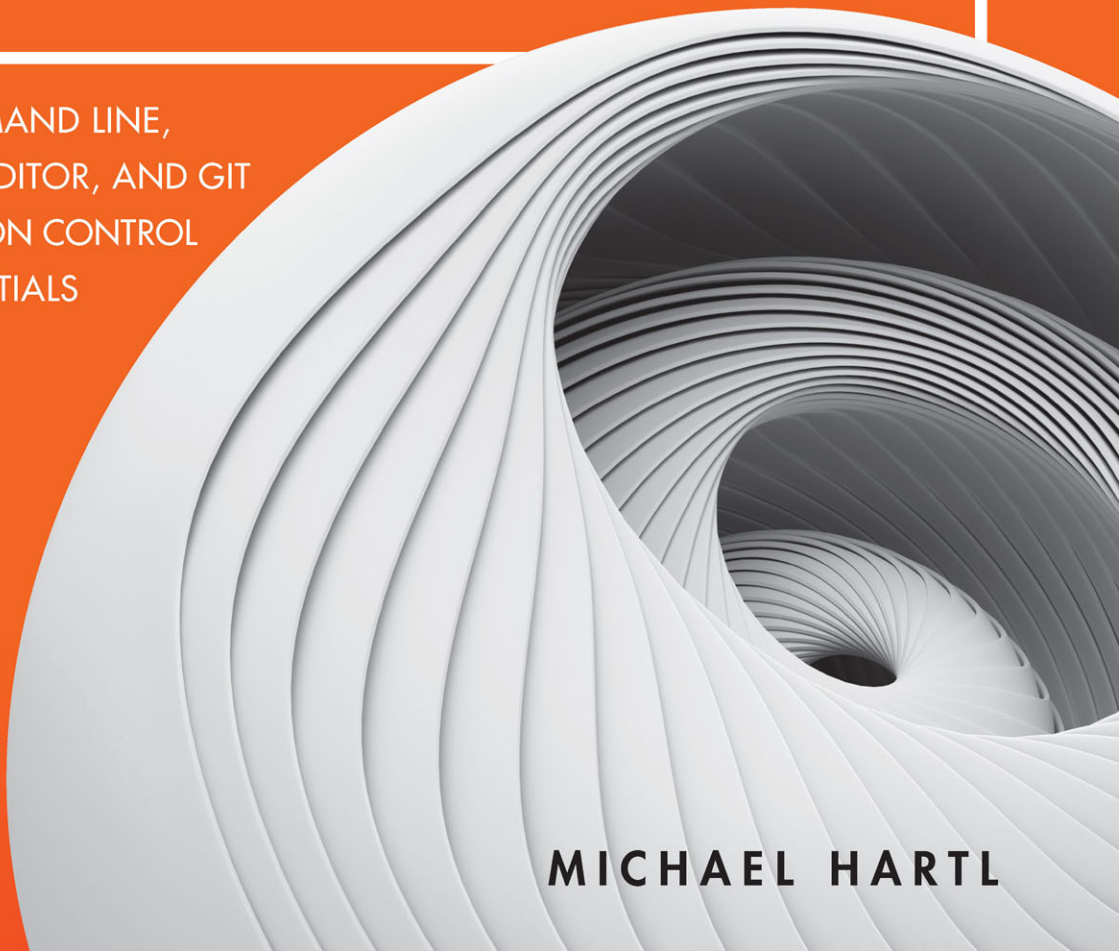
learn enough

DEVELOPER TOOLS TO BE DANGEROUS

COMMAND LINE, TEXT EDITOR, AND GIT VERSION CONTROL ESSENTIALS



MICHAEL HARTL

Praise for Learn Enough Tutorials

"Going through *Learn Enough Git* is wonderful. I am actually learning... I've done three other Git tutorials and still felt so lost. Doing it all now makes so much sense. It's like a light bulb."

—Janelle Staar

"I bought the *Learn Enough Command Line to Be Dangerous* last fall, and it's paid off sooooo many times in my new job. During my first week, I had a manager sitting right beside me giving me the 'go here, go there, do this, etc.' Having watched, read, and done the exercises, I was confident in getting around the CLI [command-line interface]—and even had him asking, 'What was that shortcut?' For this, I thank you. Now I need a 'Learn even more CLI to be dangerouser.'"

—Thomas Thackery

"I must say, this Learn Enough series is a masterpiece of education. Thank you for this incredible work!"

—Michael King

"I want to thank you for the amazing job you have done with the tutorials. They are likely the best tutorials I have ever read."

—Pedro Iatzky

This command means "Press the Escape key, then type 'colon q exclamation-point', then press the return key." We'll learn in a moment what this does and why, but for now we'll start by practicing it a couple of times in the exercises.

Box 5.4: Holy Wars: vi vs. Emacs

The Jargon File (http://www.catb.org/jargon/html/) defines *holy wars* as follows:

> **holy wars**: n.
> [from *Usenet*, but may predate it; common] *flame wars* over *religious issues*. The paper by Danny Cohen that popularized the terms *big-endian* and *little-endian* in connection with the LSB-first/MSB-first controversy was entitled *On Holy Wars and a Plea for Peace*.
>
> Great holy wars of the past have included *ITS* vs. *Unix*, *Unix* vs. *VMS*, *BSD* Unix vs. System V, *C* vs. *Pascal*, *C* vs. FORTRAN, etc. In the year 2003, popular favorites of the day are KDE vs. GNOME, vim vs. elvis, Linux vs. [Free|Net|Open]BSD. Hardy perennials include *EMACS* vs. *vi*, my personal computer vs. everyone else's personal computer, ad nauseam. The characteristic that distinguishes holy wars from normal technical disputes is that in a holy war most of the participants spend their time trying to pass off personal value choices and cultural attachments as objective technical evaluations. This happens precisely because in a true holy war, the actual substantive differences between the sides are relatively minor. See also *theology*.

As noted in the Jargon File entry, one of the longest-raging holy wars is fought between proponents of vi and its arch-rival Emacs (sometimes written "EMACS"), both of which have played important roles in the Unix computing tradition. Both also retain much popular support, although my guess is that, with the popularity of Vim, vi has taken a decisive lead in recent years. Of course, this is just the sort of statement that serves to perpetuate a holy war—likely prompting Emacs partisans to, say, make claims about the superior power and customizability of their favorite editor.

If you wanted to start a *new* holy war, you might try something like, "Happily, the vi vs. Emacs holy war is now mostly a historical curiosity, as anyone who's anyone has switched to a modern editor like Atom or Sublime." It's going to be quite a show—better bring some popcorn (Figure 5.5).[10]

10. Image courtesy of Cartno Studio/Shutterstock.

Figure 5.5: Watching a holy war play out can be entertaining.

5.2.1 Exercises

1. Start Vim in a terminal, then run the Most Important Vim Command™.
2. Restart Vim in a terminal. Before typing anything else, type the string "This is a Vim document." What happened? Confusing, right?
3. Use the Most Important Vim Command™ to recover from the previous exercise and return to the normal command-line prompt.

5.3 Editing Small Files

Now that we know the Most Important Vim Command™, we'll start learning how to use Vim for real by opening and editing a small file. In Section 5.2, we started by

running `vim` by itself, but it's more common to use a filename as an argument. Let's navigate to the home directory of our system and then run such a command, which will either open the corresponding file (if it exists) or create it (if it doesn't):[11]

```
$ cd
$ vim .bashrc
```

Here `.bashrc` is a standard configuration file for Bash.[12]

As noted above, the `vim .bashrc` command will automatically create the corresponding `.bashrc` file if it doesn't already exist on your system. This important file is used to configure the *shell*, which is the program that supplies a command line—in this case, *Bash*, which is a pseudo-acronym that stands for *Bourne Again SHell* (also written as "Bourne-again shell").[13] (Recall from Box 2.3 that the default on macOS is now Zsh, so if you're on a Mac you should follow the instructions there to switch to Bash if you haven't already.)

As is common on Unix-based systems, the configuration file for Bash begins with a dot, indicating (as noted in Section 2.2) that the file is *hidden*. That is, it doesn't show up by default when listing directory contents with `ls`, or even when viewing the directory using a graphical file browser.

We'll learn in Section 5.4 how to save changes to this file, but for now we're just going to add some dummy content so that we can practice moving around. In Section 5.2, we learned that Vim starts in normal mode, which means that we can change location, delete text, etc. Let's go into insertion mode to add some content. The first step is to press the `i` key to *insert* text. Then, type a few lines (separated by returns), as shown in Listing 5.3.[14] (There may be other existing content, which you should simply ignore.)

11. Technically, the file isn't actually created until you save it (Section 5.4), but you get the idea.

12. See this Stack Overflow thread (https://stackoverflow.com/questions/902946/about-bash-profile-bashrc-and-where-should-alias-be-written-in) if you're curious about where the "rc" comes from.

13. The first program in the sequence was the Bourne shell; in line with the Unix tradition of terrible puns, its successor is called the "Bourne again" (as in "born again") shell.

14. Recall from Section 4.1 that a tilde ~ is used to indicate the home directory, so `~/.bashrc` in the Listing 5.3 caption refers to the Bash configuration file in the home directory.

Listing 5.3: Adding some text after typing `i` to insert.
~/.bashrc

```
lorem ipsum
dolor sit amet
foo bar baz
I've made this longer than usual because I haven't had time to make it shorter.
```

After entering the text in Listing 5.3, press **ESC** (the escape key) to switch from insertion mode back to normal mode.

Now that we have some text on a few lines, we can learn some commands for moving around small files. (We'll cover some commands for navigating large files in Section 5.6.) The easiest way to move around is to use arrow keys—up, down, left, right—which is what I recommend.[15] Vim has literally jillions of ways of moving around, and if you decide to use Vim as your primary text editor I recommend learning them, but for our purposes the arrow keys are fine. The only two additional commands I feel are essential are the ones to move to the beginning and end of the line, which are `0` and `$`, respectively.[16]

5.3.1 Exercises

1. Use the arrow keys to navigate to line 4 in the file from Listing 5.3.
2. Use the arrow keys to go to the end and then the beginning of line 4. Cumbersome, eh?
3. Go to the beginning of line 4 by using the command mentioned in the text.
4. Go to the end of line 4 using the command mentioned in the text.

5.4 Saving and Quitting Files

Having learned a little about moving around and inserting text, now we're going to learn how to save a file. Our specific example will involve making a useful new Bash

15. Vim purists will tell you that there's a better way, namely, to use the `h`, `j`, `k`, `l` to move around, but it takes a lot of practice for this to become intuitive, and it's certainly not necessary to be *dangerous*.

16. These are not native keybindings (on macOS they would be Command–left arrow and Command–right arrow), which as noted in the introduction to Chapter 5 makes it harder to learn them. This is just one of many reasons I don't generally recommend beginners use Vim as their primary editor.

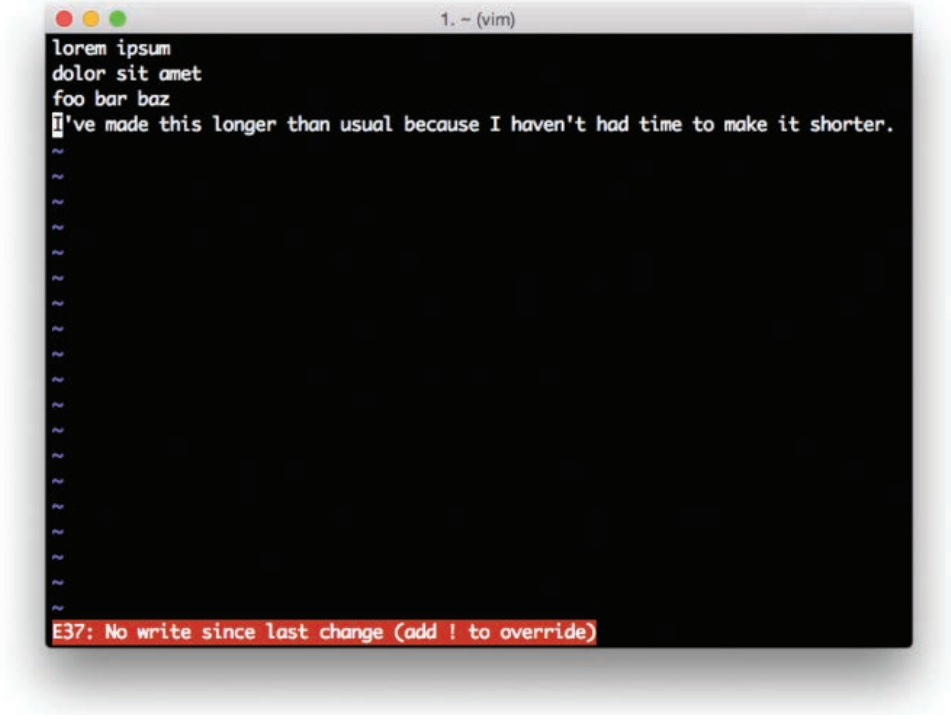


Figure 5.6: Trying to quit a file with unsaved changes.

command, but first we have to deal with the current state of the Bash profile file. The text we added in Listing 5.3 is gibberish (at least from Bash's perspective), so what we'd like to do is quit the file without saving any changes. For historical reasons, some Vim commands (especially those having to do with file manipulation) start with a colon `:`, and the normal way to quit a file is with `:q<return>`, but that only works when there are no changes to save. In the present case, we get the error message "No write since last change (add ! to override)", as shown in Figure 5.6.

Following the message's advice, we can type `:q!<return>` to force Vim to quit without saving any changes (Figure 5.7), which returns us to the command line.

You may have noticed that we're now in a position to understand the Most Important Vim Command™ introduced in Section 5.2: No matter what terrible things you might have done to a file, as long as you type `ESC` (to get out of insertion mode if

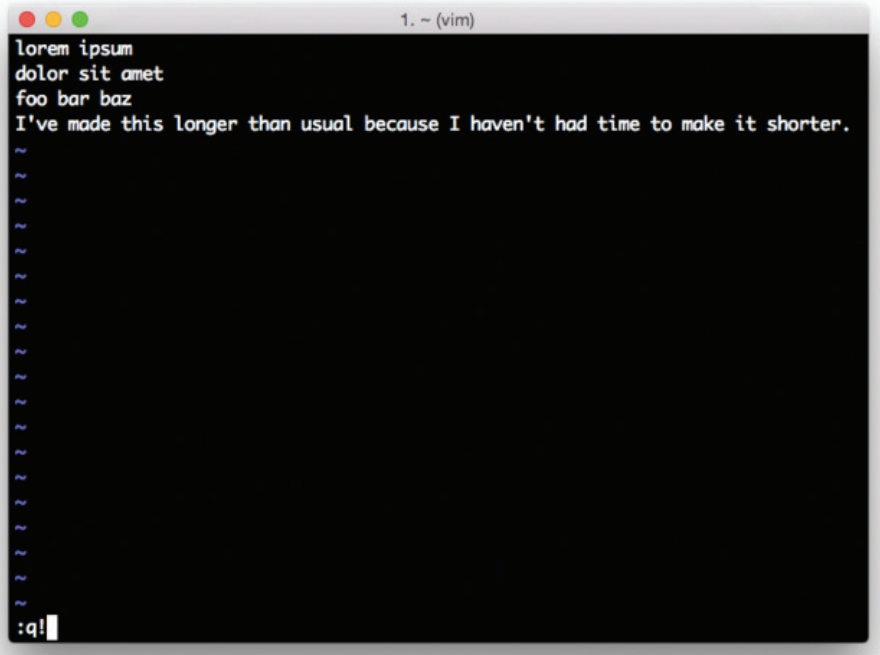


Figure 5.7: Forcing Vim to quit.

necessary)[17] followed by `:q!<return>` (to force-quit) you are guaranteed not to do any damage.

Of course, Vim is only really useful if we can save our edits, so let's add some useful text and then write out the result. As in Section 5.3, we'll work on the `.bashrc` file, and the edit we'll make will add an *alias* to our shell. In a computing context, an alias is simply a synonym for a command or set of commands. The main use for Bash aliases is defining shorter commands for commonly used combinations.[18]

17. Hitting `ESC` while in normal mode does no harm, so it's a good idea to include this step in any case.

18. To learn how to write aliases using Zsh, see "Using Z Shell on Macs with the Learn Enough Tutorials" (https://news.learnenough.com/macos-bash-zshell). TL;DR: The syntax is identical; the only difference is that you edit a file called `.zshrc` instead of `.bashrc`. (Indeed, in Bash you can actually edit a file called `.bashrc` instead, which makes the parallel even clearer.)