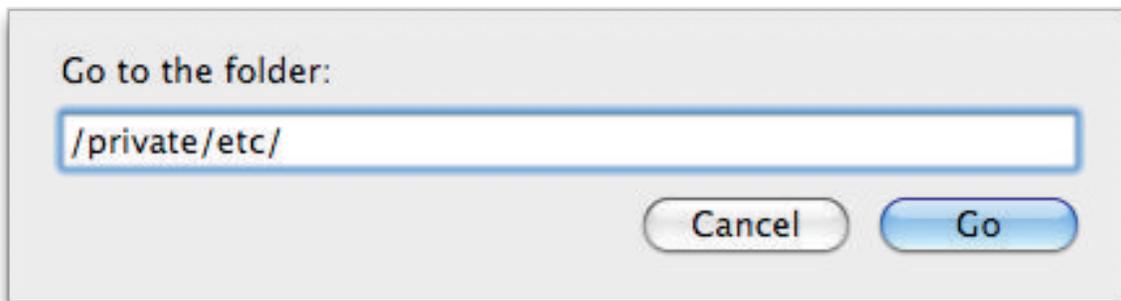


Figure 7: Open almost any directory, even hidden ones, in the Finder using the `Go to Folder` dialog.

Tip: When you're typing a path in the `Go to the Folder` dialog, you can use tab completion just as in the `bash` shell (see [Use Tab Completion](#)); that can save you considerable typing and guessing.

OPEN THE CURRENT FOLDER IN TERMINAL

Suppose you're looking at some folder in the Finder and you realize you want to run a command-line program on the items in it, such as one that renames a bunch of files. You could open Terminal and type in the path to the folder, but that can be cumbersome. Wouldn't it be great if, instead, you could just click a button and have a new shell session open in Terminal, with the current directory already set to the folder you were just looking at in the Finder?

In fact, you can do exactly that, with the help of a free AppleScript application written by Marc Liyanage. To install it, follow these steps:

1. Download OpenTerminalHere from <http://www.entropy.ch/software/applescript/>.
2. Drag the application from your Downloads folder to `/Applications/Utilities` (or wherever you want to keep it).
3. Drag the application from its new home onto the toolbar of any Finder window. (You may have to keep the mouse button down for a few seconds until a plus (+) icon appears, signifying that it's ready to add a button to your toolbar.) Move your pointer to where you want your new button to appear, and release the button.

From now on, the button (shown in **Figure 8** with an optional Leopard-style icon) appears in the toolbar of every Finder window. You can click that button at any time to open Terminal and start a new shell session with the directory preset to your current location.

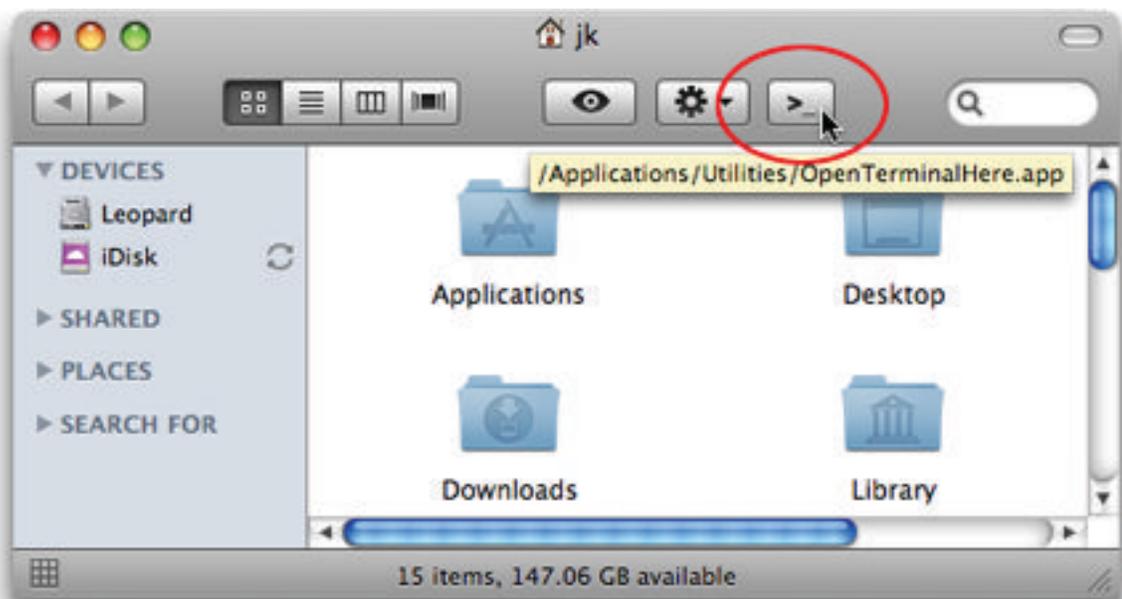


Figure 8: Click the new OpenTerminalHere button in the toolbar of any Finder window to open it in a new shell session.

OPEN A MAC OS X APPLICATION

If you ever need to open a standard Mac OS X application while on the command line, you can do it by entering the `open` command with the `-a` (application) flag. For example, to open Safari, just enter this:

```
open -a Safari
```

The `open -a` command is amazingly smart. You don't have to tell it where the application is located; it can be in `/Applications`, or `/Applications/Utilities`, or anywhere else on your disk—it doesn't matter. And you need not spell out “Safari.app” or go through any other complicated steps to get to the application.

OPEN A FILE IN MAC OS X

Similarly, you can open a particular file that you see on the command line in the default Mac OS X application for that file type—or another application. For example, if the current directory contains a graphic named `flowers.jpg`, you can open it in its default application (probably Preview) like so:

```
open flowers.jpg
```

But if you prefer to open it in Photoshop Elements, just enter this:

```
open -a Adobe\ Photoshop\ Elements flowers.jpeg
```

(Note the backslash before each space in the application name; you could also put the application name inside quotation marks.)

Log In to Another Computer

Every time you connect to another Mac to share files or other system resources, you are, in a way, logging in to that other Mac. However, in this section I describe a particular way of logging in to a remote computer—doing so using SSH (secure shell), which gives you access to the other computer’s command-line interface from within your own Mac’s command-line interface. Logging in via SSH lets you interact with another computer in the same way you interact with your current Mac from inside a Terminal window.

You can connect to almost any Mac, Unix, or Unix-like computer (and some Windows computers) using SSH, provided the other computer has SSH enabled. (To enable incoming SSH access on a Mac, check the Remote Login box in the Sharing pane of System Preferences.) If you log in to another Mac, everything should look quite familiar, whereas other operating systems may follow different conventions. For the purposes of this chapter, I assume that the remote computer is at least running a Unix-like system so that most of the things you’ve learned in this book still apply.

START AN SSH SESSION

The easiest way to start an SSH session from Terminal is to begin in an existing shell session. Then follow these steps:

1. Enter the following, substituting your user name on the remote computer for `user-name`, and the remote computer’s IP address or domain name for `remote-address`:

```
ssh user-name@remote-address
```

2. If this is the first time you’re connecting to this particular remote computer, you will see a message something like the following:

```
The authenticity of host 'macbook-pro.local  
(fe80::20c:74ee:edb2:61ae%en0)' can't be established.
```

```
RSA key fingerprint is
d0:15:73:75:04:9a:c3:2d:5b:b1:f8:c0:7d:83:52:ef.
```

```
Are you sure you want to continue connecting (yes/no)?
```

After reading the sidebar “SSH Security Considerations,” below, assuming you’re still comfortable connecting, type `yes` and press Return.

3. Text similar to the following appears on screen:

```
Warning: Permanently added 'macbook-pro.local.,
fe80::20c:74ee:edb2:61ae%en0' (RSA) to the list of known hosts.
```

And following that is a password prompt. Type your password for the remote computer and press Return.

Blind typing: As you type your password, no text appears—not even bullet or asterisk characters. That’s normal.

Assuming the remote computer accepts your password, it presents you with a new command prompt, often (but not always) accompanied by a brief welcome message.

SSH Security Considerations

SSH is a highly secure protocol, so what’s with these fingerprints and warnings?

The simplified explanation here for using SSH relies on your trust that the computer you’re connecting to is the one you think it is—that no one has hijacked your connection. The fingerprint is a unique identifier tied to each computer, and by agreeing (in Step 2) that the fingerprint is correct, you’re saying you trust this fingerprint for that computer. How would you know you can? If you’re connecting to another Mac on your home network, you can safely take it for granted. If you’re connecting to a computer at the office, a Web server, or some other commercial computer, ask the system administrator who’s in charge of it to confirm its fingerprint, and make sure it matches what you see.

Once you accept a fingerprint, your Mac remembers it and checks to see that the fingerprint matches that remote computer every time you connect to it. If it doesn’t, it may be a sign that a hacker is trying to trick you into connecting to the wrong computer.

Tip: To learn much more about the security implications of SSH connections and how to avoid potential problems, read [Take Control of Your Wi-Fi Security](#).

RUN COMMANDS ON ANOTHER COMPUTER

Once you're logged in to another computer, you run commands on it exactly the same way you do on your own Mac—just enter the command and any necessary flags and arguments.

However, you should be aware of a few potential “gotchas” when connecting to other computers:

- Your default shell on the other computer might not be `bash`, so some commands may not work the way you expect. Usually—though not always—you can switch to the `bash` shell, if it's not already running, simply by entering `bash`.
- Your `.bash_profile` (see [Customize Your Profile](#)) applies only to the `bash` shell running on your own Mac—*not* the shell that's running on the remote Mac! So your existing aliases, `PATH` variable, and other settings may not work. If you have sufficient permission, you can of course create a `.bash_profile` on the remote computer as well.
- If the other computer is a Mac, and especially if it's running the same version of Mac OS X that you are, you can assume that most programs will be in the same locations. But be aware that a program you want to use could be missing, located somewhere else, or configured in a way that denies you access.

END AN SSH SESSION

To close your remote connection, simply enter `exit`. You return to your existing shell session on your own Mac. As is the case when exiting your own shell session, it's always best to use `exit` to end a remote session gracefully, shutting down any processes that may be running and doing associated clean-up tasks.

Venture a Little Deeper

As I said in the [Introduction](#), this book isn't designed to turn anyone into a propellerhead; it's all about basic command-line proficiency. Even so, some activities you may have to perform involve some slightly geekier concepts.

In this section, I introduce you to the notions of file permissions, owners, and groups, which are essential items to understand for many command-line tasks. I also explain how to temporarily assume the power of the root user using the [sudo](#) command.

UNDERSTAND PERMISSION BASICS

As you may recall from [See What's Here](#), when you list files in the long format (`ls -l`), you can see the permissions, owner, and group of each file and directory. Every file in Mac OS X has all these attributes, and you should understand how they work because they influence what you can and can't do with each item.

Note: This section barely begins to scratch the surface of permissions. To learn the full details, I heartily recommend reading Brian Tanaka's [Take Control of Permissions in Leopard](#).

Before I get into how you read or change permissions, I want to describe the basic options. Put simply, permissions consist of three possible activities (reading, writing, and executing), performed by any of three types of user (the file's owner, the file's group, and everyone else). Three types of permission multiplied by three types of user equals nine items, each of which can be specified individually for every file and folder.

Read, Write, and Execute

Someone with permission to *read* a file can open it and see what's in it. Someone with *write* permission can modify an item or delete it. *Execute* permission, for a file, means it can be run (that is, it can behave as a program or script); for a directory, execute permission means someone can list its contents.

On the command line, read permission is abbreviated with an `r`, write permission is abbreviated with a `w`, and execute permission is abbreviated with an `x`.

User, Group, and Everyone Else

Every file and folder specifies read, write, and execute permissions for the following types of user:

- **User:** In terms of file permissions, the term *user* means the owner of a file or directory. (The user may be a person, like you, or it may be a system process, such as `www`—Mac OS X’s built-in Web server.)
- **Group:** Each file and directory also has an associated group—one or more users for whom a set of permissions can be specified. That group could have just one member (you, for example), or many. Mac OS X includes several built-in groups, such as `admin` (all users with administrator access), `staff` (all standard users without administrative access), and `wheel` (which normally contains only the root user—see [Perform Actions As the Root User](#)); you can also create your own groups.
- **Others:** Every user who is neither the owner nor in the file’s group is lumped into the “others” category.

Reading Permissions, Owner, and Group

To illustrate how this all works, suppose you find the following two items in a certain directory by entering `ls -l` (list in long format):

```
drwxr--r--  15 jk    admin    510 Aug 27 15:02 fruits
-rw-r--r--   2 root  wheel   1024 Sep 02 11:34 lemon
```

For the purposes of this section, we care about just three of the items on each line (apart from the item’s name, at the end). The initial group of characters (like `drwxr--r--`) constitutes the permissions, and the two names in the middle (like `jk admin`) are the user and group, respectively. For now, you can ignore all the other information.

Directory or Not?

The first character of the permissions string tells you whether the item in question is a directory or a regular file. So in the first example (`drwxr--r--`), the item `fruits` is a directory because its permissions string starts with a `d`. The second item, `lemon`, has a hyphen (`-`) in the first slot, which means it’s not a directory (in other words, it’s a file).

Three Permissions, Three Sets

The remaining nine positions in the mode specify the three possible permissions for user (the first three characters), the group (the middle three), and others (the final three).

In each set of three characters, the order is always the same: **r** (read), **w** (write), and **x** (execute). So picture a template with 10 slots, of which the first is the **d** character for directories:

directory	user	group	others	← Access for whom
d	rwX	rwX	rwX	← A directory with all attributes on
-	---	---	---	← A file with all attributes off

For each kind of user, each permission can be either on or off. If it's on, the corresponding letter (**r**, **w**, or **x**) appears; if it's off, you see a hyphen (-). So, for example, if the owner's permissions are **rwX**, it means she can read, write, and execute the item; if they're **r--**, she can read only.

If everybody—user, group, and others—had read, write, and execute permissions for a file, its permissions would look like this:

-rwxrwxrwx

Here are a few other combinations to make the system clear:

- Owner can read, write, and execute; group and others have no permission:

-rwx-----

- Owner can read and write; group and others can read:

-rw-r--r--

- Everyone can read and execute, but only the user can write:

-rwxr-xr-x

- Owner can read and write; group can read only; others have no permission:

-rw-r-----

Owner and Group

After the file's permissions and a number (the number of links to the item—a concept that's beyond the scope of this book) are two names.

The first of these is the file's owner (user) and the second is its group. For example in this item:

```
drwxr--r-- 15 jk admin 510 Aug 27 15:02 fruits
```

the owner is `jk` and the group is `admin`. (In some cases, owner, group, or both may be shown as numbers, such as `501`, rather than names.)

Permissions and You

When you create a file (whether by saving, copying, moving, downloading, or whatever), you become that file's owner (user).

In addition, by default, all users have read and write permission (and, for directories, execute permission) for everything in their home folders, and can read and execute shared items (such as things in the `/Applications` folder). However, users can't read or write files, or view the contents of directories, owned by other users.

Your default group (and thus, the default group of files in your home folder and new items you create anywhere) depends on a few factors, the most significant of which is what sort of user account you have (as specified in the Accounts pane of System Preferences). If you're an administrator, your default group is normally `admin`; otherwise, it's normally `staff`.

CHANGE AN ITEM'S PERMISSIONS

If you want to change an item's permissions, you use the `chmod` command (for "change mode," *mode* being a Unix way of describing an item's permissions). You can use `chmod` in a number of different ways, but what I describe here is the easiest one to understand—it's what you may sometimes hear described as `chmod`'s *symbolic* mode.

To change permissions with `chmod`, you simply indicate one or more of user, group, and others (using the abbreviations `u`, `g`, and `o` respectively), then `+` or `-` (to add or remove permissions), and one or more of `r`, `w`, and `x` (for read, write, and execute), followed by the file or directory. For example, to grant group write access to the file `file1`, you might enter this:

```
chmod g+w file1
```

To remove others' execute permission, enter this:

```
chmod o-x file1
```

You can affect multiple users at once—for example, add read access for user, group, and other in one stroke with this:

```
chmod ugo+r file1
```

You can also affect multiple permissions at once—for example, subtract read, write, and execute permission for the group and others with the following:

```
chmod go-rwx file1
```

Note: Ordinarily, you can change an item's permissions only if you are the owner or are in the item's group, and if you already have (in either capacity) write permission. In all other cases, you must use `sudo` (described ahead) before the `chmod` command.

In order to make more complex changes in one go (say, adding write permission for the user while removing execute permission for others), you must use `chmod`'s *absolute* mode, which I don't cover here—you can read all about it in *Take Control of Permissions in Leopard*.

CHANGE AN ITEM'S OWNER OR GROUP

To change an item's owner, group, or both, use the `chown` (change owner) command. It takes one or two arguments—the new owner and/or the new group, separated by a colon (:)—followed by the item you want to change. For example, to change the owner of the file `file1` to `bob` (without changing the group), enter:

```
chown bob file1
```

To change the owner of `file1` to `bob` and the group to `accounting`, enter:

```
chown bob:accounting file1
```

To change *only* the group, but not the owner, simply leave out the owner but include the colon before the group:

```
chown :accounting file1
```

However... What I just said is hypothetical, because as an ordinary user you can't change an item's owner—that would mean changing it either *to* or *from* an account to which you don't have access! Similarly, you can change an item's group only if you're a member of both the old group *and* the new group. So for all practical purposes, the `chown` command must always be performed using `sudo`, described next.

PERFORM ACTIONS AS THE ROOT USER

As a security measure, Mac OS X (like all Unix and Unix-like operating systems) prevents users from viewing or altering files that don't belong to them, including those that make up the operating system itself. However, in certain situations you may have a legitimate need to alter a file or folder of which you're not the owner—or run a command for which your user and group don't have execute permission.

Every Mac has a special, hidden account called `root`, which is a user with virtually unlimited power to change anything on the computer. The root account is disabled by default, and that's for the best. However, any administrator can *temporarily* assume the capabilities and authority of the root user, even without the root account as such having been activated.

The way you do this is to use the `sudo` (“superuser do”) command.

Sue Due: *Because the “do” in `sudo` is the actual verb do, the preferred pronunciation of the term rhymes with “voodoo.” But lots of people pronounce it to rhyme with “judo,” which is also logical—and acceptable to everyone except the nitpickiest geeks.*

For Administrators Only

Before I go any further, I must make this crystal clear: only users with administrator privileges can use `sudo`. If your Mac has just one user account, it's automatically an administrator account. However, as you create additional accounts, they only gain administrator privileges if you check the Allow User to Administer This Computer box in the Accounts pane of System Preferences.

Most Mac experts recommend using a *non-administrator* account for ordinary, day-to-day computing, logging in as an administrator only

when necessary. That’s good advice, but if you follow it, you’ll have to do one of two things before you can make use of the `sudo` command:

- Log in as an administrator first, and then run Terminal, *or*
- In your shell session in Terminal, switch to an administrator’s account using the `su` (switch user) command, like so:

```
su user-name
```

(Replace `user-name` with the short user name of an administrator, and enter that account’s password when prompted.)

Blind typing: As you type the administrator account’s password, no text appears—not even bullet or asterisk characters. That’s normal.

Using sudo

Once you’re logged in as an administrator, to perform any command as the root user, preface it with `sudo`:

```
sudo command
```

The `sudo` command prompts you to enter the administrator account password; do so now.

Blind typing: As you type your password, no text appears—not even bullet or asterisk characters. That’s normal.

The shell then performs whatever command you just entered as though you’d entered it as the root user, which ordinarily means it’s guaranteed to work as long as you entered it correctly.

If you perform a command and get a “permission denied” error, try it again with `sudo` in front of it, and in all probability it will work the second time. For example, if you try to change a file’s owner like so:

```
chown bob file1
```

and you get this message:

```
chown: file1: Operation not permitted
```

try this instead:

```
sudo chown bob file1
```

Tip: Now that you understand how `sudo` works, you may enjoy this highly geeky comic: <http://xkcd.com/149/>.

Notes and Precautions

Before you start using `sudo`, you should be aware of a few things:

- **The 5-minute rule:** Once you use `sudo` and enter your password, you can enter additional `sudo` commands, *without* being prompted for a password, for 5 minutes. The timer resets every time you use `sudo`.
- **Great power = great responsibility:** You can do almost anything with `sudo`, and that includes damaging Mac OS X beyond repair. So use `sudo` only when necessary, and only when you know what you're doing.
- **Stay for a while:** If you know you must enter a large number of commands with root privileges, you can avoid having to enter `sudo` every time by switching to the root user's shell account. (Again, surprisingly, this does *not* require that the root account actually be enabled on your Mac!) To switch to the root user's shell, enter `sudo -s` and supply your password if requested. Your prompt changes from a `$` to a `#` to signify that all commands you enter are now performed as the root user.

Be extra careful! If `sudo` alone is dangerous, `sudo -s` is asking for trouble. It's a convenience feature I personally use on rare occasions, and it can be handy in a few situations in which `sudo` alone won't do the trick. But use this with the utmost caution, and be sure to enter `exit` to log out of the root user's shell as soon as possible.

Command-Line Recipes

You've learned about the raw ingredients and the tools you need to put them together. Now it's time for some tasty recipes that put your knowledge to good use! In this section, I present a selection of short, easy-to-use commands and customizations you can perform in the bash shell. Many use features, functions, and programs I haven't mentioned elsewhere in this book, and although I describe essentially how they work, I don't go into detail about every new item included in the recipes. Just add these herbs and spices as directed, and enjoy the results!

CHANGE DEFAULTS

Most Mac OS X applications, from the Finder to Mail to iTunes, store their settings in specially formatted property list, or .plist, files. Surprisingly, applications often have hidden preferences that don't show up in their user interfaces—but if you put just the right thing in the preference file, you can change an application's behavior in interesting ways, or even turn on entirely new features.

One way to edit preference files is to open them in a text editor, or in the Property List Editor included with Apple's Xcode Tools (which you can find on your Mac OS X Install disc). But another, often easier way, is to use a command called `defaults` which can directly add, modify, or remove a preference from a .plist file. The recipes in this first set all use the `defaults` command.

Before using these commands to alter an application's defaults, quit the application if possible (obviously that's not an option with the Finder or the Dock).

Change Scrollbar Arrow Locations

By default, all scrollbars in Mac OS X have a pair of arrows (up and down) at the bottom. If you prefer a different arrangement, you can put the up arrow at the top and the down arrow at the bottom using the Appearance pane of System Preferences. (A corresponding change occurs with horizontal scrollbars.) But if you want both arrows at both ends, you need to make use of a hidden preference.

To put double arrows on both ends, enter this on a single line:

```
defaults write -g AppleScrollBarVariant -string DoubleBoth; killall Finder
```

Note: The semicolon enables you to include two (or more) separate commands on a single line. The second one, `killall Finder`, causes the Finder to quit, after which it relaunches automatically, letting the new defaults take effect.

To return your scrollbars to their original setting, enter this:

```
defaults write -g AppleScrollBarVariant -string DoubleMax; killall Finder
```

Force Incoming Mail Messages to Appear in Plain Text

As I wrote in *Take Control of Apple Mail in Leopard*, I prefer to read incoming messages in Mail in plain text format if possible. Not all messages include a plain text version, but for those that do, the following command ensures that this version appears by default:

```
defaults write com.apple.mail PreferPlainText -bool TRUE
```

If you're reading a message in plain text format and you want to switch it to a different (HTML or Rich Text) format, press Command-Option-] (you may need to press it twice to cycle to the format you want).

(To reverse this setting, repeat the command, changing `TRUE` to `FALSE`.)

Bring Dashboard Widgets onto the Desktop

This is an oldie but a goodie. If you dislike having to switch into Dashboard to see one or two commonly used widgets and would like to display them directly on your Desktop, enter this:

```
defaults write com.apple.dashboard devmode YES; killall Dock
```

Having done this, press F12 to display Dashboard. Then click on a widget and, while holding down the mouse button, press F12 again to return to the Desktop. Your widget will still be visible.

(To reverse this setting, repeat the command, changing `YES` to `NO`.)

Deactivate Dashboard

Or perhaps you don't like Dashboard at all. If you never use it, and would just as soon it never runs (even if you click its icon in the Dock), disable it entirely by entering this:

```
defaults write com.apple.dashboard mcx-disabled -boolean YES; killall Dock
```

(To reverse this setting, repeat the command, changing YES to NO.)

Expand Save Dialogs by Default

Ordinarily when you use an application's Save or Save As command, the Save dialog that appears gives you only a simple pop-up menu from which to select a location for a file; you have to click the  button to expand it so it shows your entire computer. To make all Save dialogs appear in their expanded state by default, enter this:

```
defaults write -g NSNavPanelExpandedStateForSaveMode -bool TRUE
```

(To reverse this setting, repeat the command, changing TRUE to FALSE.)

Show Hidden Files in the Finder

By default, the Finder keeps some files and folders hidden—those whose names begin with a period and many of the Unix files and directories at the root level of your disk. That's usually good, because it prevents you from changing things that could cause your computer to break, but if you want to see all your files and folders, enter this:

```
defaults write com.apple.finder AppleShowAllFiles TRUE; killall Finder
```

(To reverse this setting, repeat the command, changing TRUE to FALSE.)

Prevent Dock Icons from Bouncing

When an application wants to get your attention, its Dock icon usually bounces. If you find this distracting and want to turn off the bouncing, enter the following:

```
defaults write com.apple.dock no-bouncing -bool TRUE; killall Dock
```

(To reverse this setting, repeat the command, changing TRUE to FALSE.)

Change the Screenshot Format

When you take a screenshot in Mac OS X (using either the Grab utility or the Command-Shift-3 or Command-Shift-4 keyboard shortcuts), the resulting pictures are normally saved, on your Desktop, in PNG (Portable Network Graphics) format. If you prefer another format, such as JPEG, enter this:

```
defaults write com.apple.screencapture type JPEG; killall SystemUIServer
```

Use the same command, but substitute **TIFF**, **PNG**, or **PICT** for **JPEG** to use one of those formats.

Change Safari 4's Tabs

Shortly before this book was published, Apple released a public beta of Safari 4. One of the new features—loved by some, hated by others—is Tabs on Top, in which browser tabs appear in the window's title bar. If you prefer the old (tabs-on-a-Tab-Bar) way, enter this:

```
defaults write com.apple.safari DebugSafari4TabBarIsOnTop -bool FALSE
```

(To reverse this setting, repeat the command, changing **TRUE** to **FALSE**.)

Tip: For more hidden Safari 4 options, see this *Macworld* article: <http://www.macworld.com/article/139053/safari4prefs.html>.

PERFORM ADMINISTRATIVE ACTIONS

This group of recipes involves administrative tools—things you might need to do on a multi-user Mac, a server, or a remote Mac.

Use Software Update from the Command Line

If you want to run Software Update on your Mac without having to deal with intrusive windows, or if you want to run it on a remote Mac via SSH, enter the following command:

```
sudo softwareupdate -i -a
```

The **-i** and **-a** flags together mean “go ahead and install everything Software Update finds.”

List Users Who Logged In Recently

Is your Mac used by a number of different people? Do users sometimes log in remotely? If you'd like to know who's been logging in recently, you can get a lengthy list with this command:

```
last
```

This command lists the users who have logged into this machine, the IP address or terminal from which they logged in, and important system events such as shutdowns and restarts.

Find Interesting Stuff in Log Files

Many Unix and Mac OS X applications and background processes constantly spit out log files detailing what they've been up to and, perhaps most important, any errors they've encountered. Most system processes store their log files in `/var/log`, and although you can open them in any text editor, they tend to be so long and inscrutable as to make the exercise more bother than it's worth. Luckily, you can use the `grep` command to sift through log files looking for specific strings.

For example, to look through the main system log for every instance of the word `error` (a sure sign of an interesting entry), enter this:

```
grep error /var/log/system.log
```

Or, to look for all entries involving Time Machine (whose background process is called `backupd`), enter this:

```
grep backupd /var/log/system.log
```

If you'd rather have a paged display instead of dumping the output directly onto the command line, you can pipe it through `less`, like so:

```
grep backupd /var/log/system.log | less
```

MODIFY FILES

A number of common recipes involve modifying files in some way. Here's a selection.

Change the Extension on All Files in a Folder

The `mv` command, as I discussed in [Move or Rename a File or Directory](#), has an unfortunate shortcoming in that it can't rename

a batch of files all at once using wildcards. But never fear, you can still get the job done. Begin by creating the following shell script, using the instructions in [Create Your Own Shell Script](#):

```
#!/bin/bash
for f in $3/*. $1; do
    base=`basename $f . $1`
    mv $f $3/$base.$2
done
```

Note: This script makes use of the backtick (`) character, which is called a grave accent when placed over a vowel. It's on the same key as the tilde (~), and should not be confused with the apostrophe (') or the backslash (\).

Make sure it's located somewhere in your PATH, and that it's executable (see [Understand Permission Basics](#), earlier). I call this script `br.sh`, for “batch rename.”

To run this script, enter the script name followed by the old extension, the new extension, and the directory in which to make the change—in that order. For example, to change all the files in `~/Documents` with the extension `.JPG` to end in `.jpeg`, enter this:

```
br.sh JPG jpeg ~/Documents
```

Decompress Files

If you decide to download Unix software (or source code to compile yourself), it may be packaged in any of several unfamiliar archive formats. A file ending in `.tar` is a “tape archive” (a way of bundling files together without necessarily compressing them); the extensions `.gz` and `.bz2` refer to different compression mechanisms, and you may see a combination of these (such as `archive.tar.gz`). To decompress and/or unpack these, use one of the following commands, based on the extension(s) the file has:

```
tar -xf archive.tar
tar -xzf archive.tar.gz
tar -xjf archive.tar.bz2
```

As you can see, each compression format requires a different flag—use `-z` for `.gz` (or `.tgz`) and `-j` for `.bz2` (or `.bz`).

Convert Documents to Other Formats

Mac OS X includes a marvelous command-line tool called `textutil`, which can convert text documents between any of numerous common formats. This can be particularly useful in cases where you don't have Microsoft Word, or aren't satisfied with the way it saves files in other formats. Here are a couple of examples.

Convert a File from RTF to Word (.doc)

To convert the file `file1.rtf` (RTF format) to Word format (.doc) and save it as `file2.doc`, enter this:

```
textutil -convert doc file1.rtf -output file2.doc
```

Convert a File from Word (.doc) to HTML

To convert the file `file1.doc` (Word format) into HTML format and save it as `file1.html`, enter the following:

```
textutil -convert html file1.doc
```

(When no filename is specified, `textutil` uses the same filename with an extension corresponding to the format you requested.)

Note: The `textutil` program supports other formats too; just substitute the format of your choice for `doc` or `html` in the examples above. Among the most useful options are `txt` (plain text), `html` (HTML), `rtfd` (RTF with attachments), `docx` (Open Office XML), and `webarchive` (Web archives, like Safari uses).

WORK WITH INFORMATION ON THE WEB

The command-line environment includes a handy program called `curl` that can connect to Web, FTP, and other servers and upload or download information. Here are a few examples of how to use it.

Download a File

If you know the exact URL of a remote file on a Web, FTP, SFTP, or FTPS server, you can fetch it with this simple command (fill in the URL as appropriate):

```
curl -s -S -O URL
```

The file is downloaded to your current directory.

Save a Local Copy of a Web Page

When you browse the Web in Safari, you can save the source of any Web page. You can do the same right on the command line, without ever opening a browser, using this command:

```
curl URL > filename.html
```

For example, to save the source of the TidBITS home page to a file named `tidbits.html`, you can enter this:

```
curl http://db.tidbits.com/ > tidbits.html
```

Note that this command doesn't download graphics, style sheets, or other files linked from the Web page, so the page may not look entirely correct if you open it in a browser.

Note: This script uses the redirect (`>`) operator to send the output of a program to a file rather than displaying it on screen. That makes it a close relative of the pipe (`|`), which redirects a program's output to another program.

Put the Source of a Web Page on the Clipboard

What if you don't want to save a Web page to a file, but instead want to put it on your Clipboard so that you can paste it into another application? Enter the following:

```
curl URL | pbcopy
```

For example:

```
curl http://db.tidbits.com/ | pbcopy
```

MANAGE NETWORK ACTIVITIES

The following several recipes involve ways of getting information about local and remote networks, and the computers running on them.

Get Your Mac's Public IP Address

If your Mac is connected to the Internet using a gateway, modem, or router, its private IP address (the one you can see in the Network pane of System Preferences) probably isn't the same as the public address that other computers see.

To find out your Mac’s current public IP address, enter this:

```
curl -s http://www.showmyip.com/simple/; echo
```

The `echo` command is there only to make sure there’s a blank line after your IP address when it’s reported, to improve readability.

Get a List of Nearby Wi-Fi Networks

The AirPort menu in your menu bar lists nearby Wi-Fi networks. But if you’ve turned off that menu, or simply want to get at that information from the command line, enter this:

```
/System/Library/PrivateFrameworks/Apple80211.framework/Versions/A/Resources/airport -s
```

It displays nearby networks’ names (SSIDs), MAC addresses, encryption types, and other useful information. To learn what else this tool can do, substitute the `-h` (help) flag for `-s`. Yes, the full path is needed for executing this command: if you think you’ll use it often, you can set up an alias for it (see [Customize Your Profile](#), earlier).

Find Out Which Applications Have Open TCP/IP Network Connections

You take it for granted that your Web browser and email program are connected to the Internet. But what other applications or background processes have network connections? Is anything covertly “phoning home?” To see a list of processes you own that are accessing the Internet, enter this:

```
lsof -i
```

To see a list of all processes accessing the Internet, enter:

```
sudo lsof -i
```

Either way, you get a list of the processes on your Mac that currently have Internet connections, along with the domain names or IP addresses to which they’re connected, the ports they’re using, and other useful tidbits.

Determine if Another Computer Is Running

Did your server crash? Is another Mac on your network turned on and awake? The easy way to find out if another computer is on, awake, and connected to the network is to use the `ping` command.

Enter this (substituting the remote computer's domain name or IP address):

```
ping address
```

If you see a sequence of lines that look something like this, the computer is responding:

```
64 bytes from 216.168.61.41: icmp_seq=0 ttl=49 time=799.227 ms
```

Press Control-C to stop pinging. If more than 30 seconds go by without any such line appearing, either the computer is offline, it is configured not to respond to pings, or there's a network problem.

Get Information about an Internet Server

What's the IP address of that Web server you're connecting to? An easy way to find out is to use the host command:

```
host domain-name
```

This command returns the IP address(es) for that domain name. It also indicates if the domain name is an alias to another computer, and lists any mail exchange servers associated with that domain. For example, enter `host www.tidbits.com` to learn the IP address of the TidBITS Web server, the fact that `www.tidbits.com` is actually an alias (pointer) to a computer called `tidbits.com`, along with the domain names and IP addresses of the `tidbits.com` mail exchange servers.

Get Information about a Domain Name

If you need to find out what person or organization owns a domain name, enter the following:

```
whois domain-name
```

For example, if you enter `whois tidbits.com`, you'll learn which registrar handles the domain, the addresses of its name servers, and contact information for the domain's owner.

WORK WITH REMOTE MACS

These two recipes help you work with Macs you're accessing remotely.

Use Secure Screen Sharing via SSH

Mac OS X Leopard's built-in Screen Sharing feature gives you an incredibly easy way to see, and control, another Mac's screen. If you're sharing the screen of another Mac running Leopard (or newer), the network connection can be encrypted—choose Screen Sharing > Preferences and select Encrypt All Network Data (More Secure) to encrypt the all data flowing between the two computers. But if you're using Screen Sharing with a Mac running an older version of Mac OS X, or with any computer using VNC, someone could in theory eavesdrop on your screen-sharing session. To prevent this, you can create an SSH tunnel to encrypt the connection. First, put the following in your `.bash_profile` file (see [Customize Your Profile](#)) and then start a new shell session:

```
alias stss="(sleep 15; open vnc://127.0.0.1:5901) & ssh -C -4 -L 5901:127.0.0.1:5900"
```

Then, to start a session, enter the following, substituting your own user name and the domain name or IP address of the remote computer:

```
stss user-name@domain.com
```

As soon as you see the prompt to enter your password for the remote computer on the command line, do so. Then, a moment later, another authentication dialog should appear. Once again, enter your credentials (your short user name and password) for the remote computer. Once you've done this, a secure screen sharing session begins.

If the remote computer takes a while to respond, you may not have time to enter your password on the command line before Screen Sharing launches; if this happens, you'll get an error message. The easy way to fix this problem is to enter the `alias` command again, changing the number `15` after the command `sleep` to a higher number.

Restart a Remote Mac

If you need to reboot the Mac you're sitting in front of, you can simply choose  > Restart. But what if you're logged in to another Mac via SSH? To restart it, just enter this:

```
reboot
```

Needless to say, you should use this with caution—anyone else who happens to be using the computer at the time might be unhappy!

TROUBLESHOOT AND REPAIR PROBLEMS

These next few recipes can help you solve problems that keep your Mac from working correctly.

Delete Stubborn Items from the Trash

Occasionally, you may find that no matter what you do, you can't empty your Trash. Maybe you get an inscrutable error message, or maybe it just doesn't work. If this happens, try the following (taking all the necessary precautions associated with `sudo`, of course):

```
sudo rm -ri ~/.Trash/*
```

The `rm` command prompts you for confirmation to remove each item.

Warning! *Do be certain to type these commands exactly right; using `sudo rm` can do some serious damage if you're not careful!*

If that doesn't work, try each of these until the Trash is empty:

```
sudo rm -ri /.Trashes/*  
sudo rm -ri /Volumes/*/.Trashes/*
```

Figure Out Why You Can't Unmount a Volume

Have you ever tried to eject a CD, disk image, or network volume, only to see an error message saying the volume is in use? If so, the maddening part can be figuring out *which* process is using it so you can quit that process. So enter the following, substituting for `VolumeName` the name of the volume you can't unmount:

```
lsof | grep /Volumes/VolumeName
```

This command shows you any processes you own that are currently using this volume; armed with this information, you can quit the program (using the `kill` command if necessary, as described in [Stop a Program](#)). One frequent offender: the `bash` shell itself! If that's the case, you'll see something like this:

```
bash      14384   jk  cwd      DIR      45,8      330      2 /Volumes/Data
```

If you've navigated to a directory on this volume in your shell, Mac OS X considers it "in use." The solution in this case is to exit the shell, or simply `cd` to another directory.

If this command doesn't tell you what you need to know, repeat it, preceded by `sudo`.

Reset the Launch Services Database

Mac OS X's Launch Services database keeps track of which programs are used to open which files, among other things. If you find that the wrong application opens when you double-click files, or that icons don't match up with the correct items, you may need to reset your Launch Services database. Do it like this (enter the command as a single, long line—no space between `LaunchServices.framework` and `framework`):

```
/System/Library/Frameworks/CoreServices.framework/Frameworks/LaunchServices.  
framework/Support/lsregister -kill -r -domain local -domain system -domain user
```

Because this resets a lot of default mappings, your Mac may think you're launching applications for the first time and ask if it's okay. Agree to the alerts and you should be in good shape.

Fix Disk Problems without a Startup CD/DVD

If your startup disk has problems, the usual remedy is to start up from a Mac OS X Install DVD (or another startup volume) and run Disk Utility. If you don't have another startup volume handy, try this recipe.

First, power on (or restart) your Mac while holding down Command-S to enter single-user mode. You'll see a text display much like the inside of a Terminal window. Enter the following two commands, pressing Return in between:

```
mount -o remount,rw /  
fsck -y /
```

The `fsck` utility (“file system check,” which is like a command-line version of Disk Utility) checks most of your disk for errors, and repairs those it can. To restart your Mac normally afterward, enter `reboot`.

GET HELP IN STYLE

These two recipes let you read `man` pages in a friendlier environment than Terminal, without installing any extra software.

Read man Pages in Preview

The `man` command can save manual pages as beautifully formatted PostScript files, which Preview can then read. Because it’s a multi-step process, you need a shell function (like a shell script, but contained directly in your `.bash_profile` file) to help you do this. So, following the instructions in [Customize Your Profile](#), put the following lines in your `.bash_profile`:

```
psman()
{
man -t "${1}" | open -f -a /Applications/Preview.app/
}
```

Having done that, to view a man page in Preview, enter the following, substituting the name of whatever command you want to read about:

```
psman command
```

Et voilà! After a few seconds, a spiffy manual page opens in Preview.

Read man Pages in BBEdit

Or perhaps you’re more of a plain text, monospaced font kind of person. If you always have BBEdit open anyway, you can use it to open all your man pages instead of the built-in man program. To make this happen, add the following line to your `.bash_profile` (see [Customize Your Profile](#)), and then start a new shell session:

```
export MANPAGER="col -b | bbedit --clean --view-top"
```

Thereafter, entering `man` (followed by the command of your choice, such as `man ls`) displays the manual page in BBEdit.

DO OTHER RANDOM TRICKS

Finally, we have a bunch of interesting recipes that didn't fit neatly in any of the other categories. Enjoy!

Take a Screenshot

You can take a screenshot by pressing Command-Shift-3; the image is named Picture X by default and stored on your Desktop. But what if you want to take a screenshot and give it a different name, or store it somewhere else? Or—this is pretty cool—take a screenshot of a remote Mac you're logged in to via SSH? You can do it with this command (substituting the name and location of the saved screenshot to taste):

```
screencapture ~/myscreen.png
```

Compact a Disk Image

Of the numerous disk image formats Disk Utility can create, two of them—sparse disk images and sparse bundle disk images—have the interesting characteristic that they can expand as needed (up to a preset limit) to accommodate more files. (See my *TidBITS* article “Discovering Sparse Bundle Disk Images,” <http://db.tidbits.com/article/9673>.) The only problem is, they don't automatically shrink again when you delete files! To compact a sparse or sparse bundle image so that it takes up only the space it needs, enter the following, substituting the image's name and location as appropriate:

```
hdiutil compact image.dmg
```

Use Text-to-Speech from the Command Line

This is mostly just for fun. To have your Mac speak text using the text-to-speech voice currently selected in the Text to Speech view of the Speech pane of System Preferences, enter the following:

```
say "Hello there"
```

Note that this even works remotely, assuming you're logged in to a Mac on the other end. Use your power responsibly!

As a more practical example, you can make a quick-and-dirty count-down timer using a command like the following, substituting for 60 the number of seconds to wait before the Mac speaks the text you enter:

```
sleep 60; say "One minute has elapsed"
```

Prevent a Laptop from Waking up When Jostled During Travel

Mac laptops are designed to go to sleep automatically when you close the lid and wake up automatically when you open the lid. But if your computer happens to be bumped just the right way while it's in its case, the lid can open just enough to wake up the computer, potentially causing it to overheat, or causing your battery to run down, while it should be asleep. To prevent the computer from waking up automatically when the lid opens, enter this:

```
sudo pmset -a lidwake 0
```

Thereafter, wake your Mac by pressing a key after you open the lid.

(To reverse this setting, repeat the command, replacing the `0` with a `1`.)

Use the Power Button to Put Your Mac to Sleep

When you press and release the power button on your computer, what normally happens is that a dialog appears with Restart, Sleep, Cancel, and Shut Down buttons. If you prefer to put your computer directly to sleep with a single (short) press of the power button, enter this:

```
sudo pmset powerbutton 1
```

(To reverse this setting, repeat the command, replacing the `1` with a `0`.)

Find a File by Content

Earlier, in [Find a File](#), I showed how to use the `find` command to find a file by name. You can also use the command to find a file based on its content, but an even easier way is to use the `grep` command. Enter the following, replacing `your text` with what you want to find:

```
grep -R "your text" .
```

A couple of notes about this command:

- As written, it searches from the current directory downward. To search your entire disk (enormously time-consuming), replace the period (`.`) with a slash (`/`). Or, replace the period with whatever directory you want to use as the starting point for the search.
- This search finds partial matches without using wildcards; the string `bar` matches “baroque” and “lumbar”.

List More Directory Information

You should be thoroughly familiar with the `ls` (“list”) command, introduced in [See What’s Here](#). Among the flags that modify its behavior, I’ve described elsewhere in this book `-l` (long format) and `-h` (human-readable). But if you want `ls` to truly show you everything, you need to add a few more flags.

To make the command easier to use, add an alias to your `.bash_profile` (see [Create Aliases](#)) like this:

```
alias lll="ls -lah@e"
```

The flag `-a` lists all files, including hidden ones (those that begin with a period). The `-@` flag lists extended attributes, and the `-e` flag lists all access control lists (ACLs). (And yes, I agree that the meanings of `-@` and `-e` seem backwards at first glance!)

About This Book

Thank you for purchasing this Take Control book. We hope you find it both useful and enjoyable to read. We welcome your comments at tc-comments@tidbits.com. Keep reading in this section to learn more about the author, the Take Control series, and the publisher.

ABOUT THE AUTHOR

Joe Kissell is Senior Editor of *TidBITS*, a Web site and email newsletter about the Macintosh and the Internet, and the author of numerous print and electronic books about Macintosh software, including *Take Control of Mac OS X Backups* and *Take Control of Upgrading to Leopard*. He's also a Senior Contributor to *Macworld*. Joe has worked in the Mac software industry since the early 1990s, including positions managing software development for Nisus Software and Kensington Technology Group.



In his increasingly imaginary spare time, Joe likes to travel, cook, and practice t'ai chi. He lives in Paris with his wife, Morgen Jahnke. To contact Joe about this book, send him email at jwk@me.com and be sure to include the words *Take Control of the Mac Command Line with Terminal* in the subject of your message.

AUTHOR'S ACKNOWLEDGMENTS

Thanks to Geoff Duncan for an outstanding editing job, to the other Take Control authors and editors, and to all the members of the TidBITS Irregulars list who offered input and suggestions. Special thanks to the following people for suggesting command-line recipes: Marshall Clow, Keith Dawson, Geoff Duncan, Chuck Goolsbee, John Gotow, Kevin van Haaren, Andrew Laurence, Peter N Lewis, Chris Pepper, Larry Rosenstein, and Nigel Stanger.

SHAMELESS PLUG

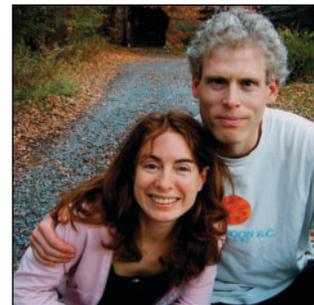
Although I write about computers as my day job, I have a great many other interests, which I write about on several Web sites, including [Interesting Thing of the Day](#) and my personal blog. You can find links to all my sites, a complete list of my publications, and more personal details about me at [JoeKissell.com](#).

ABOUT THE PUBLISHER

Publishers Adam and Tonya Engst have been creating Macintosh-related content since they started the online newsletter *TidBITS*, in 1990. In *TidBITS*, you can find the latest Macintosh news, plus read reviews, opinions, and more (<http://www.tidbits.com/>).

Adam and Tonya are known in the Mac world as writers, editors, and speakers. They are also parents to Tristan, who thinks ebooks about clipper ships and castles would be cool.

TidBITS
Mac news for the rest of us



PRODUCTION CREDITS

Take Control logo: Jeff Tolbert

Cover: Jon Hersh

Editor: Geoff Duncan

Editor in Chief: Tonya Engst

Publisher and automation master: Adam Engst

Link checking: Julie Kulik

Thanks to many TidBITS Irregulars and Control Freaks for technical edits and advice, and thanks to Amelia Sauter who lent us a newbie perspective. Chris Pepper gets the special most-comments per-line-of-manuscript award.

Copyright and Fine Print

Take Control of the Mac Command Line with Terminal

ISBN: 978-1-933671-55-0

Copyright © 2009, Joe Kissell. All rights reserved.

TidBITS Publishing Inc.

50 Hickory Road

Ithaca, NY 14850 USA

<http://www.takecontrolbooks.com/>

Take Control electronic books help readers regain a measure of control in an oftentimes out-of-control universe. Take Control ebooks also streamline the publication process so that information about quickly changing technical topics can be published while it's still relevant and accurate.

This electronic book doesn't use copy protection because copy protection makes life harder for everyone. So we ask a favor of our readers. If you want to share your copy of this ebook with a friend, please do so as you would a physical book, meaning that if your friend uses it regularly, he or she should buy a copy. Your support makes it possible for future Take Control ebooks to hit the Internet long before you'd find the same information in a printed book. Plus, if you buy the ebook, you're entitled to any free updates that become available.

Although the author and TidBITS Publishing Inc. have made a reasonable effort to ensure the accuracy of the information herein, they assume no responsibility for errors or omissions. The information in this ebook is distributed "As Is," without warranty of any kind. Neither TidBITS Publishing Inc. nor the author shall be liable to any person or entity for any special, indirect, incidental, or consequential damages, including without limitation lost revenues or lost profits, that may result (or that are alleged to result) from the use of these materials. In other words, use this information at your own risk.

Many of the designations used to distinguish products and services are claimed as trademarks or service marks. Any trademarks, service marks, product names, or named features that appear in this title are assumed to be the property of their respective owners. All product names and services are used in an editorial fashion only, with no intention of infringement of the trademark. No such use, or the use of any trade name, is meant to convey endorsement or other affiliation with this title.

This title is an independent publication and has not been authorized, sponsored, or otherwise approved by Apple Inc. Because of the nature of this title, it uses terms that are trademarks or registered trademarks of Apple Inc.; to view a complete list of the trademarks and the registered trademarks of Apple Inc., you can visit <http://www.apple.com/legal/trademark/appletmlist.html>.

Featured Titles

Now that you've seen this book, you know that Take Control books have an easy-to-read layout, clickable links if you read online, and real-world info that puts you in control. Click any book title below or [visit our Web catalog](#) to add to your Take Control collection!

Take Control of Apple Mail in Leopard (Joe Kissell): Go under the hood with the new (and old) features in Apple Mail 3. Joe gets you going and helps you get the most out of Mail. \$10

Take Control of Fonts in Leopard (Sharon Zardetto): Install, organize, and use fonts with ease in Leopard! \$15

Take Control of Mac OS X Backups (Joe Kissell): Set up a rock-solid backup strategy so that you can restore quickly and completely, no matter what catastrophe arises. \$15

Take Control of Maintaining Your Mac (Joe Kissell): Find a common-sense approach to avoiding problems and ensuring that your Mac runs at peak performance. \$10

Take Control of Permissions in Leopard (Brian Tanaka): Solve quirky problems, increase privacy, and share files better. \$10

Take Control of Running Windows on a Mac (Joe Kissell): With Intel-based Macs, it has become possible to run Windows software on a Mac, and with Joe's advice, it's easy! \$10

Take Control of Users & Accounts in Leopard (Kirk McElhearn): Find straightforward explanations of how to create, manage, and work with—and among—user accounts. \$10

Take Control of Your 802.11n AirPort Network (Glenn Fleishman): Make your AirPort network fly—get help with buying the best gear, set up, security, and more. \$15

Take Control of Your Wi-Fi Security (Engst & Fleishman): Learn how to keep intruders out of your wireless network and protect your sensitive communications! \$10