Introduction to the Command Line or Shell

Before we start learning Python, it will be useful for us to begin with some command line practice.

The command line is an interface by which you can interact with the computer using commands rather than clicking and navigating with your mouse. While it may be uncomfortable and appears as if you are hacking a computer, you are simply issuing commands to the computer just as you would with a click.

Remember that there are several different languages used at the command line. We will be learning bash script, which is a standard language on Unix/Linux systems, like many of the servers you will need to control in this program, and on Macs.

Basic Navigation and Layout

Let's get started by opening up your terminal on a Mac or your git bash shell on Windows. Once you are in, you should see a screen start with `$`.

Now let's type

```
pwd
```

This stands for *print working directory*. A directory is like a folder and is your current location in the file system.

Now let's type

```
ls
```

This lists all of the files and folders in the directory. You will likely see a lot when you first run it. This is your computer's file system, where all programs and files and folders are stored on your computer.

Before we go any further, it is helpful to know where to look for help. We can do that with the `man` command.

```
man ls
```

This will give you instructions on what is available to you regarding that specific command. Once you have type `man`, at the bottom of your screen you should see a

```
:
```

This signifies that it is waiting for a command from you. Type `q`.

You will notice that everything is cleared off your screen; this happened because we quit the prompt. Let's open it up again. This time type `/-F`. That performs a search in the commands, similarly to typing `Cmd-F` or `Ctrl-F` when you are searching in a web page.

You can scroll with the arrow keys or by typing `f` or `b` at the `:` prompt.

Now type `h`. This is worthwhile to remember because it shows you all the help commands that you would need for navigating the manual of a given function. The `^` symbol indicates that you should press the `ctrl` key to activate this function. There are many types of optimizations, but this covers the majority of them.

Motivation

We have touched on a few commands such as `ls`,`pwd`, and `man`, and you may be wondering why we would use commands when it is much faster to use a mouse. Although that may be true with what we have covered so far, this is not the case when you are navigating a server in the cloud that does not have a graphical user interface (GUI).

This may be surprising. Not every computer has a GUI because these are quite expensive; the computer must use a lot of resources to have a GUI available, and many servers do not provide this kind of interface. With time, you will likely come to appreciate the command line. It allows you to do what you want in a precise way and makes it extremely easy to automate a set of instructions.

You should also be aware that you will need to use it a fair amount as a data scientist. The vast majority of your work will involve the command line in one way or another.

Basic Commands (cont.)

Let's look at some of the more basic commands. We will navigate to the desktop and create a text file.

```
cd ~/Desktop
```

`cd` stands for *change directory* and allows you to navigate the file system. This should switch us into the desktop, which we can check with

```
pwd
```

This should print that we are on the desktop. Now we can type our next commands.

```
mkdir test_creation
touch my_file.txt
echo "Hello From the Command Line" >> my_file.txt
mv my_file.txt test_creation
echo "This is my second echo command" >> test_creation/my_file.txt
cat test_creation/my_file.txt
```

Notice I do not use spaces very often. Spaces can become cumbersome at the command line and must be escaped with a \\ character. I never use spaces in my file names any more, and with time you likely won't either.

Now let's review what happened line by line. First we created a directory with the `mkdir` command. This is on our desktop, and we should be able to see it if we navigate there with our mouse. Next we created a text file on our desktop with the `touch` command, which creates a simple plain text file that is empty. Next we used the `echo` command to append a string to `my_file.txt` (the one we just created).

`echo` simply repeats what you send it. If you type echo "Hello there", then it will print "Hello there"; however, by using `>>` we append it to `my_file.txt`. This allows us to easily append more information, which we will do shortly.

Now we are doing something that we have not done before. By using the `mv` command, we move the direction from location 1 to location 2. This puts our `my_file.txt` into the folder that we created at the command line.

The next echo command acts similarly to the way it did before but echos the string into the file that we just moved into the folder. We can see the result of that in the next line.

`cat` is a way of reading the contents of a file. When we execute this command, it will print everything in the following file(s) that are passed to it as arguments. It should print

```
$ cat test_creation/my_file.txt

Hello From the Command Line
This is my second echo command
```

We could also change directories into our new `test_creation` directory since we will be doing more in there, and can print the contents from there.

```
$ cd text_creation
$ cat my_file.txt
```

```
Hello From the Command Line
This is my second echo command
```

What is powerful about cat is that it can read more than one file at once, concatenating one onto another. Let's take a look at how that works with the echo command.

```
echo "This is the second file" > second_file.txt
```

Notice that there is only one `>` in that code snippet. When we use only one `>`, we overwrite everything in the file.

```
cat second_file.txt
```

will print out what we just wrote in our echo statement. (Notice how it also created the file for us because it didn't exist.)

Now we can run

```
cat my_file.txt second_file.txt
```

and take it to the next level with

```
echo "Edited first file to be this" > my_file.txt
cat my_file.txt second_file.txt > all_contents.txt
cat all_contents.txt
```

Now let's make a copy of our all_contents.txt file back on our desktop. We would do this as follows. *Same issue with italic as above.*

```
cp contents.txt ../contents.txt
```

Notice that the new file has appeared on your desktop. Open it and check that the contents are the same.

The `cp` command is the copy command where we copy from one place to another. However, you will notice that we copied to a `../` directory. We did this because we are using a *relative* path as opposed to an *absolute* path.

Relative vs. Absolute Paths

These are two important but straightforward distinctions. A relative path is one relative to your current location. The command line knows which directory we are currently in, so when we wrote `cd test_creation` from the desktop, it knew that we were changing into a directory relative to our current one. An alternative that is sometimes required is an absolute path. This is what you will see printed when we run `pwd`. That gives us the entire path. When we were at the desktop, we could have equivalently used the absolute path, which would be something like `cd /users/user_name/Desktop/`; however, writing the whole path is cumbersome–thus, the relative path.

Now that we have covered the distinction between these two paths, let's return to `../`.

Basic Commands (cont.)

Let's go ahead and run in our shell

```
cd ..
ls .
ls test_creation
ls ..
```

This will give us some interesting output. You may have guessed that `..` is short for "up one directory." If we are in a folder contained in another folder, we can use `cd ..` to jump up one directory in the tree (like `test_creation` to Desktop).

`.` refers to the current directory, so `ls` is simply short for `ls .`. As you may have seen in the `ls` help, we can pass a directory to `ls` to list its contents. This allows us to list everything in `test_creation` with `ls test_creation` or in the directory above Desktop with `ls ..`.