Lab 01 | Linux and Bash

ASSIGNMENT

Lab 01 | Linux & Bash: The goal of the lab is to be able to login to a bash terminal (bluehive) and use basic bash commands to copy/move around files and directories and run other commands. While most of the labs use Python notebooks, there will be some cases when you need to copy files into your directory. Learning bash is also practically since it is very useful for running bioinformatics software and it serves as a gentle introduction to interpretive environments such as Jupyter notebooks.

Instructions: Read and follow the directions in the lab. For this lab you can type your answers into a word document and upload pdf to Gradescope. There is no need to export-HTML of Jupyter session. However, if you include the command that you used to get your answer, you will be able to receive partial credit for your answer, even if it is incorrect.

SERVER LOGIN AND SHELL PROMPT

Start a Jupyter Lab session via bluehive or your own computer as described in the Prelab.

https://jupyter.circ.rochester.edu

Start a bash terminal: scroll down to the bottom of the Launcher window and click on "Terminal" under "Other". This will launch a bash terminal.

You will see a text-based, command line shell environment. In a shell, to show that the server is awaiting user input, the prompt will show:

```
[username@hostname workingdirectory]$
```

The ~ symbol is used for shorthand to indicate your home directory. On Linux file systems, directories start at the root (/) and are separated by a forward slash. Your home directory in longhand on bluehive is /gpfs/fs1/home/username.

Basic bash commands

Cheat sheet of commands:

pwd	print working directory	man	manual	chmod	change file mode
ls	list directory contents	mkdir	make directory	mv	move (rename)
cd	change directory	cat	concatenate files	cp	copy
grep	print matching lines	more/ less	scroll through text file	head/ tail	output part of file
find	find a file	sed	stream editor	awk	pattern processing
echo	display a line of text	wc	word count	nano	another text editor

Type 'pwd' and press enter to execute the command.

```
pwd
```

This command stands for “**p**rint **w**orking **d**irectory”.

In the scenario you didn't know what the `pwd` command did or you want more information about how to use it and what options there might be, you could run the `man` command to view its manual page.

```
man pwd
```

To scroll through the manual use your arrow keys or page up/down, and press 'q' to quit.

Create a directory called 'labtest' in your home directory.

```
mkdir labtest
```

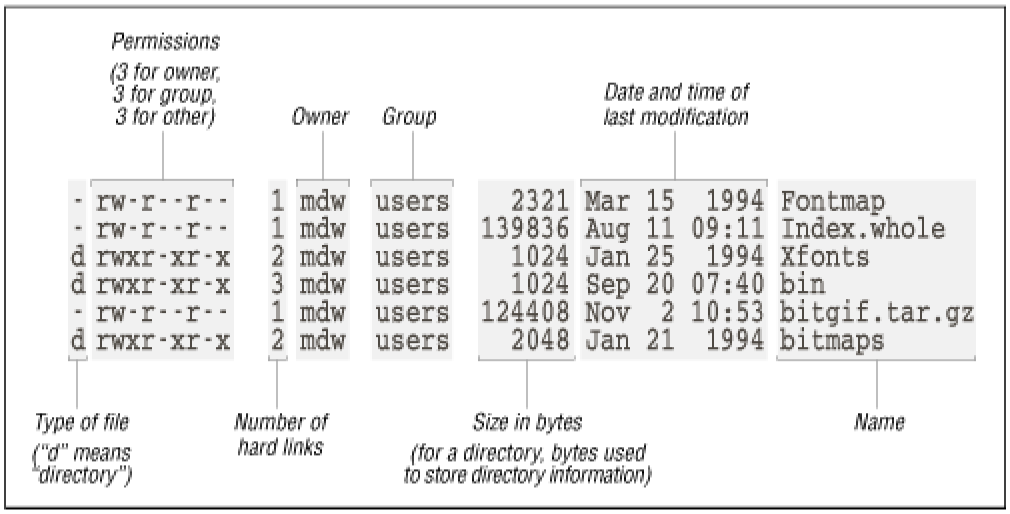
Check to see what is in your working directory using the list command:

```
ls
```

More information can be obtained using flags (options) to the 'ls' command.

```
ls -l
```

The '-l' flag is for long format and provides information about: permission, owner, group, size, etc:



```
- rw-r--r--  1 mdw  users    2321 Mar 15  1994 Fontmap
- rw-r--r--  1 mdw  users  139836 Aug 11 09:11 Index.whole
d rwxr-xr-x  2 mdw  users    1024 Jan 25  1994 Xfonts
d rwxr-xr-x  3 mdw  users    1024 Sep 20 07:40 bin
- rw-r--r--  1 mdw  users  124408 Nov  2 10:53 bitgif.tar.gz
d rwxr-xr-x  2 mdw  users    2048 Jan 21  1994 bitmaps
```

Flags can be separate 'ls -l -t' or together 'ls -lt'. The dash indicates a flag will follow, the 't' flag sorts by time stamp.

Move into the 'labtest' folder using 'cd' to "change directory".

```
cd labtest
```

Notice what our prompt has done. Our new working directory is `labtest`. To move back to your home directory you can either specify the location you want to go to exactly (username and folders may differ depending on your user name and whether you are on bluehive or your own computer:

```
cd /gpfs/fs1/home/username
```

You can also move back to your home directory using the "~" as a shortcut.

```
cd ~
```

Finally, you can move "up" one director from "labtest", which should be your home directory. To move up one directory use "..", which indicates the directory above. A single "." indicates the current directory. Thus,

```
cd ..
```

makes you move to the directory above.

Now lets remove the "labtest" folder from your directory. Make sure you are in your home directory and use the remove command with the "-r" recursive flag to remove the directory and any contents:

```
rm -r labtest
```

You will be prompted to indicate whether you are certain. You can respond with "y" for yes and "n" for no. Confirm with "y".

To remove a file you can just type "rm filename", "-r" flag is needed to indicate a recursive remove.

Getting Lab01 files into your directory

Copy the "Lab01" folder to your directory using the copy "cp" command. Include the "-r" flag to recursively copy the directory and all its contents.

If you are on bluehive use:

```
cp -r /public/jfay6/bio253/Lab01 .
```

If you are on your own computer, download the Lab01.zip file from blackboard, unzip it, and copy the folder to your current location.

The copy command requires two arguments: SOURCE followed by DESTINATION. In this case the destination is "." or the directory you are in.

It is often tedious to type long, case-sensitive names into your terminal. Bash makes this easier with Tab-completion! Try only typing the capital L first 'ls L', then the tab key to autocomplete the command if the directory exists.
This saves time in the long run, and prevents you from typing inaccurate file paths.

Text files can be viewed using the following commands: `more`, `less`, `head`, `tail`, `cat`. `cat` prints the output to the screen, `more` and `less` are programs that enable you to view the file contents, use arrows to scroll, 'q' to quit and backslash to search for text within the file. `head` and `tail` prints the first 10 or last 10 lines of a file and the '-n' flag sets the number of lines to be viewed.

The 'S288C.fasta' file in the Lab01 directory is a fasta file containing sequence from the *S. cerevisiae* lab strain (S288C). Fasta files can hold one or more sequences. Each sequence starts with a greater than symbol followed by the names of the sequence, e.g. ">chr1".

Execute each command on the 'S288C.fasta' file located in 'Lab01'.

```
cat S288C.fasta
```

Too fast, and tired of waiting for the end of the file. Use Ctrl-C to interrupt the command. Lets use `more` or `less` to scroll through the file.

```
less S288C.fasta
```

Great, but getting to the end would take a long time. Use `tail` to just see the end.

```
tail S288C.fasta
```

Editing text files can be done with `vim` or `nano`. For `vim` use ':help' to see commands and ':q' to quit. For `nano` use '^G' or control-G to get help and '^X' to exit.

To find files use 'find startingpoint expression'.

```
find . -name '*txt'
```

This command finds files starting in the current directory '.' that have a name that matches '*txt'. The * is a wildcard that matches any character one or more times and the matching expression is in single or double quotes to preserve the literal string without bash expansion.

Question 1 (4 points)

Find the file "answer1.txt" in the Lab01 folder, view its contents. What word is in this file?

Question 2 (4 points)

What are the first and last ten bases (A,G,C,T) in the S288C.fasta file?

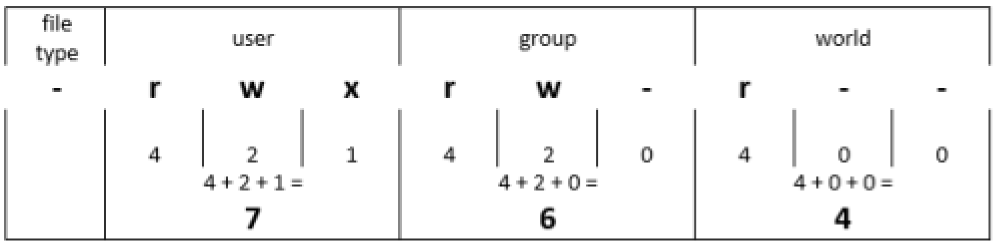
Trip-ups

Common trip-ups in bash are files or directory names with spaces, permissions and new line characters. Unix/Linux files systems can handle names with spaces, but most often they are avoided by using '.' or '_'. Spaces need to be avoided in bash since spaces are used to separate commands, flags, etc. Spaces can be used in file names if quoted to force literal interpretation or using the backslash for forces literal translation of the next character.

```
ls 'Lab01/file with spaces'
ls Lab01/file\ with\ spaces
```

In windows newlines are indicated by CR (carriage return) and LF (line feed) characters, which appear as '^M' in Linux. Mac OSX and Linux use the LF character. This can cause problems if you save a text file on a windows computer, transfer it to a Linux system and try to read it. Changing the newline characters can be done with 'sed' or 'awk'.

File permissions are another problem that can arise when transferring files from another system to Linux. Because file permissions are different between Windows and Linux, files transferred between the two can have the incorrect permissions. File permissions on Linux can be changed using the `chmod` and `chown` commands. `chown` changes the file owner or group and `chmod` changes the file mode. In Linux, file mode or permissions are set separately for the user (owner), group (group owner) and other (everyone else). Each permission consists of read, write and execute access. The 'ls -l' command shows file permissions.



The `chmod` command can be used to change permissions.

```
chmod 644 filename
chmod ug+rw filename
```

The first command gives read and write to user, but read access to the group and everyone else. The second command gives read and write to the user and group.

The `wc` command prints newline, word and byte counts for each file. The output format and flags used to get each of these are described in the manual: 'man wc'.

Question 3 (4 points)

How many lines are in the 'S288C.fasta' file?

Bash variables

Variables are simply words that hold other values. Every time you log into a Linux server and open an interactive shell, many variables are created and initialized for Bash to function properly.

Environment Variables

The environmental variable `USER` is assigned your username. To see what value some variable holds, we can use the `echo` command, which prints a line of text given as input.

```
echo $USER
```

The prefixed $ tells the `echo` command, I don't want to literally print '`USER`', but rather the value the variable `USER` contains. In each of our unique cases, and as the name variable suggests, the value printed for each of our commands vary.

User-defined Variables

Variables can also be defined as follows.

```
MESSAGE="Hello World!"
```

This has created a variable called `MESSAGE` that holds the value `"Hello World!"`.

We can view this variable's contents by using the echo command.

```
echo $MESSAGE
```

We see that "`Hello World!`" has been printed to the screen. We can use variables for file locations as well. For instance, to make a variable for the "file with spaces", let's assign the file name to a variable.

```
FILE="S288C.fasta"
head $FILE
```

More commands: grep, awk and sed

`grep`, `awk` and `sed` all perform pattern matching using regular expressions 'regex'. The basic usage of regular expressions is boolean logic (AND OR), grouping (with parenthesis), quantification (characters or matches) and wildcards (e.g. *).

grep (global regular expression print) filters input against a pattern

sed (stream editor) applies transformation rule to each line

awk a record oriented editor rather than line oriented editor like sed

In more detail than mentioned about, FASTA format is a text-based format for representing either nucleotide sequences or peptide sequences, in which base pairs or amino acids are represented using single-letter codes. A sequence in FASTA format begins with a single-line description (header), followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (">") symbol in the first column. Typically, all lines of text are shorter than 80 characters in length. FASTA can contain a single header and sequence, or multiple header, sequence combinations. For example:

```
>SEQUENCE_1
CGACAATGCACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATG
ATCGCGTATGCGAGAGTAGTGCCAACATATTGTGCTCTTTGATTTTTTGGCAACCCAAAATGGTGGCGGATGAACGAGAT
>SEQUENCE_2
```

```
TGTGCTCTCTATATAATGACTGCCTCTCATTCTGTCTTATTTTACCGCAAACCCAAATCGACAATGCACGACAGAGGAAG
CAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATCGCGTATGCGAGAGTAGTGC
```

Thus, the '>' is an indication of the number of sequences in the file and what those sequences are. Try these commands.

```
grep -c '>' S288C.fasta
sed -n '/>/p' S288C.fasta
awk '/>/ {count++} END{print count}' S288C.fasta
```

Here, the '-c' flag in grep counts lines with pattern '>' in the file S288C.fasta. A useful grep flag is '-v' which matches and prints any lines not containing the pattern. The 'sed' command suppresses auto printing (-n), matches '/>/' and prints. The 'awk' command has an expression that matches '/>/', then increments a variable 'count' with the '++', then prints.

Question 4 (4 points)

How many sequences are in the multi-fasta 'S288C.fasta' file?

I/O Redirection

Input and output in the Linux environment is distributed across three streams:

standard input (stdin)

standard output (stdout)

standard error (stderr)

Standard input is transmitted through the user's keyboard. Standard output and standard error are displayed on the user's terminal as text.

Linux includes redirection commands for each stream.

> filename

< filename

'>' redirects output to 'filename' and '<' redirects standard input from 'filename'. '>>' and '<<' do the same but append to any existing content.

```
echo 'Hello world!' > file1.txt
cat file1.txt
echo 'Hello again' >> file1.txt
cat file1.txt
cat < file1.txt >file2.txt
```

Pipes are used to redirect a stream from one program to another. When a program's standard output is sent to another through a pipe, the first program's output is received by the second program, rather than by being displayed to the terminal. Only the output of the second program will be displayed. Pipes are represented by vertical bar: '|'

```
grep '>' S288C.fasta | less
grep '>' S288C.fasta | wc
```

The first command pipes the output of grep into the 'less' program (type 'q' to quite less). The second command pipes it into the `wc` program which counts newlines, words and bytes.

Question 5 (4 points)

How many bases (A, G, C, T) are in the 'S288C.fasta' file?

Hint: don't count the headers, remove them using grep and the invert (-v) flag. Pipe this output to wc or awk to count the characters excluding the lines with headers. Note that wc character counts include new lines which you don't want to include.

Bash Scripts

When you only plan on executing a bash command once, it only makes sense to execute the command and move on. However, if you wish to generate a series of commands to do something more complicated, it makes sense to write the commands in a file referred to as a ***script***, so that the series of commands can be executed more easily and more than once.

Let's create our first bash script in our `lab01` directory using the `nano` text editor:

```
vim myscript.sh
```

Notice we gave this file an ending of .sh. In this case, that is short for shell to denote this is a shell script file. It is not necessary that we gave our file the extension, but it informs the user that the contents of the file are intended to be run using a Linux shell. Similarly, .docx files are used to indicate a MS word document.

To enter edit mode, type "i". Then enter the following in the first line of your script file:

```
#!/bin/bash
```

This statement is referred to as a "she-bang" statement, and it instructs the shell running this script of what application to use to execute the contents. In this case the 'bash' program located in '/bin/'.

In the next line let's insert our 'Hello World' statement:

```
echo "Hello World!"
# This line is a comment
```

Comments are lines prefixed with a # and end when a newline character is encountered. Comment lines are not executed. To save type ":w" and hit return. To quit type ":q" and hit return. To run this script type:

```
bash myscript.sh
```

To make your script directly **executable**, use the `chmod` command to add an e**x**ecutable permission to everyone '+x'.

```
chmod +x myscript.sh
ls -l
```

We can see the permissions on this file have now been altered to include execution. Now run the script using:

```
./myscript.sh
```

The leading ./ is needed by the interpreter to know where the script 'myscript.sh' is located. You don't need to specify the location of grep because the location of grep is already loaded as an environmental variable using $PATH. To see all environmental variables that have been set, use 'env'. To see your path use 'echo $PATH'.