Unix command line: convenient, flexible, universal

The pace is intentially fast. There are multiple very good tutorials available for you to review what we're going to learn today:
https://www.codecademy.com/learn/learn-the-command-line
http://swcarpentry.github.io/shell-novice/
http://korflab.ucdavis.edu/unix_and_perl/ (concentrate only on the unix shell tutorial)

Open a Terminal window: from the Finder -> Applications -> Utilities
The first thing you notice is that there's a little prompt that tells you something about who you are (username), the machine you're using, and the directory you're in, and ends with $. On my laptop it looks like this:

```
gadget:~ sarahjwheelan$
```

This prompt is extremely customizable! you can have it display the time, special symbols, quotes, colors, and more.

All commands that you'll type are designated with a black background to make them clearer:

```
$ ls
```

this shows you the files and folders in the directory that the terminal is seeing. Compare this with the folder that you can see in the graphical interface in Finder.

You can get a more informative list:

```
$ ls -l
drwxrwxr-x    5 sarahjwheelan  admin        170 Jun 30  2014 Applications
-rw-r--r--@   1 sarahjwheelan  staff    1198058 Jun 18  2014 DatasetS6.gtf
drwx------+ 192 sarahjwheelan  staff       6528 Apr  6 07:56 Desktop
drwx------+  36 sarahjwheelan  staff       1224 Apr  6 07:21 Documents
drwx------+  11 sarahjwheelan  staff        374 Apr  2 17:47 Downloads
drwxr-xr-x    4 sarahjwheelan  staff        136 Mar 18 15:43 GENE-E
drwx------+  57 sarahjwheelan  staff       1938 Apr  2 16:27 Library
-rw-r--r--    1 sarahjwheelan  staff       4470 Jan 26 22:46 MBC003_mets.pdf
```

And with another option you get human-readable file sizes.

```
$ ls -lh
drwxrwxr-x    5 sarahjwheelan  admin   170B Jun 30  2014 Applications
-rw-r--r--@   1 sarahjwheelan  staff   1.1M Jun 18  2014 DatasetS6.gtf
drwx------+ 192 sarahjwheelan  staff   6.4K Apr  6 07:56 Desktop
drwx------+  36 sarahjwheelan  staff   1.2K Apr  6 07:21 Documents
```

```
drwx------+  11 sarahjwheelan  staff   374B Apr  2 17:47 Downloads
drwxr-xr-x    4 sarahjwheelan  staff   136B Mar 18 15:43 GENE-E
drwx------+  57 sarahjwheelan  staff   1.9K Apr  2 16:27 Library
-rw-r--r--    1 sarahjwheelan  staff   4.4K Jan 26 22:46 MBC003_mets.pdf
```

what are those letters?
`drwxr-xr-x`
`-rwxr-xr-x`
`-rw-r--r--`

Each set of three fields specifies read, write, execute permissions for each type of user: owner, group, and world.
The format is `-OOOGGGWWW`
where the first character indicates the type of object (directories are 'd', regular files are '-', other types of links have other keys)
and the triples can be
`rwx`
`rw-`
`r-x`
`r--`

or other combinations of read, write, and execute; things that are writeable or executable by a class of users are generally also readable by that class, so `-w-`, for example, is very unusual. The examples below are written without the leading character.

```
r--r--r--  owner, group and world can read
rwx------  owner can read, write & execute, group and world have
no access at all
rwxr-xr-x  owner can read, write & execute, group and world can
read and execute but not write
rw-r--r--  owner can read and write, group and world can only
read
```

You can change these permissions, if you have permission to do so (e.g. you are the owner). The simplest way to do this is to use the `chmod` command and to understand the encoding for the different permissions.

Each triple can be thought of as three numbers, where 0 means no permission and 1 means permission:
`r-- is 100`
`rw- is 110`

so we can treat these as binary numbers, where 001 is 1, 010 is 2, 100 is 4, 110 is 6 etc.

so `$ chmod 755 filename`

will change the permissions for that file to `rwxr-xr-x`

and `$ chmod 644 filename`
is `rw-r--r--`

There's a shorthand as well; e.g. `$ chmod g+rw filename` adds read and write permissions to the group field.

On a Mac, you can see a file's permissions in the GUI if you highlight a file and choose 'Get Info' from the finder.

Note: the Mac filesystem uses access control lists (ACL), so there is often a character at the end of the 9-character permissions string. Don't worry about it; these are almost never important and it is easy to use these incorrectly.

Let's find a more interesting directory.
How about "Desktop"? In the finder window, double click on 'Desktop' to see what's there.

In the terminal window, we'll do the same thing. To change directories, use

```
$ cd Desktop
```

And look at the directory contents:

```
$ ls
```

It should be exactly the same as what you see in the Finder window -- all we're doing is using a text interface.

Now we can look at some files. Make sure that the CGmaterials directory is on your desktop and unzipped.

```
$ cd ~/Desktop/CGmaterials
```

Notice that we typed the path with a ~ at the beginning. The ~ is shorthand for "my directory." The Unix architecture is inherently a shared filesystem. To see the full path to this folder, now that you're in it, use 'pwd' (pathname to current working directory)

```
$ pwd
/Users/sarahjwheelan/Desktop/CGmaterials
```

pwd is a great command and will help orient you if you start getting lost.

This is an absolute path. From anywhere in the filesystem you can use this command and you'll get to this directory.

What other users are on this computer?

```
$ ls /Users
```

Now let's see what files and directories are inside CGmaterials.

```
$ ls -l
```

There should be a directory named chip. You can get into this directory:

```
$ cd /Users/sarahjwheelan/Desktop/CGmaterials/chip
```

but now go back up one directory to where you started:

```
$ cd ..
```

The shorthand for "directory above this one" is simply ".." so this will get you back to /Users/Shared/CGmaterials/.

Now just try this:

```
$ cd chip
```

Note the lack of the preceding / in this command. This means that you are using a relative path. This is a lot shorter to type. You can see what's in this directory without even changing directories, though!

We can also look at what's in the chip directory, or in other directories, without actually going there.

```
$ cd ..
$ ls chip
```

Now, make sure you're in the materialsCG directory. Go to the Finder window and confirm that what you're seeing in the GUI is what you see in the Terminal window.

```
$ ls
$ cd chip
$ ls
```

There's a file of ChIPseq results -- this was an experiment using an antibody to a histone mark. After alignment, we used the program, MACS, to delineate regions of enrichment in the IP sample. Those regions are in bed format: `chip_peaks.bed`

This is a plain tab-delimited text file and we could open it in any text editor if we wanted, or even in a spreadsheet program, though we would probably have to change the suffix.

But this is easier:

```
$ less chip_peaks.bed
```

This puts you into an interactive environment in which you can read the file and search for text strings. Hit the space bar to page forward, and the letter 'q' to quit. Try searching for the string '`peak_92694`' by typing a forward slash, and then the string. The program will jump to that position in the file and will highlight the text. If it can't find your search pattern, it will tell you.

The '`less`' command does a lot more than this! I could try searching on the web to see what it does, but good luck finding information about '`less`' through google :)

There's a better way. Unix/Linux environments have extensive built-in help manuals for most of the native programs. To access the help manual for '`less`', type

```
$ man less
```

and you can read A LOT about the command. This is a really powerful utility. All manual pages appear in a reader that uses the '`less`' architecture, so you can page forward with the space bar or `Control-F`, backward using `Control-B`, and quit using 'q'.

You can save yourself some typing and take advantage of tab completion:

```
$ less chi
```

and then hit the tab key. If there is an unambiguous way to complete that filename, it will do it, and you can hit the return key and the file opens in 'less'. If there isn't an unambiguous completion, it will show you the possible completions. This works for any command, not just 'less'. This not only saves typing but will reduce errors, especially if you're doing a lot of copying and renaming.

Editing files with nano

There are a lot of command line text editors; vi and vim are some of the older ones, emacs is newer and quite powerful, and nano and pico are lightweight and easy to learn. emacs is a good one to learn, as it has integrated support for many programming languages. We will start with nano, as it's very simple.

```
$ nano
```

nano is great -- just start typing! This is a keyboard-controlled editor, and nearly all of the commands start with `Control-` (this is abbreviated with a carat `^`, for example, `^K` means `Control-K`)

nano has a buffer of the most recent changes, so you can cut and uncut text as much as you want. Type `^G` to read about the multiple options in this editor.

When you've finished typing, save this file. Type `^X` to exit, and it will ask whether you want to save what you've typed. Type 'y' and save it as `testfile.txt`.

Look at your new file:

```
$ less testfile.txt
```

You can edit it, again using nano.

For the time being we'll see what happens when we rename files, duplicate them, and move them into and out of directories.

```
$ cp testfile.txt testfile2.txt
$ ls
```
now there are two identical files.

```
$ mv testfile.txt testfile3.txt
$ ls
```
now we still have testfile2.txt but the 'mv' (move) command has renamed testfile.txt.

Let's make a new directory, and copy a file into it.

```
$ mkdir testdirectory
$ ls
$ cp testfile2.txt testdirectory/testfile.txt
```

(use tab completion here!)

```
$ ls testdirectory
```

Things look good! there's the file. What if I did this:

```
$ cp chip_peaks.bed testdirectory/testfile.txt
```

what's in the file now?

```
$ cd testdirectory
$ less testfile.txt
```

OR just try this:

```
$ less testdirectory/testfile.txt
```

I have clobbered my file! Whatever was in it has been irreversibly overwritten with the contents of chip_peaks.bed.

Similarly, when I remove a file, for example,

```
$ rm testfile2.txt
```

the file is gone. There is no 'undo,' there is no 'find it in the trash and get it back.' Some people reassign their 'rm' command to move the file to the trash folder, but you can still copy files onto each other and you won't be able to get them back. Having a constant backup will help, and using tab completion will help (this will tell you whether there is a file with the same name, in your target folder). However, the rule is this:

Unix commands assume that you know what you're doing. There is no undo. Commands, especially those that create or move files, will overwrite files if files with those names already exist.

Let's look more closely at the ChIPseq output file.
We can look at the first 10 lines very easily with 'head'

```
$ head chip_peaks.bed
```

There are 5 columns: chromosome, start, stop, peak ID, and -log10(p-value).

There is a 'tail' command, too, and you can probably guess what it does.

How many entries are there?

```
$ wc chip_peaks.bed
   35921  179605 1707524 chip_peaks.bed
```

The output from wc is the count of lines, words, and characters (including line breaks) in the file. If we just want the number of lines we can do this:

```
$ wc -l chip_peaks.bed
   35921 chip_peaks.bed
```

This is something you will use often, as it takes a lot less time than getting the word and character counts on a big file, and you don't often need those values anyway.

This file is a little hard to look at because the fields are tab-delimited but aren't all the same width, so the columns are offset. We can use another little utility, `column`, to see the file in a more organized format:

```
$ column -t chip_peaks.bed
```

This is looking better. The -t flag tells the `column` utility to show the file as a table. Check out the manual page for more information.

The whole file just whizzed by, though. It would be nicer to be able to actually see it. We can pipe commands together, separated by the | symbol. This directs the output from one command into another. We'll use this to send the output of the `column` utility to `less`, so that we can page through the file after it has been formatted:

```
$ column -t chip_peaks.bed | less
```

Much better! we can now page forward and backward through the file and it will retain the formatting.

There are a lot of very powerful utilities for looking at files; these are typically a lot faster than any script that you can write, so it's worth learning them. The simplest is the 'sort' utility, which is extremely flexible, and fast. You can sort alphabetically or numerically, increasing or decreasing, ignoring case or not, and can sort by whichever column(s) you like, in whatever priority you like. The default is to sort by the first column, alphabetically.

```
$ sort chip_peaks.bed
```

Again, it would be nice to just look at the top few lines. Here, I will sort the file and instead of dumping the output directly to the screen, I will give it to the 'less' command:

```
$ sort chip_peaks.bed | less
```

I could also just show the first ten lines of the sorted file:

```
$ sort chip_peaks.bed | head
```

Notice how the chromosome coordinates (second column) are sorted alphanumerically -- 10001 comes before 101. We can fix this. Let's sort first by the chromosome name, alphabetically, and next by the coordinate of the start of the peak (column 2), numerically.

```
$ sort -k1,1 -k2,2n chip_peaks.bed | less
```

The last column has -log(p-values) -- the higher the number, the more confident the peak. I'd like to know what the largest and smallest p-values are for this data. One way to do this is to sort the values numerically and take a look.

Sort the file by increasing p-value. The p-value is the 5th column, so we'll tell the sort command to look only at the 5th column. If we gave it -k2,3 then it would sort on fields 2 and 3 from each line; giving it -k5,5 restricts the sorting to the 5th column.

```
$ sort -k5,5 chip_peaks.bed
```

OK, that's not helpful, as the default is an alphabetical sort.

try again with the 'n' for numeric:

```
$ sort -k5,5n chip_peaks.bed | less
```

by decreasing p-value -- use 'r' for reverse:

```
$ sort -k5,5nr chip_peaks.bed | less
```

by position on a chromosome (not terribly useful here) and this time use the 'head' function:

```
$ sort -k2,2 chip_peaks.bed | head
```

by chromsome name and then the p-value, in decreasing order:

```
$ sort -k1,1 -k5,5nr chip_peaks.bed | head
```

I might want to just look at the top scoring peak, and put this information into a file. I can do this!

the `head` utility can show as many rows as you like; just give it a flag, e.g. `head -12 chip_peaks.bed` will show the first 12 lines.

```
$ sort -k1,1 -k5,5nr chip_peaks.bed | head -1
```

and then use ">" to redirect the output into a new file:

```
$ sort -k1,1 -k5,5nr chip_peaks.bed | head -1 > bestpeak.txt
```

and look at the new file:

```
$ less bestpeak.txt
```

Be careful with the ">" command. As with the other unix utilities, it is not polite! it will overwrite a file if it already exists.

I can get the lowest scoring peak and add it to the file that I've already created (though 'bestpeak' is probably not the best name any more!). This uses the '>>' operator, which will create a file if it doesn't already exist, or will append to a file if the file does exist.

```
$ sort -k1,1 -k5,5n chip_peaks.bed | head -1 >> bestpeak.txt
```

and look at the file again:

```
$ less bestpeak.txt
```

If I want only a subset of these data, there are dozens of ways to go about it. One very common command is '`grep`'.

```
$ grep chr1 chip_peaks.bed
```
This will pull out any row that contains the text 'chr1'. Note that you also got a bunch of other chromosomes (chr10, chr11 etc) because they matched the string as well. If you really want the data from only chr1, you can tell grep to match it as a word, meaning that it will only be retrieved if it has whitespace on both sides (a space or tab). To do this, use the -w flag. Remember that you can always check the manual pages (`man grep`) to get information on the options available for each utility.

```
$ grep -w chr1 chip_peaks.bed | less
```

Let's put it in a new file, and then look at the beginning of the file.

```
$ grep -w chr1 chip_peaks.bed > chr1peaks.bed
$ head chr1peaks.bed
```

It's not sorted. We can sort it in a piped command, before we put it in the file. Note that this overwrites the previous file. I'll sort by chromosome coordinate, and I'll be sure to make this a numeric sort.

```
$ grep -w chr1 chip_peaks.bed | sort -k2,2n > chr1peaks.bed
$ head chr1peaks.bed
```

I don't need the peak IDs, and I'd like to clean up the table. There are a few ways to extract columns from text files. One of the simplest is the '`cut`' command. It assumes that your file is tab-delimited, though you can use the -d option to specify a different

delimiter. Despite the name, it does not alter your original file. The -f option specifies the column (field) that you want to extract from the file.

```
$ cut -f1 chr1peaks.bed
```

That's a little boring. Let's add some more columns. You can specify multiple fields, either separately or in a range:

```
$ cut -f 1 -f 2 chr1peaks.bed | less
$ cut -f 1-3 -f 5 chr1peaks.bed | less
```

We'll put the first three columns into a file:

```
$ cut -f 1-3 chr1peaks.bed > chr1_1_3.txt
```

and the last column in another file:

```
$ cut -f 5 chr1peaks.bed > chr1_5.txt
```

and we can use the 'paste' function to stick these together! In this case, we get the same result as an earlier 'cut' command, but the function is generally very useful.

```
$ paste chr1_1_3.txt chr1_5.txt > chr1_final_columns.txt
```

A much more powerful, but sometimes confusing, method for managing delimited files is the '`awk`' scripting language. In awk, the fields are numbered and designated with a $, and the command is set off with straight quotes and curly brackets, like this:

```
$ awk '{print $1}' chr1peaks.bed
$ awk '{print $1 "\t" $2}' chr1peaks.bed
```

In the second command there is a special character, \t, which denotes a tab -- since you can't type a space, tab, or newline in a command, these "escape" characters are used instead.

Finally, we'll stick two files together.

```
$ head chr1peaks.bed > chr1peakshead.txt
$ tail chr1peaks.bed > chr1peakstail.txt
$ cat chr1peakshead.txt chr1peakstail.txt
```

The `cat` command simply displays all the contents of the files passed to it. If you give `cat` only one filename, it will show you the contents of that file -- this is another of the dozens of methods you can use to look through files.

```
$ cat chr1peakshead.txt
```

You can use `cat` to create a new file, as well, by asking it to display the contents of two files and then using the redirect command to put that output into a new file.

```
$ cat chr1peakshead.txt chr1peakstail.txt > newcatfile.txt
```

Now you should have a few files in your directory. Type `ls` to list them. We can use a wildcard to list only the files that start with the characters, chr1:

```
$ ls chr1*
```

The * syntax indicates that you want to print anything that starts with chr1 and then has some (or no) trailing characters, no matter what they are.

Finally, a little script:

```
$ for i in `ls chr1*`; do wc -l $i; done
```

We can even put this in a file and run it:

```
$ nano listscript.sh
```

```
#!/bin/bash

for i in `ls chr1*`
do
   wc -l $i
done
```

Change the permissions for this file, to make it executable.

```
$ chmod 755 listscript.sh
```

And run your script!

```
$ sh listscript.sh
```

Exercises:

1. What are the permissions for newcatfile.txt? Change these permissions to read-only (by owner, group, and world). Now try to delete the file. What happens? Change the permissions back. Now delete the file. This is a simple way to make sure that you don't accidentally delete important files (for example, the raw data from your sequencing run). (hint: `chmod`)

2. What does the gzip utility do? (look at the manual page). Use gzip to compress one of the files in your directory. gzip is fantastically effective for biological sequences, as these files have low information content to begin with. Use gunzip to uncompress the

file. How much memory does the file occupy, uncompressed? how big is the compressed file? (hint: `ls -lh`)

3. You can use the old and very effective archive utility, `tar`, ('tape archive') to compress a directory into a single file. It is convenient to transfer sets of files this way as they are kept organized, and compression reduces the risk of file corruption in transit. Look at the man page for `tar`. Create a directory and create some emtpy files in it, by using the commands `mkdir` and `touch`. Use `tar` to create an archive from the directory and its contents. Now, extract the directory and remake the archive, this time with compression. (hint: `tar -cf, tar -czf, tar -xf` etc)

Note that like most Unix utilities, tar is not careful about clobbering files, so if you unpack an archive full of files in your directory and you already had files with those names (e.g. README), they will be overwritten.

4. Create a file called chrnames, that contains these lines:

chr2
chr5
chr7

The script below will create a new blank file for each line in this file. Run it, to check.

```
#!/bin/bash

for i in `cat chrnames`
do
   echo $i
   touch $i.peaks.bed
done
```

Modify the script so that it reads the file, and for each line in the file, it should create a new file containing all lines in chip_peaks.bed that match the chromosome specified.

5. On a Unix platform, compiling programs is a common task. We will download and compile samtools (a very popular toolkit for manipulating alignment data). Download the source release from here: http://www.htslib.org/download/
and follow the instructions to compile and install it. A convenient place to install this is in /usr/local/bin/ (if you are the administrator of the computer). If you are not the administrator, simply install it into any folder that you own.

You may run into trouble . . . if you see an error, try typing `which gcc` and see what you get. If you are using a Mac, you may not have a compiler installed, in which case you'll need to get the Developer Toolkit.