‹ (http://swcarpentry.github.io/shell-novice/02-filedir/)

Working With Files and Directories

› (http://swcarpentry.github.io/shell-novice/04-pipefilter/)

Overview

Teaching: 15 min
Exercises: 0 min
Questions

- How can I create, copy, and delete files and directories?
- How can I edit files?

Objectives

- Create a directory hierarchy that matches a given diagram.
- Create files in that hierarchy using an editor or by copying and renaming existing files.
- Display the contents of a directory using the command line.
- Delete specified files and/or directories.

We now know how to explore files and directories, but how do we create them in the first place? Let's go back to our `data-shell` directory on the Desktop and use `ls -F` to see what it contains:

```
$ pwd
```

```
/Users/nelle/Desktop/data-shell
```

```
$ ls -F
```

```
creatures/  molecules/          pizza.cfg
data/       north-pacific-gyre/ solar.pdf
Desktop/    notes.txt           writing/
```

Let's create a new directory called `thesis` using the command `mkdir thesis` (which has no output):

```
$ mkdir thesis
```

As you might guess from its name, `mkdir` means "make directory". Since `thesis` is a relative path (i.e., doesn't have a leading slash), the new directory is created in the current working directory:

```
$ ls -F
```

```
creatures/  north-pacific-gyre/  thesis/
data/       notes.txt            writing/
Desktop/    pizza.cfg
molecules/  solar.pdf
```

📌 Two ways of doing the same thing

Using the shell to create a directory is no different than using a file explorer. If you open the current directory using your operating system's graphical file explorer, the `thesis` directory will appear there too. While they are two different ways of interacting with the files, the files and directories themselves are the same.

📌 Good names for files and directories

Complicated names of files and directories can make your life very painful when working on the command line. Here we provide a few useful tips for the names of your files from now on.

1. Don't use whitespaces.

   White spaces can make a name more meaningful but since whitespace is used to break arguments on the command line is better to avoid them on name of files and directories. You can use `-` or _ instead of whitespace.
2. Don't begin the name with `-`.

   Commands treat names starting with `-` as options.
3. Stay with letters, numbers, `.`, `-` and _.

   May of the others characters have an special meaning on the command line that we will learn during this lesson. Some will only make your command not work at all but for some of them you can even lose some data.

If you need to refer to names of files or directories that have whitespace or another non-alphanumeric character you should put quotes around the name.

However, there's nothing in it yet:

```
$ ls -F thesis
```

Let's change our working directory to `thesis` using `cd`, then run a text editor called Nano to create a file called `draft.txt`:

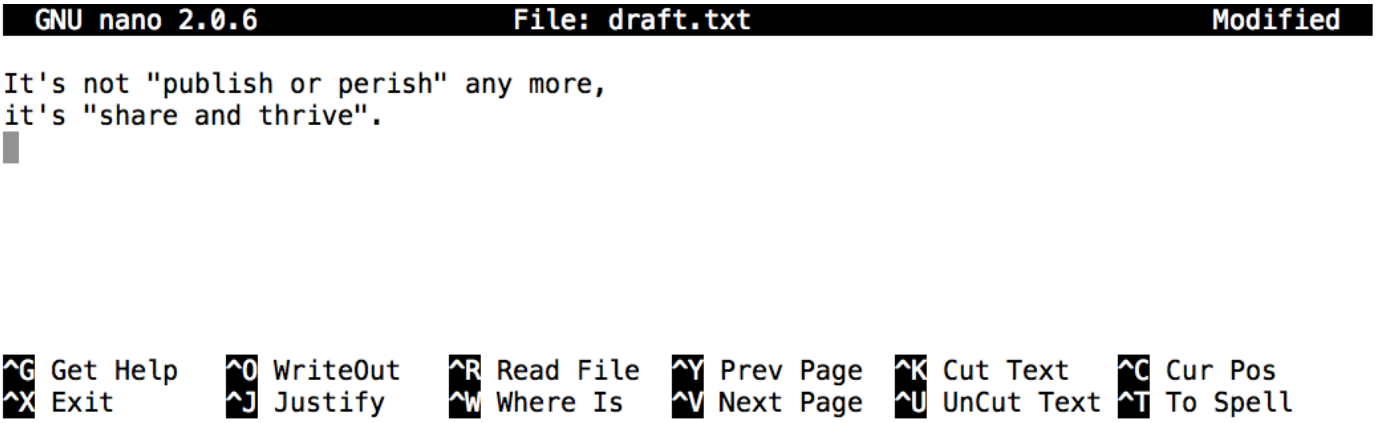
```
$ cd thesis
$ nano draft.txt
```

📌 Which Editor?

When we say, "`nano` is a text editor," we really do mean "text": it can only work with plain character data, not tables, images, or any other human-friendly media. We use it in examples because almost anyone can drive it anywhere without training, but please use something more powerful for real work. On Unix systems (such as Linux and Mac OS X), many programmers use Emacs (http://www.gnu.org/software/emacs/) or Vim (http://www.vim.org/) (both of which are completely unintuitive, even by Unix standards), or a graphical editor such as Gedit (http://projects.gnome.org/gedit/). On Windows, you may wish to use Notepad++ (http://notepad-plus-plus.org/). Windows also has a built-in editor called `notepad` that can be run from the command line in the same way as `nano` for the purposes of this lesson.

No matter what editor you use, you will need to know where it searches for and saves files. If you start it from the shell, it will (probably) use your current working directory as its default location. If you use your computer's start menu, it may want to save files in your desktop or documents directory instead. You can change this by navigating to another directory the first time you "Save As…"

Let's type in a few lines of text. Once we're happy with our text, we can press `Ctrl-O` (press the Ctrl or Control key and, while holding it down, press the O key) to write our data to disk (we'll be asked what file we want to save this to: press Return to accept the suggested default of `draft.txt`).



Once our file is saved, we can use `Ctrl-X` to quit the editor and return to the shell.

📌 Control, Ctrl, or ^ Key

The Control key is also called the "Ctrl" key. There are various ways in which using the Control key may be described. For example, you may see an instruction to press the Control key and, while holding it down, press the X key, described as any of:

- `Control-X`
- `Control+X`
- `Ctrl-X`
- `Ctrl+X`
- `^X`

In nano, along the bottom of the screen you'll see `^G Get Help ^O WriteOut`. This means that you can use `Control-G` to get help and `Control-O` to save your file.

`nano` doesn't leave any output on the screen after it exits, but `ls` now shows that we have created a file called `draft.txt`:

```
$ ls
```

```
draft.txt
```

Let's tidy up by running `rm draft.txt`:

```
$ rm draft.txt
```

This command removes files (`rm` is short for "remove"). If we run `ls` again, its output is empty once more, which tells us that our file is gone:

```
$ ls
```

📌 Deleting Is Forever

The Unix shell doesn't have a trash bin that we can recover deleted files from (though most graphical interfaces to Unix do). Instead, when we delete files, they are unhooked from the file system so that their storage space on disk can be recycled. Tools for finding and recovering deleted files do exist, but there's no guarantee they'll work in any particular situation, since the computer may recycle the file's disk space right away.

Let's re-create that file and then move up one directory to `/Users/nelle/Desktop/data-shell` using `cd ..`:

```
$ pwd
```

```
/Users/nelle/Desktop/data-shell/thesis
```

```
$ nano draft.txt
$ ls
```

```
draft.txt
```

```
$ cd ..
```

If we try to remove the entire `thesis` directory using `rm thesis`, we get an error message:

```
$ rm thesis
```

```
rm: cannot remove `thesis': Is a directory
```

This happens because `rm` by default only works on files, not directories.

To really get rid of `thesis` we must also delete the file `draft.txt`. We can do this with the recursive (https://en.wikipedia.org/wiki/Recursion) option for `rm`:

```
$ rm -r thesis
```

📌 With Great Power Comes Great Responsibility

Removing the files in a directory recursively can be very dangerous operation. If we're concerned about what we might be deleting we can add the "interactive" flag `-i` to `rm` which will ask us for confirmation before each step

```
$ rm -r -i thesis
rm: descend into directory ‘thesis’? y
rm: remove regular file ‘thesis/draft.txt’? y
rm: remove directory ‘thesis’? y
```

This removes everything in the directory, then the directory itself, asking at each step for you to confirm the deletion.

Let's create that directory and file one more time. (Note that this time we're running `nano` with the path `thesis/draft.txt`, rather than going into the `thesis` directory and running `nano` on `draft.txt` there.)

```
$ pwd
```

```
/Users/nelle/Desktop/data-shell
```

```
$ mkdir thesis
$ nano thesis/draft.txt
$ ls thesis
```

```
draft.txt
```

`draft.txt` isn't a particularly informative name, so let's change the file's name using `mv`, which is short for "move":

```
$ mv thesis/draft.txt thesis/quotes.txt
```

The first parameter tells `mv` what we're "moving", while the second is where it's to go. In this case, we're moving `thesis/draft.txt` to `thesis/quotes.txt`, which has the same effect as renaming the file. Sure enough, `ls` shows us that `thesis` now contains one file called `quotes.txt`:

```
$ ls thesis
```

```
quotes.txt
```

One has to be careful when specifying the target file name, since `mv` will silently overwrite any existing file with the same name, which could lead to data loss. An additional flag, `mv -i` (or `mv --interactive`), can be used to make `mv` ask you for confirmation before overwriting.

Just for the sake of inconsistency, `mv` also works on directories — there is no separate `mvdir` command.

Let's move `quotes.txt` into the current working directory. We use `mv` once again, but this time we'll just use the name of a directory as the second parameter to tell `mv` that we want to keep the filename, but put the file somewhere new. (This is why the command is called "move".) In this case, the directory name we use is the special directory name `.` that we mentioned earlier.

```
$ mv thesis/quotes.txt .
```

The effect is to move the file from the directory it was in to the current working directory. `ls` now shows us that `thesis` is empty:

```
$ ls thesis
```

Further, `ls` with a filename or directory name as a parameter only lists that file or directory. We can use this to see that `quotes.txt` is still in our current directory:

```
$ ls quotes.txt
```

```
quotes.txt
```

The `cp` command works very much like `mv`, except it copies a file instead of moving it. We can check that it did the right thing using `ls` with two paths as parameters — like most Unix commands, `ls` can be given thousands of paths at once:

```
$ cp quotes.txt thesis/quotations.txt
$ ls quotes.txt thesis/quotations.txt
```

```
quotes.txt   thesis/quotations.txt
```

To prove that we made a copy, let's delete the `quotes.txt` file in the current directory and then run that same `ls` again.

```
$ rm quotes.txt
$ ls quotes.txt thesis/quotations.txt
```

```
ls: cannot access quotes.txt: No such file or directory
thesis/quotations.txt
```

This time it tells us that it can't find `quotes.txt` in the current directory, but it does find the copy in `thesis` that we didn't delete.

📌 What's In A Name?

You may have noticed that all of Nelle's files' names are "something dot something", and in this part of the lesson, we always used the extension `.txt`. This is just a convention: we can call a file `mythesis` or almost anything else we want. However, most people use two-part names most of the time to help them (and their programs) tell different kinds of files apart. The second part of such a name is called the **filename extension**, and indicates what type of data the file holds: `.txt` signals a plain text file, `.pdf` indicates a PDF document, `.cfg` is a configuration file full of parameters for some program or other, `.png` is a PNG image, and so on.

This is just a convention, albeit an important one. Files contain bytes: it's up to us and our programs to interpret those bytes according to the rules for plain text files, PDF documents, configuration files, images, and so on.

Naming a PNG image of a whale as `whale.mp3` doesn't somehow magically turn it into a recording of whalesong, though it *might* cause the operating system to try to open it with a music player when someone double-clicks it.

🖉 Renaming Files

🖉 Moving and Copying

🖉 Organizing Directories and Files

🖉 Copy with Multiple Filenames

🖉 Listing Recursively and By Time

Creating Files a Different Way

Moving to the Current Folder

Using `rm` Safely

Using `rm` Safely

Copy a folder structure sans files

Key Points

- `cp old new` copies a file.
- `mkdir path` creates a new directory.
- `mv old new` moves (renames) a file or directory.
- `rm path` removes (deletes) a file.
- `rmdir path` removes (deletes) an empty directory.
- Use of the Control key may be described in many ways, including `Ctrl-X`, `Control-X`, and `^X`.
- The shell does not have a trash bin: once something is deleted, it's really gone.
- Nano is a very simple text editor: please use something else for real work.
