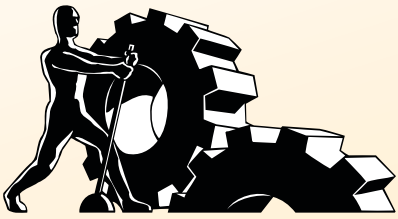


Check for Updates
Make sure you have the latest information!



TidBITS Publishing Inc.

Take Control of

v1.0

The Mac Command Line with Terminal

Joe Kissell

\$10

[Help](#)

[Catalog](#)

[Feedback](#)

[Order Print Copy](#)

Click here to buy the full 111-page "Take Control of the Mac Command Line with Terminal" for only \$10!

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, published in March 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 from 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.

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 have 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**).

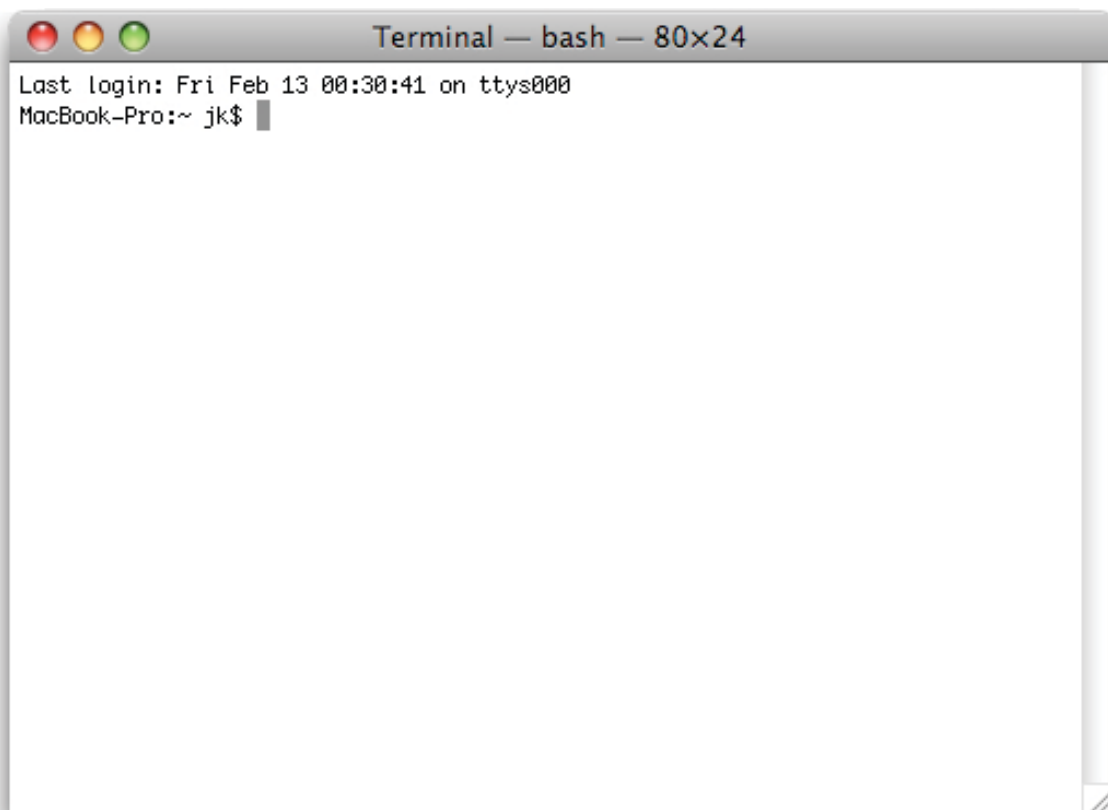


Figure 1: The Terminal window harks back to pre-graphical days.

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
```

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`).

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.

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.

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@
```

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.
```

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.

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.

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