Terminal Commands 101

Some Opening Remarks

Throughout this document, I will use font that looks like `this` for terminal commands, keyboard prompts (e.g. press `spacebar`), and text that you may see in the terminal window.

You should save the accompanying document `tutorial.txt` to your desktop so that you can complete one of the exercises without added headache.

Opening and Using Terminal

On Macs, the terminal is a pre-installed application that allows users to interact with their computer via command prompts. By default, it can be found in your Utilities folder. If you are not sure how to access it, conduct a spotlight search by pressing `Command-Spacebar` and then typing `terminal`. Opening terminal should yield a small window with a white background. You should have a cursor that allows you to type in this window. However, you cannot use this cursor to click; if you need to navigate the command line (e.g. to fix a typo), use your arrows.

Using terminal commands is simple: just type the appropriate (case-sensitive) syntax and then press `enter`!

Ten Basic Commands

1. **Command 1: `pwd`**

The command `pwd` stands for *present working directory* and typing it will return your current directory. ("Directory" and "folder" are mostly interchangeable terms.)

Exercise: type ***`pwd`***

Since you just opened terminal, the directory that is returned is known as your *home directory*. Your home directory is analogous to a base of operations: it holds all of your personal files, programs, and directories. By default, the name of your home directory is identical to the name your username; my home directory is `/Users/LoganArnold2`. Additionally, the tilde (`~`) is a shorthand that refers to your home directory.

A *path* is a description of a file's location. For example, `/Users/LoganArnold2/README` (equivalently, `~/README`) is a path to a file called `README` contained within my home directory. Notice that forward slashes (`/`) separate components of a path.

2. **Command 2: `ls`**

The command `ls` will *list* the contents of your present working directory.

Exercise: type ***`ls`***

You probably have folders named `Applications/`, `Desktop/`, and `Downloads/`, among other folders and maybe even some files or programs located in your home directory. Notice, too, that these directories are displayed with a `/` at the end of their name; the ending `/` differentiates between files and directories.

It is worth mentioning that `ls` does not, in fact, list all of the files in your present working directory, but only those ones that do not begin with a dot (`.`). Files beginning with `.` are known as *hidden files* and usually contain important program configuration information. They are hidden because you should not change them unless you are very familiar with Unix!

To list all files in your home directory, including those whose names begin with `.`, type `ls -a`. The `-a` after `ls` is an example of a *flag* and can be thought of as an argument or option for the command `ls` that changes

the behavior of the command. (Most command line arguments have a plethora of flags, but this tutorial will only scratch the surface of this topic.)

Exercise: type `ls -a`

3. **Command 3:** `cd`

Moving around directories is simple. The command `cd` stands for *change directory* and combining `cd` with a directory named `dir` will change your present working directory to `dir`.

Exercise: type `cd Desktop/`

Your present working directory should now be your Desktop. Feel free to verify this using either `pwd` or `ls`.

Unix has some shortcuts that are quite useful, especially when changing directories.

- When typing the name of a directory or file within your present working directory, pressing `tab` will auto-complete as much as possible. For example, in the previous exercise, typing `cd Desk` followed by `tab` would likely auto-complete to the desired `cd Desktop/`
- A single dot, `.`, refers to the current directory. Thus, typing `cd .` essentially does nothing because you are changing your directory to your current directory. This may not seem useful at first, but it can save a lot of typing in more advanced settings.
- Two dots, `..`, refers to the *parent directory*, the directory in which your current directory is located.

Lastly, it is worth mentioning that `cd` on its own (i.e. without specifying `dir`) will change the present working directory to the home directory. This can be helpful if you are lost within a hierarchy of sub-directories!

Exercise: Navigate to your home directory, then back to the Desktop - twice. You should be able to navigate from the Desktop to your home directory using (at least) two distinct commands!

4. **Command 4:** `mkdir`

So far, all of the directories you have seen and navigated through have been ones that were on your computer previously. Often, you might be interested in creating a new directory. The command `mkdir`, short for *make directory* followed by `dir` will create a new directory named `dir` (assuming a directory with this name did not exist previously).

Exercise: type `mkdir TerminalTutorial` *and change your working directory to* `TerminalTutorial/`

5. **Command 5:** `cp`

The `TerminalTutorial` directory is currently empty (verify with `ls`). The command `cp` is used to copy a file from one location to another. Specifically, `cp file dir` copies `file` to `dir`.

Exercise: type `cp ../tutorial.txt .`

This command copied the accompanying `tutorial.txt` document into your present working directory by specifying the *relative path* from your present working directory to the file `tutorial.txt`. (Recall that `..` refers to the parent directory, in this case `Desktop/`, and `.` refers to the current directory.) Since `cp` copies files, you still have a version saved on the desktop.

`cp` can also be used to copy the contents of an existing file to a new file; this can be useful if you want to create a backup of a particular file or if you want to make changes to a file while keeping a copy of the original. The syntax to accomplish this is `cp file1 file2` where `file1` and `file2` are the names of the original and new files, respectively.

Exercise: type `cp tutorial.txt tutorial.backup.txt` *followed by* `ls` *to verify that you created a copy*

6. **Command 6:** `mv`

Sometimes you may prefer to move or rename a file rather than create a copy. `mv` is used analogously to `cp` but it *moves* the contents of a file rather than copying them. Thus, `mv file dir` moves `file` to `dir`, while `mv file1 file2` renames `file1` to `file2`.

Exercise: Use `mkdir` to make a directory called `backups` and then type `mv tutorial.backup.txt backups`. Use `ls`, then change directories to `backups/` and use `ls` again to verify that `tutorial.backup.txt` has moved.

Exercise: Use `mv tutorial.backup.txt tutorial.txt` to revert the name of the file to its original

7. **Command 7: `cat`, `head`, and `tail`**

Terminal has some built-in text editors (e.g. `nano` and `vim`) that may be of interest to you as a "next step" after completing this tutorial. (Alternatively, it is possible to open some other popular text editors in terminal, such as `Atom`.[1]) For simplicity, however, suppose you are not interested in editing a document but rather merely want to read its contents. This is precisely the purpose of `cat`, short for *concatenate*: `cat file` displays the contents of `file` in the terminal.

Exercise: type `cat tutorial.txt`

While `cat file` displays the entirety of `file`, sometimes it is more convenient to look at only the beginning or end of `file`; this can be done by the appropriately named commands `head` and `tail`, respectively. By default, each of these commands displays the first (or last) 10 lines of `file`. The flag `-n` followed by a number will display the first (or last) number of lines of `file`. (Thus `head file` and `head -n 10 file` produce identical results.)

Exercise: type `head tutorial.txt` and `tail tutorial.txt`

Exercise: use `head` and `tail` along with `-n` to display the first and last line of `tutorial.txt`

It is worth noting that `cat`, `head`, and `tail` are simple commands that will not yield useful results when used on many file formats, e.g. `.pdf`.

8. **Command 8: `grep`**

The command `cat` can quickly fill the terminal with text if you use it on a large file. Instead of reading through an entire file, suppose you are instead interested in only a subset of it, but that the subset is not merely the beginning or ending of the file. For instance, `tutorial.txt` contains a few lines of sample data that mimic a data file that you may come across in your research. Perhaps you are only interested in the values from the month of January. The command `grep` can be used to search text files. `grep expression file` searches `file` for `expression` and displays the contents of all of the lines in `file` that contain `expression`. (`grep` is short for *globally search a regular expression*[2] *and print.*) Note that by default `grep` is case-sensitive; the flag `-i` performs case insensitive matching.

Exercise: type `grep Jan tutorial.txt`

`grep` can also be used to search for phrases; use single or double quotes (`'` or `"`) to enclose the phrase.

Exercise: type `grep Hello tutorial.txt` and `grep 'Hello, world.' tutorial.txt` and compare the results

9. **Command 9: `rm` and `rmdir`**

Deleting files and directories is both useful and simple. `rm file` and `rmdir dir` deletes a file named `file` or a directory named `dir`, respectively. Use these commands with caution, however! It can be difficult or impossible to recover items deleted via these commands.

Exercise: type `rm tutorial.txt` then `ls` to verify that `tutorial.txt` has been deleted

[1] See http://tutorials.jumpstartlab.com/academy/workshops/terminal_and_editor.html for a short tutorial.

[2] You may wish to Google regular expressions to learn some more tools that enhance the functionality of `grep` and other command line arguments.

Exercise: Change directories to the parent directory, `TerminalTutorial/`, and then type `rmdir backups` to delete the directory of backups. Again, use `ls` to verify that the directory has been deleted.

10. **Command 10: `man`**

This tutorial has just scratched the surface of what can be done using terminal commands. If you forget what a particular command does, want to learn more about the flags that can be used for the command, or come across a new command altogether, the terminal offers a robust description of each command. `man command` opens up the *manual* for `command`.

Exercise: Use `man` to read more about one or more of the commands covered in this tutorial. Type `q` to exit the manual when finished.

Exercise: Use `man` to learn about a command not covered in this tutorial, such as `diff`, `sed`, or `gzip`

A Bonus Command: `clear`

By now, your terminal is likely filled with text. Although opening a new terminal window is an easy way to obtain a clean slate, your working directory in that new window will initially be your home directory. If you are buried in a hierarchy of sub-directories, it more convenient to use the command `clear` to remove the text in the terminal while maintaining your working directory.

Exercise: type `clear` to clean your terminal window, then `pwd` to verify that your current directory is `TerminalTutorial/`