Now that you can fluidly use your terminal to bound between directories all over your computer I'll show you some actions you can perform on folders and files. One of the first actions you'll probably want to take when opening up a fresh terminal is to create a new folder or file. You can **m**a**k**e a **dir**ectory with the mkdir command, followed by the path to the new directory. First let's look at the contents of my home directory:

```
cd
ls
## Desktop
## Documents
## Photos
## Music
## todo.txt
```

I want to create a new directory to store some code files I'm going to write later, so I'll use mkdir to create a new directory called Code:

```
mkdir Code
ls
## Desktop
## Documents
## Photos
## Music
## todo.txt
## Code
```

It worked! Notice that the argument Code to mkdir is a relative path, however I could have specified an absolute path. In general you should expect Unix tools that take paths as arguments to accept both relative and absolute paths.

There are a few different ways to create a new file on the command line. The most simple way to create a blank file is to use the touch command, followed by the path to the file you want to create. In this example I'm going to create a new journal entry using touch:

```
touch journal-2017-01-24.txt
ls
## Desktop
## Documents
## Photos
## Music
## todo.txt
## Code
## journal-2017-01-24.txt
```

A new file has been created! I've been using ls to list the files and folders in the current directory, but using ls alone doesn't differentiate between which of the listed items are folders and which are files. Thankfully you can use the -loption with ls in order to get a **l**ong listing of files in a directory.

```
ls -l
## drwxr-xr-x  2 sean  staff  68 Jan 24 12:31 Code
## drwxr-xr-x  2 sean  staff  94 Jan 20 12:44 Desktop
## drwxr-xr-x  2 sean  staff  24 Jan 20 12:44 Documents
## drwxr-xr-x  2 sean  staff  68 Jan 20 12:36 Music
## drwxr-xr-x  2 sean  staff  68 Jan 20 12:35 Photos
## -rw-r--r--  1 sean  staff  90 Jan 24 11:33 journal-2017-01-24.txt
## -rw-r--r--  1 sean  staff  70 Jan 24 10:58 todo.txt
```

There is a row in the resulting table for each file or folder. If the entry in the first column is a d, then the row in the table corresponds to a **d**irectory, otherwise the information in the row corresponds to a file. As you can see in my home directory there are five directories and two files. The string of characters following the d in the case of a directory or following the first - in the case of a file represent the permissions for that file or directory. We'll cover permissions in a later section. The columns of this table also show who created the file, the group that the creator of the file belongs to (we'll cover groups later when we cover permissions), the size of the file, the time and date when the file was last modified, and then finally the name of the file.

Now that we've created a file there are a few different ways that we can inspect and edit this file. First let's use the wccommand to view the **w**ord **c**ount and other information about the file:

```
wc todo.txt
##       3      14      70 todo.txt
```

The wc command displays the number of lines in a file followed by the number of words and then the number of characters. Since this file looks pretty small (only three lines) let's try printing it to the console using the cat command.

```
cat todo.txt
## - email Jeff
## - write letter to Aunt Marie
## - get groceries for Shabbat
```

The cat command is often used to print text files to the terminal, despite the fact that it's really meant to con**cat**enate files. You can see this concatenation in action in the following example:

```
cat todo.txt todo.txt
## - email Jeff
## - write letter to Aunt Marie
## - get groceries for Shabbat
## - email Jeff
## - write letter to Aunt Marie
## - get groceries for Shabbat
```

The cat command will combine every text file that is provided as an argument.

Let's take a look at how we could view a larger file. There's a file inside the Documents directory:

```
ls Documents
## a-tale-of-two-cities.txt
```

Let's examine this file to see if it's reasonable to read it with cat:

```
wc Documents/a-tale-of-two-cities.txt
##      17    1005    5799 Documents/a-tale-of-two-cities.txt
```

Wow, over 1000 words! If we use cat on this file it's liable to take up our entire terminal. Instead of using cat for this large file we should use less, which is a program designed for viewing multi-page files. Let's try using less:

```
less Documents/a-tale-of-two-cities.txt
I. The Period

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,
it was the epoch of belief,
it was the epoch of incredulity,
it was the season of Light,
it was the season of Darkness,
it was the spring of hope,
it was the winter of despair,
we had everything before us, we had nothing before us, we were all going direct
Documents/a-tale-of-two-cities.txt
```

ou can scroll up and down the file line-by-line using the up and down arrow keys, and if you want to scroll faster you can use the spacebar to go to the next page and the b key to go to the previous page. In order to quit less and go back to the prompt press the q key.

As you can see the less program is a kind of Unix tool with behavior that we haven't seen before because it "takes over" your terminal. There are a few programs like this that we'll discuss throughout this book.

There are also two easy to remember programs for glimpsing the beginning or end of a text file: head and tail. Let's quickly use head and tail on a-tale-of-two-cities.txt:

```
head Documents/a-tale-of-two-cities.txt
## I. The Period
##
## It was the best of times,
## it was the worst of times,
## it was the age of wisdom,
## it was the age of foolishness,
## it was the epoch of belief,
## it was the epoch of incredulity,
## it was the season of Light,
## it was the season of Darkness,
```

As you can see head prints the first ten lines of the file to the terminal. You can specify the number of lines printed with the -n option followed by the number of lines you'd like to see:

```
head -n 4 Documents/a-tale-of-two-cities.txt
## I. The Period
##
## It was the best of times,
## it was the worst of times,
```

The tail program works exactly the same way:

```
tail Documents/a-tale-of-two-cities.txt

of an atrocious murderer, and to-morrow of a wretched pilferer who had robbed a
farmer’s boy of sixpence.
All these things, and a thousand like them, came to pass in and close upon the
dear old year one thousand seven hundred and seventy-five. Environed by them,
while the Woodman and the Farmer worked unheeded, those two of the large jaws,
and those other two of the plain and the fair faces, trod with stir enough,
and carried their divine rights with a high hand. Thus did the year one
thousand seven hundred and seventy-five conduct their Greatnesses, and myriads
of small creatures—the creatures of this chronicle among the rest—along the
roads that lay before them.
```

We’ve now gone over a few tools for inspecting files, folders, and their contents including ls, wc, cat, less, head, and tail. Before the end of this section we should discuss a few more techniques for creating and also editing files. One easy way to create a file is using **output redirection**. Output redirection stores text that would be normally printed to the command line in a text file. You can use output redirection by typing the greater-than sign (>) at the end of a command followed by the name of the new file that will contain the output from the proceeding command. Let’s try an example using echo:

```
echo "I'm in the terminal."
## I'm in the terminal.
echo "I'm in the file." > echo-out.txt
```

Only the first command printed output to the terminal. Let’s see if the second command worked:

```
ls
## Desktop
## Documents
## Photos
## Music
## todo.txt
## Code
## journal-2017-01-24.txt
## echo-out.txt
cat echo-out.txt
## I'm in the file.
```

Looks like it worked! You can also **append** text to the end of a file using two greater-than signs (>>). Let’s try this feature out:

```
echo "I have been appended." >> echo-out.txt
cat echo-out.txt
## I'm in the file.
## I have been appended.
```

Now for a **word of warning**. Imagine that I want to append another line to the end of echo-out.txt, so typed echo "A third line." > echo-out.txt into the terminal when really I meant to type echo "A third line." >> echo-out.txt(notice I used > when I meant to use >>). Let’s see what happens:

```
echo "A third line." > echo-out.txt
cat echo-out.txt
## A third line.
```

Unfortunately I have unintentionally overwritten what was already contained in echo-out.txt. There’s no undo button in Unix so I’ll have to live with this mistake. This is the first of several lessons demonstrating the damage that you should try to avoid inflicting with Unix. Make sure to take extra care when executing commands that can modify or delete a file, a typo in the command can be potentially devastating. Thankfully there are a few strategies for protecting yourself from mistakes, including managing permissions for files, and tracking versions of your files with Git, which we will discuss thoroughly in a later chapter.

Finally we should discuss how to edit text files. There are several file editors that are available for your terminal including vim and emacs. Entire books have been written about how to use both of these text editors, and if you're interested in one of them you should look for resources online about how to use them. The one text editor we will discuss using is called nano. Just like less, nano uses your entire terminal window. Let's edit todo.txt using nano:

```
nano todo.txt
  GNU nano 2.0.6                File: todo.txt

- email Jeff
- write letter to Aunt Marie
- get groceries for Shabbat



^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To
  Spell
```

Once you've started nano you can start editing the text file. The top line of the nano editor shows the file you're currently working on, and the bottom two lines show a few commands that you can use in nano. The carrot character (^) represents the Control key on your keyboard, so you can for example type Control + O in order to save the changes you've made to the text file, or Control + X in order to exit nano and go back to the prompt.

nano is a good editor for beginners because it works similarly to word processors you've used before. You can use the arrow keys in order to move your cursor around the file, and the rest of the keys on your keyboard work as expected. Let's add an item to my to-do list and then I'll save and exit nano by typing Control + O followed by Control + X.

```
  GNU nano 2.0.6                File: todo.txt

- email Jeff
- write letter to Aunt Marie
- get groceries for Shabbat
- write final section of "command line basics"


^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To
  Spell
```

Now let's quickly check if those changes were saved correctly:

```
cat todo.txt
## - email Jeff
## - write letter to Aunt Marie
## - get groceries for Shabbat
## - write final section of "command line basics"
```

You can also create new text files with nano. Instead of using an existing path to a file as the argument to nano, use a path to a file that does not yet exist and then save your changes to that file.

Summary

- Use mkdir to create new directories.
- The touch command creates empty files.
- You can use > to redirect the output of a command into a file.
- >> will append command output to the end of a file.
- Print a text file to the command line using cat.
- Inspect properties of a text file with wc.
- Peak at the beginning and end of a text file with headand tail.
- Scroll through a large text file with less.
- nano is simple text editor.