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

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

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

CHAPTER 4

Working with Files and Directories

You probably spend a lot of your time working with files and directories. You write programs, change configuration files, copy and move files around your projects, rename files, and maybe even back things up.

In the last chapter, you learned how to use the CLI to move around the filesystem. In this chapter, you'll manipulate that filesystem. You'll concatenate files, read larger files, and manage permissions. You'll move, rename, and copy files and directories, all without ever using a graphical tool.

Creating Files

In Creating and Reading Files, on page ?, you learned how to use the echo command and redirection to create files. This is one of many ways to create files on the command line. Let's look at a few alternatives.

If you only need to create an empty file, you can use the touch command. This command is designed to update the timestamp of a file. It's common for programs to watch a file for changes and then react to those changes by executing a process. For example, you might have a process that runs tests whenever a file changes, but you don't actually want to open the file and make a change. You can use touch to modify the file without actually changing the contents. This would then trigger the program or process monitoring the file.

However, if the specified file doesn't exist, the touch command creates the file. This makes touch a very popular tool for creating files quickly.

Test it out. Navigate to your home directory:

```
$ cd
```

Now, use touch to create a new file named file.txt:

```
$ touch file.txt
```

Verify that it exists by using the ls -lh command:

```
$ ls -lh file.txt
-rw-r--r-- 1 brian brian 0 Mar  2 12:50 file.txt
```

The file doesn't have any contents, but it was created successfully. This is a handy way to create a blank file that you can then modify elsewhere.

Remember that touch updates a file's timestamp whenever you run it. Wait a minute and run the touch command on this file again. Then get a new listing:

```
$ touch file.txt
$ ls -alh file.txt
-rw-r--r-- 1 brian brian 0 Mar  2 12:51 file.txt
```

You'll notice that the timestamp has changed.

You can use touch to operate on more than one file at once. All you have to do is provide it with a list of filenames, separated with spaces. Give it a try:

```
$ touch index.html about.html style.css
```

This creates the three files, index.html, about.html, and style.css in the current directory. Verify this with the ls command:

```
$ ls file.txt index.html about.html style.css
about.html  file.txt  index.html  style.css
```

You're not limited in where you create the files. You can create a file in the current directory and in the Documents directory as well, all with a single command:

```
$ touch this_goes_in_home.txt Documents/this_goes_in_Documents.txt
```

By specifying a relative path or a full path to a directory, you can create files anywhere on your filesystem.

Creating Files with Content

The touch command creates blank text files, and as you already know, you can capture the output of programs to a text file by redirecting the program's output to a file. Let's review this by creating a new text file in the current directory that contains the text "Hello, World". Use the echo command to print out the text and then redirect it to a file:

```
$ echo 'Hello, World' > hello.txt
```

If the file hello.txt doesn't exist, it gets created. If it does exist, *it gets overwritten.* So you have to be incredibly careful with this command. You could accidentally erase a file's contents this way.

To append text to the file instead of overwriting its contents, use >> instead of >. So, to append another line to the file hello.txt, you can do this:

```
$ echo 'How are you today' >> hello.txt
```

You can use the > and >> symbols to redirect any program's output messages to a file. For example, if you wanted to save the list of files in the current directory to a file, it's as easy as:

```
$ ls -alh > files.txt
```

You can then append the output of another command to the same file using >>:

```
$ ls -alh ~/Documents >> files.txt
```

The files.txt file will contain the output of both commands. You'll dive into how this works in greater detail in Chapter 5, Streams of Text, on page ?.

Writing Multiple Lines to a File

You can create files from program output, and you can create a new file with a line of text, but you can also create a new file with multiple lines of text right from the command line, without opening a text editor.

In Creating and Reading Files, on page ?, you used the cat command to view the contents of files. The cat command reads the contents of files and displays them to the screen. However, you can use cat to create new files. Let's create a simple text file with several lines. Execute this command:

```
$ cat > names.txt
```

After pressing Enter, you'll see a different prompt and a flashing cursor:

```
>
```

The cat command is waiting for input. Type the following lines, pressing Enter after each one:

```
> Homer
> Marge
> Bart
> Lisa
> Maggie
```

After the last line of the file, press Enter one more time, then press Ctrl+d. This saves the contents to the file.

Use cat again to view the file to ensure the contents were saved:

```
$ cat names.txt
Homer
Marge
Bart
Lisa
Maggie
```

Here's how that worked. You told cat to start accepting text from the keyboard. Every time you pressed the Enter key, cat saved the line to the file. Pressing Enter after the last line (Maggie) saved *that* line to the file as well. The sequence Ctrl+d exits the process. If you forget to press Enter after the last line, you won't save the last line to the file.

To avoid that issue entirely, tell cat to terminate when it sees a specific string on its own line. Try this command:

```
$ cat << 'EOF' > names.txt
> Homer
> Marge
> Bart
> Lisa
> Maggie
> EOF
```

Instead of a blank line at the end, you use the text EOF. Then, when you invoke cat, you tell it to take in keyboard input until it sees the line EOF. The text EOF is short for "end of file," and it's mostly a convention; you can use any combination of characters you want, like DONE or END.

This is a great way to create files without switching to your GUI and breaking your flow. You'll use this throughout the book and explore how it works in Chapter 5, Streams of Text, on page ?.

Combining Files

The cat command is the go-to tool for looking at the contents of small files. If you send multiple files to the cat command, it will read them all in the order you specified. Try it. Create two files in your current directory:

```
$ echo "Hello" > hello.txt
$ echo "Goodbye" > goodbye.txt
```

Now, use the cat command to read both files:

```
$ cat hello.txt goodbye.txt
Hello
Goodbye
```

As you can see, the output shows the content of both files. The cat command is actually designed to concatenate files. And as you've seen already, you can redirect standard output to a file using the > symbol.

Let's see this in action. Websites often have a common header and footer, and instead of repeating that content in every file, you can store the common contents in templates, and then place the page-specific bits in their own files. Try it out.

First, use cat to create a new file named header.html with the following content:

files/header.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>My Site</title>
  </head>
  <body>
    <div class="content">
      <header>
        <h1>AwesomeCo</h1>
      </header>
```

Create it with the following command:

```
$ cat << 'EOF' > header.html
> <!DOCTYPE html>
> <html lang="en-US">
>   <head>
>     <meta charset="utf-8">
>     <title>My Site</title>
>   </head>
>   <body>
>     <div class="content">
>       <header>
>         <h1>AwesomeCo</h1>
>       </header>
> EOF
```

Next, create a file named footer.html with the following content:

files/footer.html

```
      <footer>
        <small>Copyright &copy; 2019 AwesomeCo</small>
      </footer>
    </div>
  </body>
</html>
```

Use the same method to create this file too:

```
$ cat << 'EOF' > footer.html
>       <footer>
>         <small>Copyright &copy; 2019 AwesomeCo</small>
>       </footer>
>     </div>
>   </body>
> </html>
> EOF
```

Finally, create a file named main.html that contains this:

files/main.html

```
<main>
  <h2>Welcome!</h2>
  <p>This is the main page!</p>
</main>
```

Here’s the command:

```
$ cat << 'EOF' > main.html
> <main>
>   <h2>Welcome!</h2>
>   <p>This is the main page!</p>
> </main>
> EOF
```

Now, join the three files to create a new file named index.html using cat:

```
$ cat header.html main.html footer.html > index.html
```

Print out the contents of the new file to verify that it contains the output you expected:

```
$ cat index.html
<!DOCTYPE html>
<html lang=“en-US”>
  <head>
    <meta charset=“utf-8”>
    <title>My Site</title>
  </head>
  <body>
    <div class=“content”>
      <header>
        <h1>AwesomeCo</h1>
      </header>
<main>
  <h2>Welcome!</h2>
  <p>This is the main page!</p>
</main>
```

```
      <footer>
        <small>Copyright &copy; 2019 AwesomeCo</small>
      </footer>
    </div>
  </body>
</html>
```

The lines in all three files were combined into a single file. The indentation looks off since we didn't indent the contents of the main.html in this example.

Reading small files is easy. Let's explore looking at larger ones.

Reading Larger Files

The cat command reads small files nicely. But as you learned in Redirecting Streams of Text, on page ?, some files are too large to read with cat because they scroll off the screen, so you can use the more command to display a file one page at a time. Let's review that command to explore the contents of the system log, the log that holds messages from various applications and operating system processes.

On Ubuntu, you'll find this in /var/log/syslog:

```
$ more /var/log/syslog
```

On macOS, the system log is located at /var/log/system.log instead:

```
$ more /var/log/system.log
```

The first page of the file will display on the screen. Press the Enter key to see the next line of the file, and the Spacebar key to jump to the next page. Press q to quit, or page to the end of the file to return to your prompt.

The more command is a legacy program designed to read a file forward only, with no way to go backward. That's why you have the newer less command.

Less Is More

The less program was introduced to overcome some of the limitations more comes with. With less, you can navigate through the file using arrow keys and even perform some searches. For example, pressing / and typing in a search term followed by the Enter key jumps to the first occurrence of the search term and highlights the other entries.

On many systems, the more command is simply an alias of the less command, which causes some confusion. If you don't see any difference between these programs on your system, then that's probably what's going on.

less has many more features, and as a result it's also a lot bigger. Some smaller Linux distributions don't include it at all, so it's good to know the differences between these commands. But for daily use, your OS includes the less command, so you should be comfortable using that.

Both less and more are handy ways to read through a large amount of text, but sometimes you don't need to see the whole file. Sometimes you only want to look at the beginning or the end.

Reading the Beginning and End of a File

Sometimes the most interesting information in a file is in the first few lines or in the last few lines. That's where the head and tail commands come in handy.

The head command reads the first ten lines from a file and displays them to the screen:

```
$ head /var/log/syslog
```

If you want a different number of lines from the file, use the -n argument to specify the number of lines you want to see. To grab the first line in the /var/log/syslog file, use this command:

```
$ head -n 1 /var/log/syslog
```

The tail command displays the *last* ten lines from a file by default.

```
$ tail /var/log/syslog
```

Like head, tail also supports specifying the number of lines you want to view by using the -n switch. However, with tail, the count starts from the end of the file. Try it out with the names.txt file you created in Writing Multiple Lines to a File, on page 7. Use the -n switch to show only the last two names in the file:

```
$ tail -n 2 names.txt
Lisa
Maggie
```

If you specify the number with a plus sign, tail will start counting from that line and display everything until the end of the file. Give it a try. Read the names.txt file but don't display the first two lines. Instead, tell tail to start with the third line:

```
$ tail -n +3 names.txt
Bart
Lisa
Maggie
```

The first two lines are skipped. This is also a handy way of removing lines from a file or splitting a file. Use the > symbol to send the result to a new file instead of to the screen:

```
$ tail -n +3 names.txt > children.txt
$ cat children.txt
Bart
Lisa
Maggie
```

tail has another powerful feature that comes in handy when you're debugging things. You can use tail to "follow" a file and see its changes on the screen. Give it a try:

```
$ tail -f /var/log/syslog
```

You'll see the last few lines of the file displayed on your screen, but you won't be returned to your prompt. As changes happen in the file, your screen will update, displaying the new lines. This is incredibly helpful when viewing the logs for an application or other process in real time.

Press Ctrl+c to stop watching the file and return to your prompt.

You'll work with less, head, and tail again in Creating Pipelines of Data, on page ?. But let's shift focus and look at creating files outside of the home directory.