



TAKE CONTROL OF  
**THE MAC**  
**COMMAND**  
**LINE** WITH  
**TERMINAL**

*by* **JOE KISSELL**  
**\$14.99**

**3rd**  
**EDITION**

# Table of Contents

<b>Read Me First.....</b>	<b>6</b>
Updates and More .....	6
Basics.....	7
What's New in Version 3.2.1 .....	8
What Was New in Version 3.2 .....	9
<b>Introduction .....</b>	<b>10</b>
<b>macOS Command Line Quick Start.....</b>	<b>13</b>
<b>Understand Basic Command-Line Concepts .....</b>	<b>15</b>
What's Unix?.....	15
What's a Command Line? .....	16
What's a Shell? .....	17
What's Terminal? .....	18
What Are Commands, Arguments, and Flags? .....	20
What Changed in Catalina? .....	24
What Changed in Big Sur? .....	31
What Changed in Monterey? .....	32
What Changed in Ventura? .....	34
How to Download Scripting Languages .....	34
<b>Get to Know (and Customize) Terminal .....</b>	<b>36</b>
Learn the Basics of Terminal .....	36
Modify the Window .....	38
Open Multiple Sessions .....	39
Change the Window's Attributes .....	40
Change Your Current Shell .....	43
Set a Default Shell.....	44
<b>Look Around .....</b>	<b>48</b>
Discover Where You Are .....	48
See What's Here .....	49
Repeat a Command .....	51
Cancel a Command.....	53

Move into Another Directory .....	53
Jump Home .....	56
Understand How Paths Work .....	57
Understand the macOS Directory Structure .....	59
Use Tab Completion .....	61
Find a File .....	63
View a Text File .....	65
Get Help .....	67
Clear the Screen .....	69
End a Shell Session .....	69
<b>Work with Files and Directories .....</b>	<b>71</b>
Create a File .....	71
Create a Directory .....	72
Copy a File or Directory .....	73
Move or Rename a File or Directory .....	75
Delete a File .....	77
Delete a Directory .....	78
Use Symbolic Links .....	79
<b>Work with Programs .....</b>	<b>81</b>
Learn Command-Line Program Basics .....	81
Run a Program or Script .....	84
Run a Program in the Background .....	89
See What Programs Are Running .....	90
Stop a Program .....	93
Edit a Text File .....	95
Create Your Own Shell Script .....	98
<b>Customize Your Defaults .....</b>	<b>102</b>
About Startup Files .....	102
Edit .zshrc or .bash_profile .....	103
Create Aliases .....	103
Modify Your PATH .....	105
Change Your Prompt .....	105
<b>Bring the Command Line into the Real World .....</b>	<b>108</b>
Get the Path of a File or Folder .....	108

Open the Current Directory in the Finder .....	110
Open a Hidden Directory without Using Terminal .....	110
Open the Current Folder in Terminal .....	111
Run a Shell Script from the Finder .....	113
Open a macOS App .....	113
Open a URL .....	113
Open a File in macOS .....	114
<b>Log In to Another Computer .....</b>	<b>115</b>
Start an SSH Session .....	115
Run Commands on Another Computer .....	117
End an SSH Session .....	118
Transfer Files with sftp or scp .....	118
<b>Work with Permissions .....</b>	<b>122</b>
Understand Permission Basics .....	122
Change an Item's Permissions .....	126
Change an Item's Owner or Group .....	128
Perform Actions as the Root User .....	129
<b>Learn Advanced Techniques .....</b>	<b>134</b>
Pipe and Redirect Data .....	134
Get a Grip on grep .....	138
Add Logic to Shell Scripts .....	141
<b>Install New Software .....</b>	<b>152</b>
Use Command Line Tools for Xcode .....	153
Install Unix Software from Scratch .....	155
Use a Package Manager .....	159
<b>Learn Command-Line Shortcuts .....</b>	<b>167</b>
Terminal Tips and Shortcuts .....	167
Zsh Tips and Shortcuts .....	169
<b>Command-Line Recipes .....</b>	<b>176</b>
Change Defaults .....	176
Perform Administrative Actions .....	182
Modify Files .....	188
Work with Information on the Web .....	193
Manage Network Activities .....	194

Do More with SSH .....	198
Work with Remote Macs .....	201
Troubleshoot and Repair Problems .....	202
Get Help in Style .....	211
Do Other Random Tricks .....	214
<b>About This Book .....</b>	<b>222</b>
Ebook Extras .....	222
About the Author and Publisher .....	223
<b>Also by Joe Kissell .....</b>	<b>225</b>
<b>Copyright and Fine Print .....</b>	<b>226</b>

# Read Me First

Welcome to *Take Control of the Mac Command Line with Terminal, Third Edition*, version 3.2.1, published in December 2022 by alt concepts. This book was written by Joe Kissell and edited by Geoff Duncan.

This book introduces you to the macOS 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. Most of the examples work with 10.6 Snow Leopard and later, although a few techniques require more recent versions of macOS.

If you want to share this ebook with a friend, we ask that you do so as you would with a physical book: “lend” it for a quick look, but ask your friend to buy a copy for careful reading or reference. Discounted [classroom and user group copies](#) are available.

Copyright © 2022, Joe Kissell. All rights reserved.

---

## Updates and More

---

You can access extras related to this ebook on the web (use the link in [Ebook Extras](#), near the end; it’s available only to purchasers). On the ebook’s Take Control Extras page, you can:

- Download any available new version of the ebook for free, or buy any subsequent edition at a discount.
- Access the book in both PDF and EPUB formats. (Learn about reading on mobile devices on our [Device Advice](#) page.)
- Read the ebook’s blog. You may find new tips or information, as well as a link to an author interview.

If you bought this ebook from the Take Control website, it has been added to your account, where you can download it in other formats and access any future updates.

---

## Basics

---

Please be aware of the following special considerations:

- **Spurious hyphens!** When you view this ebook in EPUB format, your ebook reader (such as Apple Books or Kindle) may insert extra hyphens in the longer lines of text that are provided as examples of what to type on the command line. You can mitigate this problem by viewing the text in a single column, with a smaller font, and in a landscape orientation. In some cases, you can turn off auto-hyphenation to remove these spurious hyphens. For example, if you are reading in Books in iOS, you can go to Settings > Books and turn off the Auto-hyphenation switch. However, with auto-hyphenation off, Books may now cut off some wider lines of command-line text.

If you are reading this ebook to absorb the material conceptually, this won't be a problem, but if you want to type the commands on your Mac, consider downloading the PDF of this ebook onto your Mac, in order to read it there. As a bonus, you can copy the command-line text out of the PDF and paste it on the command line. Read [Ebook Extras](#) for help with downloading the PDF.

- **Entering commands:** I frequently tell you to “enter” a command 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 afterward 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. As I mentioned just above, if you're reading this on a Mac, you can copy the command from the ebook 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 type it. Similarly, be sure to type all punctuation—such as hyphens, periods, 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. (In the EPUB version of this book, the font shown might not be monospaced. Also, be sure to *read the first item in this list*, to avoid entering unnecessary hyphens.)
- ▶ *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.

---

## Settings vs. Preferences

---

In macOS Ventura, Apple replaced System Preferences with System Settings, and in most apps (including Terminal), what was formerly a Preferences window is now a Settings window. In this book, I generally use a shorthand like “go to Terminal > Settings/Preferences” or “open System Preferences/System Settings” to reflect these two possibilities. When I list System Settings separately, that applies only to Ventura or later.

---

## What's New in Version 3.2.1

---

This small update makes the book current with macOS 13 Ventura:

- Added a brief [What Changed in Ventura?](#) topic



- Added tip about using 1Password 8 in [Set Up a Password-less SSH Login](#)
- Updated [Read man Pages in Preview](#) with a new technique that still works in Ventura
- Updated mentions of Preferences windows to reflect Ventura’s use of the term Settings

---

## What Was New in Version 3.2

---

Version 3.2 updated the book to cover macOS 12 Monterey and added a few other small but useful changes:

- Added the topic [What Changed in Monterey?](#) to cover the removal of PHP and Python from macOS and changes to proxy icon support
- Updated [Clear the Screen](#) with more accurate information
- Mentioned a new flag available starting in Monterey for the `rm` command; see the sidebar [Warning! The rm Command Has No Safety Net](#)
- Mentioned that you can run shortcuts from the Shortcuts app on the command line; see [Run a Script](#)
- Added the topic [Open a URL](#) to describe a new option for the `open` command starting in Monterey
- Added a link to an article about `grep`, `sed`, and `awk`; see [Get a Grip on grep](#)
- Adjusted the text in [Homebrew](#) to account for the new default installation location on M-series Macs
- Updated the recipe [Remove the Proxy Icon Hover Delay in Big Sur](#) to reflect the fact that it’s no longer needed in Monterey
- Added a recipe to [Use Old-Style Alerts in Big Sur and Later](#)
- Added a recipe to [Turn Off Auto Boot](#)

# 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 was simply 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 do my work 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 few decades, and I sometimes find myself faced with some mundane task, such as deleting a file that refuses to disappear from the Trash or changing an obscure system preference. After wasting time puzzling over how to accomplish my task—and perhaps doing some web searches—I discover that the Mac'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 app 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 30-odd 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, the Mac's beautiful 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, macOS 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 macOS 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 an app called Terminal, located in the Utilities folder inside your Applications folder. It's a simple app 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 Mac publications, you've undoubtedly seen tips from time to time that begin, "Open Terminal and type in the following...". Many Mac users find that sort of thing intimidating. What do I *click*? 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.

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 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. In particular, the last chapter 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 many other things, this book shows you:

- How to figure out what's preventing a disk from disconnecting (unmounting or ejecting)
- How to tell which apps are currently accessing the internet
- How to rename lots of files at once, even if you're not running Yosemite or later
- 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. That's true, but the command line is infinitely more flexible—and Terminal is free!

I should be clear, however, that this book won't turn you into a command-line expert. I would need thousands of pages to describe everything you can accomplish with the command line. Instead, my goal is to cover the basics and get you up to a moderate level of familiarity and competence. And, based on feedback from the first two editions of this book, I've expanded the scope of this revised third edition to include a number of topics that are a bit more advanced.

Most of my examples work with any version of macOS from 10.6 Snow Leopard on, although a few techniques require later versions; I point those out as we go along.

# macOS Command Line Quick Start

This book is mostly linear—the later sections tend to build on the earlier sections. For that reason, 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 chapter, [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](#).
- Become familiar with the most common tool for accessing the command line; see [Get to Know \(and Customize\) Terminal](#).
- Navigate using the command line; see [Look Around](#).

## ***Learn basic skills:***

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

## ***Go beyond the Terminal window:***

- Integrate the command line and the Mac’s graphical interface; see [Bring the Command Line into the Real World](#).
- Use the command line to control another Mac; see [Log In to Another Computer](#).

***Earn your propeller beanie:***

- Learn about users, groups, permissions, and the infamous [sudo](#) command; see [Work with Permissions](#).
- [Learn Advanced Techniques](#) such as piping and redirecting data, using the [grep](#) search tool, and adding logic to your shell scripts.
- Go beyond what's built into macOS by downloading third-party command-line programs; see [Install New Software](#).
- Find quicker and easier ways to perform common activities in Terminal; see [Learn Command-Line Shortcuts](#).

***Put your skills into practice:***

- Do cool (and practical) stuff on the command line; see [Command-Line Recipes](#).

# 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 chapter explains the ideas and terminology I use throughout the book, providing context for everything I discuss later.

---

## 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 or later running on an Intel-based Mac is a true Unix operating system. Earlier versions of Mac OS X, and any version running on PowerPC-based Macs, were technically Unix-like.

macOS 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, the developer of NeXTSTEP, was the company that Steve Jobs founded after leaving Apple in 1985.

When Apple bought NeXT in 1996, they 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 (later renamed OS X and now macOS): it's Unix underneath, but with lots of extra stuff that's not in other versions of Unix. If you took macOS and stripped off the graphical interface, the Cocoa application programming interfaces (APIs), and all the built-in apps such as Mail and Safari, you'd get the Unix core of macOS. 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 though 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.

---

## 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 macOS, it isn't quite correct to say that you're working in Darwin when you're using the macOS command line. In fact, the command line gives you a way to interact with all of macOS, only part of which is Darwin.



# Get to Know (and Customize) Terminal

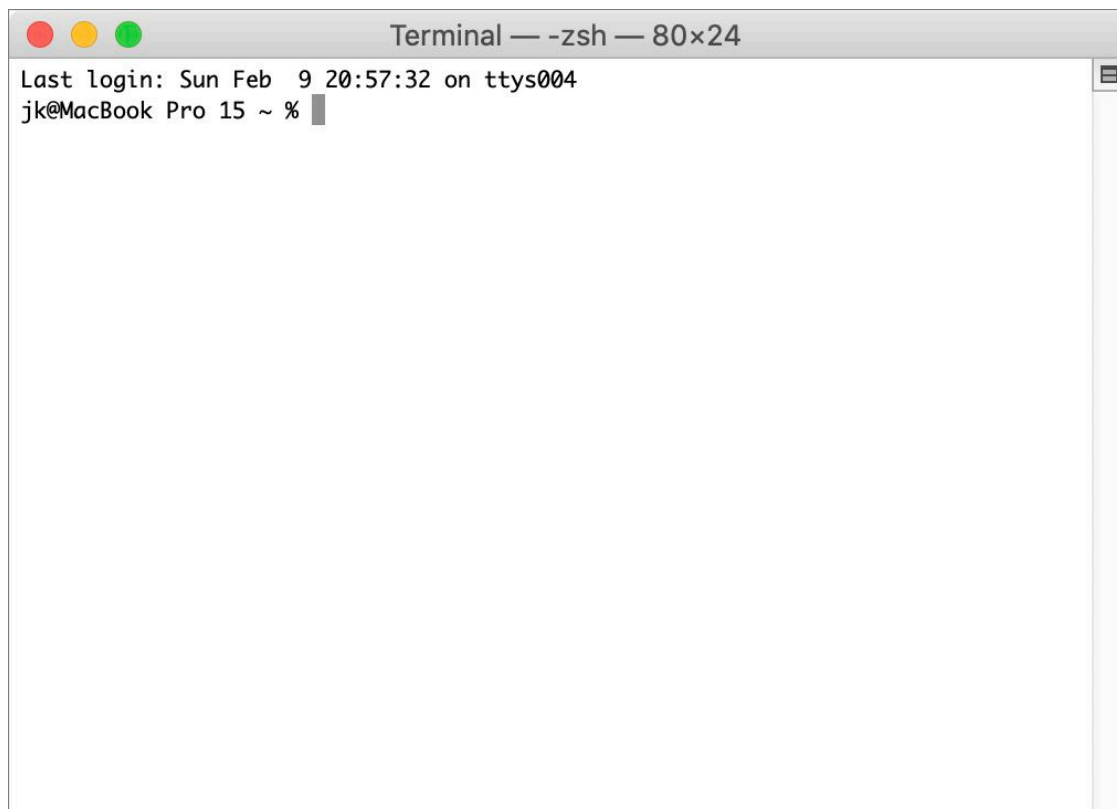
As I mentioned in [What's Terminal?](#), the app you're most likely to use for accessing the command line in macOS is Terminal. Since you'll be spending so much time in this app, a brief tour is in order. In addition, you may want to adjust a few settings, such as window size, color, and font, to whatever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

---

## Learn the Basics of Terminal

---

The moment has arrived. Find the Terminal app (inside the folder [/Applications/Utilities](#)), double-click it, and take a Zen moment to contemplate the emptiness (**Figure 2**). The exact appearance of the window will vary based on your version of macOS.



**Figure 2:** 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 `zsh` shell). As long as you're in this window, you can largely forget about your mouse or trackpad: with a couple of notable exceptions (see the sidebar [Using a Mouse in Terminal](#)), 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: Sun Feb  9 20:57:32 on ttys004
```

That last part, `on ttys004`, 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):

```
jk@MacBook Pro 15 ~ % █
```

If you're using `bash` rather than `zsh`, it may look more like this:

```
MacBook-Pro-15:~ jk$ █
```

Either way, 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 the *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, `jk@MacBook Pro 15` in the case of the `zsh` shell, is my short username followed by the `@` sign and the name of my Mac. Next is the tilde (`~`), which signifies that I'm currently in my home directory (which, for me, is `/Users/jk`). And finally, the `%` signifies that I'm logged in as an ordinary (non-root) user. (I say more about the `%` and the `bash` equivalent, `$`, in the sidebar [The %, \\$, and # Symbols on My Command Line](#), ahead.) If your short username is cindy and your computer's name, as shown in System Settings >

# Look Around

In this chapter, I help you find your way around your Mac 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
```

**Note:** As a reminder, to *enter* something on the command line, type it and press Return or Enter afterward.

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 directory structure, 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 in a deeply embedded directory, being able to tell exactly where you are can be a huge help.

**Note:** When I say “disk” in this book, I’m referring to any mass-storage device, not necessarily a mechanical hard disk. It could be an SSD or other flash storage.

---

## 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:

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

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 the more-helpful long format by adding the `-l` flag:

```
ls -l
```

This produces a result something like:

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

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. See the sidebar on the next page, [Making Output \(More\) Human-Readable](#), to find out how to turn that number into a nicer format. (In the case of a directory, the number shown by `ls -l` doesn't tell you the total size of the directory's *contents*, only the size of the information stored *about* the directory.

# 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 chapter covers the essentials of interacting with files and directories.

---

## Create a File

---

I want to begin by mentioning a curious command called `touch` that serves two interesting functions:

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

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`.

**Note:** 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! A few other potentially useful things to be aware of:

- You can create a new directory in some other location than your current one (for example, you could enter `mkdir ~/Documents/apples`).
- If you want to create a hierarchy of directories—for example, you want to create a directory called `oranges` inside `~/Documents/fruit/citrus/` and the `fruit` and `citrus` directories don't already exist—add the `-p` flag (for example: `mkdir -p ~/Documents/fruit/citrus/oranges`).

# Work with Programs

Every command that 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 chapter, I explain some of the different types of programs you may encounter and how to run them (and stop them).

I also show you how to edit files on the command line, and I talk about *shell scripts*, a special kind of program you can create 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 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 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.

**Note:** This procedure changes the password for your Mac's user account, which applies everywhere (not just on the command line).

Programs of this sort include `ssh`, which lets you [Log In to Another Computer](#), and `sftp`, 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).

**Note:** Starting in 10.11 El Capitan, whenever you use a command that prompts you for a password, such as `passwd` or `sudo`, a key  icon appears after the password prompt to remind you that whatever you type will not be shown on screen. It also means you're in Secure Input mode, so third-party apps (such as 1Password, Keyboard Maestro, and TextExpander) can't enter text on your behalf.



# Customize Your Defaults

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 defaults, which are stored in a special file your shell reads every time it runs. In this chapter, I explain how this process works and how you can use it to save typing, customize your prompt, and more.

---

## About Startup Files

---

A *startup file* (also known as an initialization file, or informally as 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, this file 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.

For complicated reasons, `zsh` and `bash` load different startup files under different circumstances—and in some cases, more than one such file loads when you begin a session. Rather than detail which files do what and when, I'll cut to the chase: for most people, the optimal startup file to use for your defaults under `zsh` is `~/.zshrc`, and for `bash`, the optimal file is `~/.bash_profile`. Those are the ones I cover in this book.

Fortunately, most of the common configurations are the same for both `zsh` and `bash` (I point out some minor differences ahead). So, if you're moving from `bash` to `zsh` and you already customized `.bash_profile` but don't yet have a `.zshrc` file, you can make a great first pass by duplicating the old file but with a new name:

```
cp ~/.bash_profile ~/.zshrc
```

**Note:** Customizations you make in `.zshrc` or `.bash_profile` 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 `.zshrc` or `.bash_profile`

---

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

```
nano ~/.zshrc (for zsh) or
```

```
nano ~/.bash_profile (for bash)
```

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 that you want (as described on the following pages). When you're finished editing `.bash_profile`, save it (Control-O) and close it (Control-X). Ordinarily, the changes take effect with the next shell session (window or tab) you open. To load the modified profile immediately, enter `source .zshrc` (for `zsh`) or `source .bash_profile` (for `bash`).

---

## Create Aliases

---

In the Finder, an alias is a small file that serves as a pointer to another file (for something comparable to Finder aliases on the command line, refer to [Use Symbolic Links](#)). 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 macOS, you can make an alias, so that entering `dir` runs the `ls` command. Or, maybe there's

# Bring the Command Line into the Real World

So far in this book I've largely ignored the Mac'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 chapter, 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.

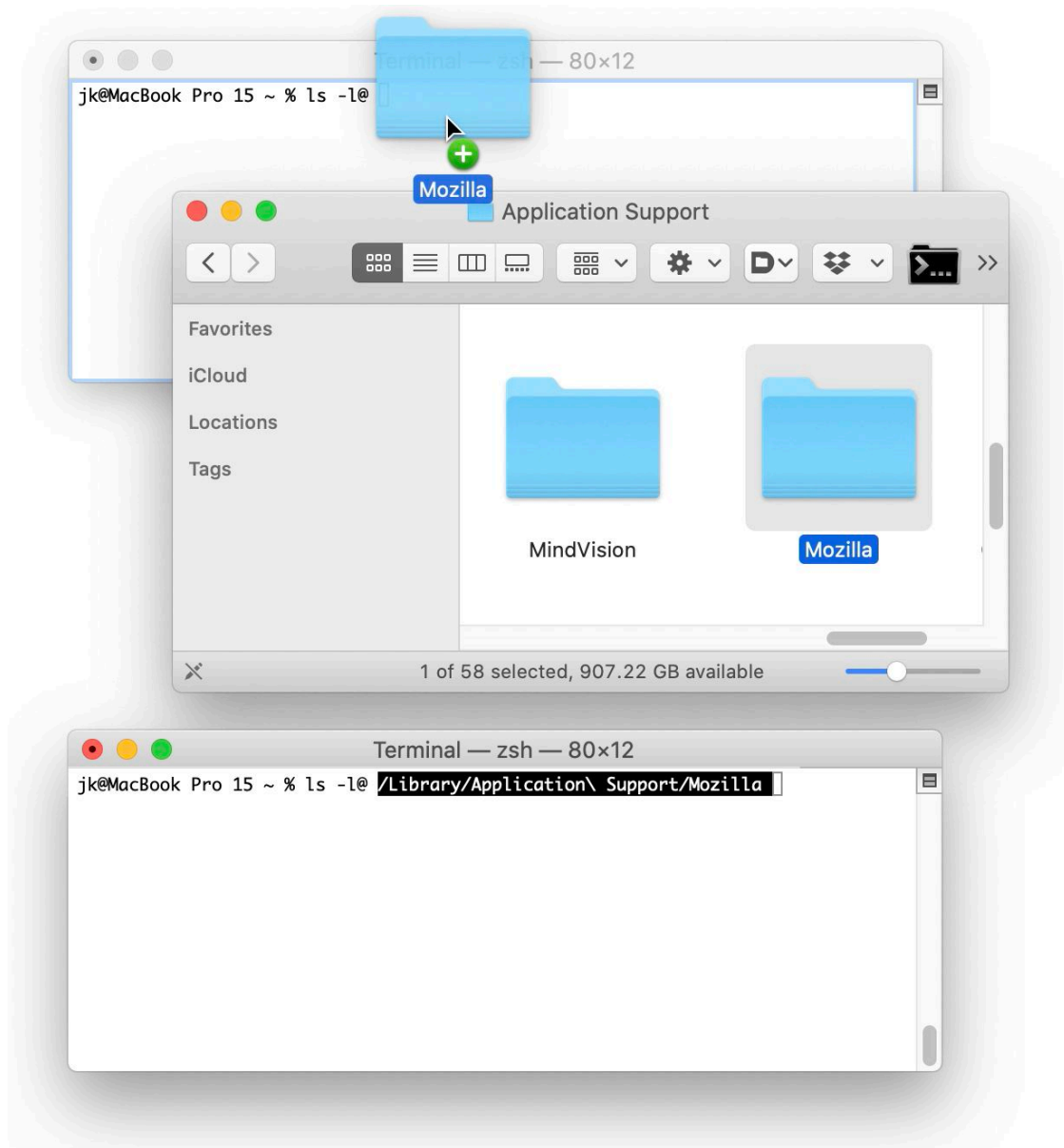
As soon as you release the mouse button, Terminal copies the path of the file or folder you dragged onto the command line. It even escapes spaces and single quotation marks with backslashes for you automatically! You can then press Return to run the command.

For example, suppose you want to use the `ls -l@` command to list the contents of a folder with their extended attributes (a type of *metadata*, or extra information about files and folders in addition to their actual

contents), which you can't see in the Finder—see the sidebar [What's with the + and @ Characters?](#). 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 7**.



**Figure 7:** Drag a file or folder into the Terminal window (top); when you release the mouse button, you get that item's full path (bottom).

# 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 chapter, 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, enable Remote Login in System Settings > General > Sharing or System Preferences > Sharing.)

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 username on the remote computer for *username*, and the remote computer’s IP address or domain name for *remote-address*:

`ssh username@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:64ce:eeb4:68ae%en0)' can't be established.
RSA key fingerprint is
d0:18:53:71:04:9d:c4:2e:5b:b1:f9:a0:7d:80:51:ef.
Are you sure you want to continue connecting (yes/no)?
```

After reading the sidebar “SSH Security Considerations,” just ahead, 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:64ce:eeb4:68ae%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.

**Note:** 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.

# Work with Permissions

Everything you do on your Mac, and especially on the command line, is governed by *permissions*—which user(s) can do which things with which items, under which circumstances. In this chapter, I introduce you to file permissions, along with the closely related notions of owners and groups. 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 macOS has all these attributes, and you should understand how they work because they influence what you can and can't do with each item.

Before I get into how to 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 any file or folder.

## Read, Write, and Execute

Someone with permission to *read* a file can open it and see what's inside 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 `_screensaver`, which is exactly what it looks like.)
- **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. macOS 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



# Learn Advanced Techniques

Now that you know the basics of working with the command line, I want to show you a few techniques that build on your knowledge and enable you to perform more advanced tasks.

First I tell you how to [Pipe and Redirect Data](#)—two powerful (and related) techniques you can apply to many different commands in order to combine them in useful ways and do more with your data. Next, you'll [Get a Grip on grep](#), a tool that helps you locate files containing specified patterns of characters. Finally, I explain the basics of how you can [Add Logic to Shell Scripts](#), making them much more useful than simple sequences of commands.

As you can imagine, these are but a few of many advanced techniques for using the command line, but I've found them to be consistently helpful, and I hope you will too.

---

## Pipe and Redirect Data

---

Most of the time when you enter commands on the command line, the output—a list of files, the date, the contents of a log, or whatever—is shown directly on the screen. But that isn't always what you want.

For example, maybe the output of some command is a list of hundreds or thousands of files, but that's more information than you need; you want to filter the list to show only files that meet certain criteria. Or, maybe having that list in a Terminal window isn't useful to you, but if it were in a BBEdit document, it would be. In cases like these, you can use either of two commands to take a command's output and do something other than display it on the screen. You can also use a slightly different command to redirect a command to use a different input.

## Pipe (|)

The pipe operator, which is the `|` symbol that you get when you type Shift-`|`, sends the output of a command to another *program*. To use it, you type the first command, then a space, the `|` character, another space, and the name of the second program. Like so:

```
program | other-program
```

We saw the pipe earlier, in [Ps](#), and there are also a few instances of this in [Command-Line Recipes](#), but let me give you some further examples to illustrate how this works and what you might do with it.

If I used the `ls /Library/Preferences` command to show me everything in my `/Library/Preferences` folder, that would be a pretty long list. But suppose I remembered that most of the items in that folder started with `com.apple` and I wanted to see just the last, say, 10 items because that would filter out most of the Apple stuff. And then I remember that the [Tail](#) command does exactly that. Ordinarily, `tail` expects you to give it a *file* as an argument. But instead, I could give it a file *listing* as an argument, using the pipe operator, like so:

```
ls /Library/Preferences | tail
```

And that does what I expect—it shows just the last 10 items from that directory. If I wanted to show the last 15, I could instead enter:

```
ls /Library/Preferences | tail -n 15
```

Most flags and arguments work as usual with piped commands. The exception: commands expecting a file as an argument normally put the file *after* the command, but when you use a pipe, the order is reversed.

How about another example? If I used the `locate` command to find all the files containing *Apple* in the name—again, an awkwardly large number—they’d all scroll by at a dizzying speed. If instead I wanted to be able to page through them one screenful at a time—hey, just like you can do with `less` (see [View a Text File](#))—I can just pipe the output of `locate` into `less`, like so:

```
locate Apple | less
```

# Install New Software

With just the software macOS includes (and perhaps a few shell scripts you write on your own or find on the web), you can do a tremendous number of useful activities on the command line. But sooner or later you're likely to encounter a task that requires a command-line program you don't already have, which means you'll have to find and install it yourself. (Admittedly, this is not for everyone, and if the next few paragraphs give you a headache, skip ahead to [Command-Line Recipes](#) and forget I ever mentioned installing your own software!)

Fortunately, the vast majority of command-line software created for Unix and Unix-like operating systems (such as the various Linux distributions) can run on your Mac too! (Refer back to [What's Unix?](#) for the differences between “Unix” and “Unix-like.”) Tens of thousands of command-line programs are at your disposal! Just a handful of examples:

- **alpine:** An email client
- **ffmpeg:** A tool for recording, converting, and streaming audio and video
- **lynx:** A command-line web browser (yes, really)
- **pdftohtml:** A program that converts—you'll never guess!—PDFs to HTML format
- **postgresql:** A relational database manager
- **wget:** A tool for downloading files from the web

Except... on the command line, it's almost never as simple as downloading an app and running it. Because each Unix and Unix-like operating system is a bit different, in most cases, a given program must be *compiled* for the specific platform in question—that is, the raw source code (in a language such as C) has to be processed by a program called a *compiler* to produce a *binary* file that runs on the target

system. (In fact, compiling can be vastly more complex than this description suggests, but that's the basic idea.)

So, if you have an interest in adding third-party command-line software to your Mac, you'll first need the tools that are required to compile and install them. You can get them easily (read [Use Command Line Tools for Xcode](#), next), and in the process gain a bunch of extra programs that may be useful to you on their own.

Next, you have a choice:

- If you're a glutton for punishment (or want to see how things work), you can [Install Unix Software from Scratch](#). (Do it at least once in your life, just for the experience.)
- If you'd like to make life easier for yourself, however, you can often use a special program called a *package manager* to do the heavy lifting of finding, downloading, and (if necessary) compiling the software you want (see [Use a Package Manager](#)). Package managers are way faster and more convenient than compiling software from scratch, although not every program you may want to install is available in that form.

---

## Use Command Line Tools for Xcode

---

Let's start with something simple: a free software package from Apple called Command Line Tools for Xcode. This collection includes nearly 100 new command-line programs, mostly intended to perform functions useful to developers but not needed by the typical Mac user. However, since you now know your way around the command line, you're not a typical Mac user! And in order to install new command-line software, you'll almost certainly need tools such as `make` (to build a set of binary files from their source files), which in turn relies on a compiler such as `gcc`.

# Learn Command-Line Shortcuts

Throughout this book I’ve mentioned tips, tricks, and shortcuts for performing a wide variety of command-line tasks, and in the next chapter ([Command-Line Recipes](#)) I show you dozens of prebuilt, ready-to-use commands and scripts. In this chapter, however, I wanted to pull together some of my favorite tips for using the Terminal app itself, and `zsh` in particular, that didn’t quite fit anywhere else. As you become more adept at the command line, you’re also likely to be more annoyed at inefficiencies, and this chapter aims to eliminate quite a few of them!

---

## Terminal Tips and Shortcuts

---

I want to begin with a few words about useful but oft-overlooked features in the Terminal app itself—features that work regardless of which shell is running.

**Note:** All these shortcuts work in 10.11 El Capitan and later; many of them also work in earlier versions of macOS.

### Paste Selected Text

Let’s say you select some text in your Terminal window and you want to put that on the command line. You could copy it (⌘-C) and then paste it (⌘-V), but that’s two steps, and you would lose whatever was previously on your clipboard. Terminal has a one-step command to do both together—without losing whatever else might be on your clipboard: with text selected, just press ⌘-Shift-V (or choose Edit > Paste Selection).

## Select Output from the Previous Command

Many commands in Terminal produce output (think `ls`, `cat`, `locate`, and so on). If you want to select all and only the output from the most recent command (for example, to paste into a book you're writing about the command line!), press ⌘-Shift-A.

## Erase Output from the Previous Command

Just as you can *select* output from the previous command (as above), you can *erase* that output—just press ⌘-L. (This erases the command itself from your Terminal window as well.) You can keep pressing ⌘-L to erase the output of previous commands, too!

## Scroll Back to the Previous Command

If you want to *repeat* a previous command, you can use the arrow keys to place previous commands on the command line. But what if you don't want to repeat a command—you just want to scroll up in the Terminal window to where the last command was? Press ⌘-↑. That scrolls the display up to the previous command and highlights it. (You can also, of course, press ⌘-↓ to move to the next command.)

**Tip:** You can combine the last two items! For example, if you press ⌘-↑ followed by ⌘-Shift-A, you can select the output from *earlier* commands—not just the most recent one.

## Search Your Command History

Yet another way to see (and reenter) commands you issued previously is to *search* them. Press Control-R to enter history search mode. The command prompt then looks like this (in `zsh`):

```
bck-i-search: _
```

In `bash`, it looks like this:

```
(reverse-i-search)`': _
```

# 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 chapter, I present a selection of short, easy-to-use commands and customizations you can perform in the `zsh` or `bash` shells. Many use features, functions, and programs I haven’t yet mentioned, 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!

***Misplaced hyphens!*** Your ebook reader may insert extra hyphens into longer lines of command-line text shown in this ebook. Please see the first item under [Basics](#), earlier, for more information about how to avoid extra hyphens.

---

## Change Defaults

---

Most macOS apps, from the Finder to Mail to Music, store their settings in specially formatted property list, or `.plist`, files. Surprisingly, apps 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 app’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 Apple’s Xcode development environment (which is available as a [free download from the Mac App Store](#)). 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 app’s defaults, quit the app if possible (obviously that’s not an option with the Finder or the Dock,

but the recipes that involve those apps include directions to force-quit and relaunch them).

**Tip:** Various websites list hundreds of additional settings you can change—for example, see [defaults-write.com](https://defaults-write.com) and [dotfiles](https://dotfiles.org).

## 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`.)

## Change the Screenshot Format

When you take a screenshot in macOS (using the ⌘-Shift-3, ⌘-Shift-4, or ⌘-Shift-5 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 -string "jpeg";  
killall SystemUIServer
```

Use the same command, but substitute `bmp`, `gif`, `pdf`, `png`, or `tiff` for `jpeg` to use one of those formats.

**Note:** You may notice these file types are in lowercase. If you use uppercase instead, the command still works, but the file extension will also be uppercase. Conversely, if you’re currently getting screenshots with uppercase extensions and you prefer them to be lowercase, this command can solve that problem for you.



# 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](#).

---

## Ebook Extras

---

You can [access extras related to this ebook](#) on the web. Once you're on the ebook's Take Control Extras page, you can:

- Download any available new version of the ebook for free, or buy a subsequent edition at a discount.
- Access the book in both PDF and EPUB formats. (Learn about reading on mobile devices on our [Device Advice](#) page.)
- Read the ebook's blog. You may find new tips or information, as well as a link to an author interview.
- Find out if we have any update plans for the ebook.

If you bought this ebook from the Take Control website, it has been automatically added to your account, where you can download it in other formats and access any future updates.

### More Take Control Books

This is but one of many Take Control titles! We have books that cover a wide range of technology topics, with extra emphasis on Macs and other Apple products.

You can buy Take Control books from the [Take Control online catalog](#) as well as from venues such as Amazon and the Apple Books Store. But it's a better user experience and our authors earn more when you buy directly from us. Just saying...

Our ebooks are available in two formats, PDF and EPUB, which are viewable on any computer, smartphone, tablet, or e-reader. All are DRM-free.

---

## About the Author and Publisher

---



Joe Kissell is the author of more than 60 books about technology. As of 2017, he also became the publisher of Take Control Books, when alt concepts—the company he runs along with his wife, [Morgen Jahnke](#)—acquired the Take Control series from TidBITS Publishing Inc.’s owners, Adam and Tonya Engst.

Joe is also a contributing editor to TidBITS and a senior contributor to Macworld. Before he began writing full-time in 2003, Joe spent nearly eight years managing software development. He holds a bachelor’s degree in Philosophy and a master’s degree in Linguistics.

In his rare non-work hours, Joe likes to travel, walk, cook, eat, and practice t’ai chi. He and Morgen live in Saskatoon, Saskatchewan, Canada, with their two children and their cat. To contact Joe about this book, [send him email](#) and *please* include [Take Control of the Mac Command Line with Terminal](#) in the subject of your message. You can also follow him on Twitter ([@joekissell](#)) or visit his personal website, located at [JoeKissell.com](#).

### Author’s Acknowledgments

Thanks to Geoff Duncan for an outstanding editing job, and to the following people for suggesting command-line recipes: Marshall Clow, Keith Dawson, Geoff Duncan, Chuck Goolsbee, Jon Gotow, Kevin van Haaren, Andrew Laurence, Peter N Lewis, Chris Pepper, Larry Rosenstein, and Nigel Stanger.

## Credits

- Publisher: Joe Kissell
- Editor: Geoff Duncan
- Cover design: Sam Schick of [Neversink](#)
- Logo design: Geoff Allen of [FUN is OK](#)

# Also by Joe Kissell

Click any book title below or [visit our web catalog](#) to add more ebooks to your Take Control collection!

[\*Are Your Bits Flipped?\*](#): Discover common tech misconceptions and how to avoid them.

[\*Take Control of 1Password\*](#): Use this powerful password manager to create, store, enter, and sync personal data on all your devices.

[\*Take Control of Apple Mail\*](#): Learn the ins and outs of Apple's email app in macOS and iOS.

[\*Take Control of Automating Your Mac\*](#): Work more efficiently on your Mac with time-saving shortcuts of all kinds.

[\*Take Control of Backing Up Your Mac\*](#): Protect your Mac's valuable data from any sort of mishap.

[\*Take Control of DEVONthink 3\*](#): Master this powerful information management tool.

[\*Take Control of iCloud\*](#): Make the most of Apple's online service for storing, syncing, and sharing data.

[\*Take Control of Monterey\*](#): Discover what's new in macOS 12, and get all the information you need to upgrade safely.

[\*Take Control of Your Digital Legacy\*](#): Make sure your important digital information is preserved for future generations.

[\*Take Control of Your Online Privacy\*](#): Learn what's private online (not much)—and what to do about it.

[\*Take Control of Your Paperless Office\*](#): With your Mac, scanner, and this ebook, you'll finally eliminate the chaos of overflowing paper.

[\*Take Control of Your Passwords\*](#): Overcome password overload! Stay safe while eliminating the hassles and confusion of passwords.

# Copyright and Fine Print

*Take Control of the Mac Command Line with Terminal, Third Edition*

ISBN: 978-1-990783-03-6

Copyright © 2022, Joe Kissell. All rights reserved.

[alt concepts](#), 419 8B-3110 8th St. East, Saskatoon, SK S7H 0W2 Canada

**Why Take Control?** We designed Take Control electronic books to help readers regain a measure of control in an oftentimes out-of-control universe. With Take Control, we also work to streamline the publication process so that information about quickly changing technical topics can be published while it's still relevant and accurate.

**Our books are DRM-free:** This ebook doesn't use digital rights management in any way because DRM 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, they 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.

**Remember the trees!** You have our permission to make a single print copy of this ebook for personal use, if you must. Please reference this page if a print service refuses to print the ebook for copyright reasons.

**Caveat lector:** Although the author and alt concepts 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 book is distributed "As Is," without warranty of any kind. Neither alt concepts 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.

**It's just a name:** Many of the designations in this ebook 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. No such use, or the use of any trade name, is meant to convey endorsement or other affiliation with this title.

**We aren't Apple:** 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 registered trademarks or service marks of Apple Inc. If you're into that sort of thing, you can view a [complete list](#) of Apple Inc.'s registered trademarks and service marks.