#Introduction to the command line or shell

Before we start learning Python, it will be useful for us to begin with a little bit of 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 as you might expect. While it's scary and certainly appears like you're hacking a computer, you're really just issuing commands to the computer just as you would with a click.

This command line exercise will be oriented towards Unix and Linux systems. Not windows computers. Therefore you should be using something like git bash on your windows computer to launch the shell.

Basic Navigation and Layout

Let's get started by opening up your terminal on mac or your git bash shell on windows. Once you get in there you should get a screen with a `$` that starts. That's where the fun begins.

Now let's type

```
pwd
```

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

Now let's type:

```
ls
```

Now what this does is list all the files and folders in the directory. You'll likely see a lot when you first run it. This is your computers file system, where all programs and files and folders are on your computer.

Now before we go any further, it's really 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've hit man, you're in an interesting prompt, at the bottom of your screen you should see a

```
:
```

This signifies that it's waiting for a command from you. Try typing `q`.

Woah! See how that just cleared everything off your screen, that's because we `quit` the prompt. Let's open it up again and this time type `/-F`. That performs a search in the commands - like how you might type `Cmd-F` or `Ctrl-F` when you're 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 will definitely be your friend because it shows you all the help commands that you would ever need about navigating the manual of a given function. The ^ symbol means you should be hitting the `ctrl` key to activate this function. There are all sorts of optimizations here but that's the majority of them.

Motivation

Now at this point we've touched on a couple of commands like `ls`,`pwd`, and `man` and you're probably saying to yourself - "This is dumb, I can do this with my mouse, way faster." And while at this point in time that may be true, what happens when you're navigating a server in the cloud that doesn't have GUI or graphical user interface.

This was something that amazed me at first, but every computer doesn't have a graphic interface because they're actually fairly expensive computationally - the computer has to use up a lot of resources to have it be available and many servers just don't provide this kind of interface. I promise that with time, you will come to love the command line as I have. It's a fantastic way of doing what you want in a precise way and making it super easy to automate a set of instructions.

The other reason you should learn to like it is that you're going to have 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 - Con't

Let's show you some of the more basic commands, what we're going to do is navigate to the desktop and create a text file. So let's get started.

```
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
```

Which should print out 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
```

One of the things that you'll notice is that I don't use spaces very often. That's because they're a pain at the command line and have to be escaped with a \ character. I never use spaces in my file names any more and with time you likely won't either.

Now that we've covered that, let's go through line by line what happened. First we created a directory with the `mkdir` command. This is on our Desktop and we should be able to see it if we navigated there with our mouse. Next we created a text file, on our Desktop, with the `touch` command which just 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).

Now all echo does is repeat back what you send it, if you type in echo "Hello there" then it will just print out "Hello there", however by using `>>` we append it to `my_file.txt`. This allows us to easy append more information which we will do shortly.

Now we're doing something that we haven't done before my 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 does a similar thing to what we 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 out everything in the following file(s) that you pass 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'll be doing more things in there, and print the contents from there.

```
$ cd text_creation
$ cat my_file.txt

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

What's cool about cat is that it can actually read more than one file at once, concatenating one onto another. Let's show you how that works with the echo command.

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

Notice how there's 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 created the file for us too because it didn't exist.)

now what we can run is:

```
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.

```
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'll notice that we copied to a `../` directory. This is because we're using a *relative* path as opposed to a *absolute* path.

Relative vs. Absolute Paths

These are two pretty important distinctions but are straight-forward enough. A relative path is one relative to your current location, the command line knows what directory we're 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 (required at sometimes) is an absolute path. This is what gets 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 we'd obviously get tired of writing that whole path, thus the relative path.

Now that we've covered the distinction between these two, let's go ahead and return to `../`.

Basic Commands - Con't

Let's go ahead and run in our shell:

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

This will give us some interesting output. At this point if you haven't quite realized, `..` is short for up one directory. So if we've 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 actually just short for `ls .`. As you probably saw in the `ls` help, we can pass a directory to `ls` to list its contents. This is what allows us to list everything in test_creation with `ls test_creation` or in the directory above Desktop with `ls ..`.