Extracted from:

Small, Sharp Software Tools

Harness the Combinatoric Power of Command-Line Tools and Utilities

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Small, Sharp Software Tools

Harness the Combinatoric Power of Command-Line Tools and Utilities

Brian P. Hogan
edited by Tammy Coron

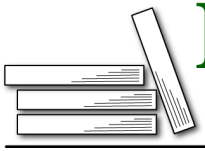
Small, Sharp Software Tools

Harness the Combinatoric Power of Command-Line Tools and Utilities

Brian P. Hogan

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Susan Conant
Development Editor: Tammy Coron
Copy Editor: L. Sakhi MacMillan
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-296-1
Book version: P1.0—May 2019

When you hear about the command-line interface, or CLI, from other developers, you often hear about how much faster it is than a graphical environment. But if you're like most developers, you're probably pretty good with your computer. You know some keyboard shortcuts that help you move even faster. The best way to see the real value of the CLI is to just dive right in to some hands-on activities, while learning a few time-saving techniques along the way. In this chapter, you'll use the command line to do some things you probably know how to do already through the graphical interface. You'll work with files and directories, navigate around a bit, install some software, and get more information about the commands you're typing. We're just going to scratch the surface in this first chapter; we'll go into more detail on many of the topics in the rest of the book, and we'll even review these topics again to help them stick. This chapter is designed to give you a taste of working with the command-line interface.

But first, you have to find the command-line interface.

Accessing the Command-Line Interface

Getting access to the command-line interface, often called the *shell*, varies based on your operating system. Usually, you'll access it through a program called a *terminal*, short for *terminal emulator*.

If you're on a Linux machine with a GUI, you can usually launch its terminal app with Control+Alt+t, or by searching for a Terminal program in your list of programs. When the terminal opens, you'll see something like this:

```
brian@puzzles:~$
```

This is the *prompt*, and it's where you'll enter commands. We'll explore its meaning shortly.

To access the command-line interface on a Mac, hold down the Command key on your keyboard and press Space. This brings up the Spotlight window. Type terminal into the box and press Enter. This launches the Terminal program. The prompt you'll see looks like this:

```
puzzles:~ brian$
```

Windows 10 has a few command-line interfaces. The classic Command Prompt and the PowerShell interfaces aren't compatible with the command-line interface on Linux, BSD, or macOS systems. They have their own commands and approaches to solving problems. So you won't be using those interfaces in this book. Instead, you'll use the Bash on Windows feature for Windows 10.

To do this, you'll enable the Windows Subsystem for Linux[1] and then download Ubuntu from the Windows Store. It's a free download that installs a version of Ubuntu on top of your Windows operating system, and it's fully supported by Microsoft. There are other flavors of Linux available, but you'll use Ubuntu in this book.

First, open the Control Panel and select Programs. Then, click Turn Windows Features On Or Off. Locate and enable the option for "Windows Subsystem for Linux." Then reboot your computer.

When the computer reboots, open the Windows Store and search for Ubuntu. Install it and launch it once it installs.

You'll see a console window open and Ubuntu will install some additional components and configure itself:

```
Installing, this may take a few minutes...
Installation successful!
```

Once it finishes extracting, you'll get prompted to create a new user account and password. This new account isn't connected to your Windows account in any way. To keep things easy to remember, use the same username as your Windows user. For the password, choose anything you like. You won't see your password as you type it, as it's hidden for security purposes.

```
Please create a default UNIX user account. The username does not need to match
your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: brian
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Default UNIX user set to: brian
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

You'll then be placed at a Bash prompt. Type `exit` to close the window.

To open Bash on Windows in the future, open a new Command Prompt or PowerShell window and type `bash` again. Alternatively, choose Ubuntu from the Start menu.

Now that you have the CLI open, you can start exploring.

1. https://docs.microsoft.com/en-us/windows/wsl/install-win10

Getting Your Bearings

When you first open the CLI, you'll be presented with something that looks like this:

```
brian@puzzles:~$
```

This is the prompt, and it's the CLI's way of telling you it's ready for you to type a command. This prompt is from the Ubuntu operating system. If you're on a Mac, your prompt might look like this:

```
puzzles:~ brian$
```

These prompts may look cryptic at first, but there's valuable information here. The prompts in these examples show the username (brian), the computer's hostname (puzzles), and the *current working directory*, or your current location on the computer's filesystem.

In this case, the current working directory is represented by a tilde (`~`), which means your *home directory*. Your home directory is where you'll find documents, music, and settings for your programs. You have total control over your home directory. You can create and delete files and directories, move things around, and even install whole programs without administrative privileges. When you launch the CLI, it'll open the session in your home directory.

Why the Tilde Is Used to Represent the Home Directory

In the 1970s, the Lear-Siegler ADM-3A terminal was in widespread use. On the ADM-3A keyboard, the tilde shared the same key as the `Home` key.

The computer's disk stores files in a hierarchy of folders, or *directories*, which are called the *filesystem*. You'll explore this in detail in Chapter 3, Navigating the Filesystem, on page ?. When you use the GUI, you click a folder to open it and see its contents, and an indicator at the top of the GUI window tells you where you are on the filesystem.

Your prompt may tell you what directory you're currently viewing. But a clearer way to tell is with the `pwd` command, which stands for "print working directory."

At the prompt, type:

```
$ pwd
/home/brian
```

The command prints the full *path*, or location on the filesystem, to the current working directory. In this case, the current working directory is your home

directory, and the path you see depends on which operating system you're using. For example, on macOS, you'll see /Users instead of /home.

In a GUI, you'd look at the folder name in the top of the file window to see where you are. On the CLI, you use pwd to get that information.

Now that you know where you are, look at the contents of your home directory by using the ls command. This command lists the contents of the directory.

```
$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
```

You might see different files and directories in your home directory, as each operating system sets things up a little differently.

If your system has a GUI, you'll see a directory named Desktop, and anything you place in that directory will show up on your computer's graphical desktop. After all, any shortcuts, directories, or files on your Desktop have to be stored *somewhere* on your computer, right?

Using the CLI, navigate to your Desktop directory using the cd command. The cd command is short for "change directory." It's somewhat like clicking a folder to open it up in the GUI environment, except it's a little more powerful because you can use it to jump to any directory on your filesystem immediately, without going through any intermediate directory. You'll look at that in Chapter 3, Navigating the Filesystem, on page ?.

Type this command to navigate to the Desktop directory if you're using a Mac or running a Linux distribution with a GUI:

```
$ cd Desktop
```

If you're using Bash on Windows, the Desktop directory isn't located in the same place. The Windows Subsystem for Linux uses its own filesystem that's separate from the one that Windows uses, with its own home directory. However, you can still follow along, as the Windows Subsystem for Linux makes your Windows desktop available. Execute the following command, substituting your_username for your Windows username.

```
$ cd "/mnt/c/Users/your_username/Desktop"
```

You're now in the Desktop directory, which you can verify with the pwd command just to make sure.

Now, let's create a file on your desktop that contains some text.

Creating and Reading Files

The echo command takes text and prints it back to the screen. At the prompt, type the following command:

```
$ echo Hello there
```

You'll see the text you entered printed back to you. You'll use echo in scripts you write to display instructions or output to users. But you can also use it to insert text into files if you combine it with a feature called *redirection*.

Let's create a text file in your Desktop directory that contains the text "Hello". You'll call this file greetings.txt. At the prompt, type:

```
$ echo Hello > greetings.txt
```

When you press the Enter key you won't see any visual feedback because the output of the echo command was redirected to the file you specified. If you look at your computer's graphical desktop, you should see a new icon that wasn't there before for greetings.txt. You just used the command-line interface to create a file that contains some text. Right now it's a cool parlor trick, but the implications are important; this is one way you can programmatically create files on the filesystem.

You can do so much more with redirection, but let's move on and continue your introductory tour. You'll work with echo a lot more soon enough.

But first, make sure that this text file does indeed contain the text you placed inside. You could open this file with a graphical text editor, but you can display the contents of any file from the command line quickly with the cat command:

```
$ cat greetings.txt
```

This command reads the contents of a given file and displays them on the screen.

Since the file was small, it all fit on one screen. But try reading the contents of a much larger file and see what happens. On macOS and Ubuntu, there's a dictionary of words located at /usr/share/dict/words. Use the cat command to display its contents:

```
$ cat /usr/share/dict/words
```

If you're using Ubuntu on Windows 10, you might not have this file, but you can try using a different file for this exercise instead.

You'll see the contents of the file scroll by until it reaches the end. You'll encounter this a lot when working on the command line. You might have the

source code to a program you're working on, a document you're editing, or a log file you're using to diagnose an issue.

You can use the more and less commands to read longer files one page at a time. Use the less command to read the contents of that huge dictionary file:

```
$ less /usr/share/dict/words
```

You'll see the first page of the file on the screen. Use the Enter key to display the next line, the Spacebar to jump to the next page, and the q key to exit and return to your prompt.

Since you're using less, you can use the arrow keys to move forward and backward through the file. On most systems, the less and more commands run the same program. The more command is a legacy program with limited features, but it's still quite popular in documentation and on some minimalist operating systems due to its smaller size.

Now, let's look at redirecting program output to files and other programs, something you'll find yourself doing quite often.

Redirecting Streams of Text

When you used the echo command to create a file, you took the output of one command and directed it somewhere else. Let's look at this in more detail.

Execute this command to view all of the running processes on your computer:

```
$ ps -ef
```

The ps command shows you the processes running on your computer, and the -ef options show you the processes for every user in all sessions. Once again, the output of the command streams by.

This problem can be solved in a couple of ways. The first approach would be to capture the output to a file by using the > operator, just like you did to create a text file with echo:

```
$ ps -ef > processes.txt
```

You could then open that file in your favorite text editor and read its contents, or you could use less to view it one page at a time.

But a better way involves less steps. You can redirect the output of the ps command to the less command by using the pipe (|) character. Give it a try:

```
$ ps -ef | less
```

Now you can read the output more easily.

Clearing and Resetting Your Terminal

After typing several commands, your terminal might become a little harder to read. Type the clear command to clear the contents of the screen. Everything in the terminal will be wiped away and your prompt will start at the top of the screen just as if you'd opened a new terminal.

In some cases, you may need to do more than just clear your screen. Your terminal might begin behaving strangely after you've run some programs—for example, if you accidentally used the cat command to read an executable file. Instead of closing the terminal and starting a new one, try the reset command, which resets your terminal session and usually fixes the problem. The reset command works in conjunction with your Terminal program, so its actual behavior depends on how your Terminal program is configured.

As you work on the command line, you'll find yourself taking the output of one program and sending it off to another program, which will then output something new. Sometimes you might use three or four programs together to achieve a result. Imagine the data that flows as a stream of text that can be processed by programs. Each program that processes the text does one thing. You'll learn more detail about this concept later in the book. For now, turn your attention back to files and directories.

Creating Directories

Directories help you organize your files and projects. The mkdir command lets you create a directory. Let's use it to create a directory called website on the Desktop. At the prompt, assuming you're still in your Desktop directory, type:

```
$ mkdir website
```

If your computer's graphical desktop is visible, you'll see the directory appear. Otherwise, use the ls command to view it:

```
$ ls
```

Once you've created the directory, you can use the cd command to navigate into that directory:

```
$ cd website
```

You can then create new directories inside of this directory for images, style sheets, and scripts. You'll explore more efficient ways to create complex directory structures for projects later. For now, take a look at how you can get back to your home directory.

Going Home

No matter where you are, you can execute a single command that will take you back to your home directory. As you recall, the tilde (~) always refers to *your* home directory.

So if you type

```
$ cd ~
```

you'll return to your home directory regardless of where you are on the filesystem.

You can save a couple of keystrokes, because entering `cd` followed by `Enter` will do the same thing. Try it out:

```
$ cd
```

Either of these methods will always take you back to your home directory, no matter where you are on the filesystem.

Using Autocompletion

If you need to reference a filename or directory in a command, you can type part of the name, followed by the `Tab` key, and the CLI will attempt to autocomplete the word for you. Try this out. Switch to the `Documents` directory in your home directory like this:

```
$ cd ~/Doc<Tab>
```

As soon as you press `Tab`, the word `Documents` will expand. This technique serves two purposes. First, it saves you from typing the whole name, which means you'll make less typos. But second, the CLI only completes filenames and directory names it can find. If it can't complete it, there's a good chance it doesn't have enough information, or the file doesn't actually exist. Try this out. Navigate to your home directory:

```
$ cd
```

Then type:

```
$ cd D<Tab>
```

You won't see anything. This is because Bash doesn't have enough information to do the completion, because you probably have a `Documents` directory as well as a `Downloads` directory.

But if you press `Tab` again, you'll see a list of possible options:

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

Type a few more characters and press Tab to let it autocomplete the rest of the directory name.

Now, try autocompleting var from your Home directory:

```
$ cd va<Tab>
```

This time, pressing Tab doesn't do anything. And pressing it a second time doesn't either, since there's no var directory within the current directory. You can use this as a good test while you're learning how to navigate around; if you can't autocomplete the filename or directory, you might not be looking in the right spot.

Some tasks, like creating files outside of your home directory, or installing programs system-wide, require that you run commands with additional privileges.

Elevating Privileges

You have complete and total control over all of the files in your home directory. But you don't have free reign over directories and files across the whole disk. You can only do certain things with superuser privileges. On Linux, Unix, BSD, and macOS systems, this is called the *root* user. To keep things more secure and to prevent accidents, regular user accounts are restricted from modifying things outside of their home directories.

Try to create a new directory called /var/website:

```
$ mkdir /var/website
```

This command will fail with the following error:

```
mkdir: /var/website: Permission denied
```

You're not allowed to create files in the /var directory; only certain users can do that. But thanks to the sudo command, you can execute a single command as the root user, without logging in as that user. To use it, prefix the previous command with sudo, like this:

```
$ sudo mkdir /var/website
```

Think of this sudo command as "superuser do mkdir /var/website." The command will complete successfully, and you can verify that it exists by using the ls command to view the contents of the /var directory:

```
$ ls /var/
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  snap
```

```
spool  tmp  website
```

The website directory is now listed in the output.

The sudo command is powerful but dangerous. You're running the command as a privileged user, so if the command does something sinister, you could be in a lot of trouble. It also bypasses any file permission restrictions, meaning you could accidentally alter or delete files owned by anyone if you accidentally ran the wrong command. Use this with care!

One place you're likely to use sudo is when modifying system-wide configuration files, or, as you'll try next, installing additional programs or tools on your operating system.