BIO334 Day 1: UNIX

Introduction and connecting to Renku environment

It's not a coincidence that the UNIX operating system is the foundation of the most widely used operating systems in the world, including Android, macOS, iOS and Linux. It is powerful, versatile, secure, and its core (called kernel) is written for almost on any architecture (processor). It is also the operating system that runs on vast majority of super-computers in the world and almost all of the internet infrastructure.

You can communicate with the operating system through GUI (Graphical User Interface), which is different on each platform, or through *shell* (also colloquially called *command line* or *terminal),* which is almost identical in all UNIX based operating systems. If you know how to use shell, you know how to communicate with any UNIX based system. These exercises will teach you basic shell commands and fundamentals on how to utilize the power of the command line.

Similarly, to windows on UNIX based systems the file system consists of tree-like structure of folders, subfolders, and files. The top-level is called *root directory* and it's designated with one slash `"/"` (it's more or less equivalent to `"C:\"` in Windows) and each subfolder is separated by a forward slash, like this:

```
/home/daniel/work/collaborations/cats/hovercat.jpg
```

At first the command line environment may seem unfamiliar and slow to use, but give it a chance and quickly you will learn that you can do things that wouldn't be possible in a standard GUI interface.

Before we start using the command line, you will have to create your own virtual environment on the Renku platform:

- Log in.
- Select the Environment tab.
- Click [New]
- Set the default Enviroment to "/lab", leave the other options as is.
- Click "Start Enviroment"
- Wait few seconds for the virtual machine to build and click "connect"
- Select "Terminal"

When you see "$" sign followed by the blinking input cursor you are ready to go, so proceed to exercise 1.

The  sign means that it's something you need to solve (solutions to these exercises can be read from the solutions file)

The  sign are useful tips or annotations.

Exercise 1: basic file and directory manipulation

The dollar character ($) is a convention the signifies that you are in the command line input, don't type it).

$ **whoami**	prints out your user name (don't worry if you do not recognize it, on your computer it will bear your name)
$ **pwd**	prints the current directory
$ **cd ~**	change directory command, the tide character is a shortcut to your home directory. In UNIX systems each user has a home directory. This will be your playground.
$ **pwd**	prints the current directory (notice the directory change). The last subdirectory is named after you, the user.
$ **ls -l**	ls means list, -l means long; this command lists all files, permissions and directories in the current directory
$ **mkdir** tmp	make directory; the command creates the directory "tmp" in your home directory
$ **cd** tmp	brings you to your newly created directory tmp
$ **pwd**	you can see that your current working directory changed
$ **cd ..**	move one level up
$ **ls -l**	you should see the newly created tmp directory
$ **press** ↑ (arrow up key)	press arrow up and down to go through recent commands
$ **vi** tmp/poem.txt	start editing file tmp/poem.txt
press i	press i to switch to "insert" mode in the vi editor, while in this mode "insert" tag should be displayed at the bottom left corner of the page.
http://www.poetryfoundation.org/poem/171621	open web page and copy/paste the poem to the poem.txt file in your vi editor while in "insert"mode
press **ESC**	press ESC to switch to "command" mode in the vi editor. The "insert" tag on the bottom of the page should disappear.
type **:w**	Now you are back in the command mode. press **:** and then **w** and **ENTER** to write contents to file. The commands will appear in the lower left corner.
type **:q**	press **:** and then **q ENTER** to quit the vi editor and return to the shell
$ **cat** tmp/poem.txt	displays the poem
$ **cat**	what happens now? Try typing anything and press enter.
press **Ctrl** and **c**	(you are in the stdin mode where cat command listens to your keyboard input and outputs anything you type to the screen). Admittedly not very useful. Press the **Ctrl+c** combination to *kill* the current command. Remember that combination. It is a useful trick to exit an unwanted command.
$ **head** tmp/poem.txt	displays first 10 lines of the poem
$ **tail** tmp/poem.txt	displays last 10 lines of the poem
$ **man** head	use **man** to get more information about the head command; find and write down the head command with the correct parameter to display not only the first 10 lines of the poem but the entire poem (20 lines)
$ **less** tmp/poem.txt	less is a convenient command for displaying file content. Large files that do not fit in the screen can be scrolled using "up" and "down" arrow. Mouse/trackpad won't work. Return by pressing **q**

`$ grep house tmp/poem.txt`	if you are searching for something, you can use **`grep`** `text` to display only lines matching `text` How many lines you see?
`$ wc -l tmp/poem.txt`	displays the number of lines in the file
`$ wc -w tmp/poem.txt`	counts the words in the file
`$ cd tmp`	Enter the "tmp" directory.
`$ cp poem.txt poem2.txt`	**`cp`** command is used to make a copy the file. The second argument can be either a file name - this will make a copy of the file under the new name - or a path to a directory. The latter will create the copy of the file in the specified directory under the same name. For example `$ cp poem.txt ../` will create a copy of poem.txt in the directory above.
`$ ls -l`	you should see 2 files now
`$ cd ..`	go back one level
`$ cp -r tmp poems`	you just made a copy of the entire directory "tmp" as "poems"
`$ ls -l`	you should see now both directories "tmp" and "poems"
`$ ls tmp/*.txt`	you can use the wildcard (*) to list files and directories that match your expression. In this case, all the files that ends with "`.txt`" extension.
`$ rm -r tmp`	**`rm`** removes files and folders. The -r flag stands for recursive. It will delete this folder and all files and folders within.
`$ clear`	clears the terminal display

More often than not you won't have to write down the whole command name or the whole file name, and it's enough to write the first one or two letters and press [tab] and UNIX will fill the rest for you. If there is an ambiguity which command or file you have actually meant, press [tab] again and UNIX will display all the possibilities which start with your characters.

Try to type **`less p`** in the command line and then press [tab] 4 times. Describe what has happened each time you press [tab].

Try to use [tab] as often as possible throughout the exercises. It should be a second nature to tap it all the time.

Exercise 2: pipes and redirecting input / output

The pipe operator takes the output from one command and uses it as the input for the next command:

```
$ command1 | command2 | command3 …
```

The standard output of command1 is redirected (piped) to the standard input of command2, etc.

It's one of the most powerful tools at your disposal. This way you can combine an arbitrary number of programs in one command. In more technical terms, the pipe operator is used to create concurrently executing processes that pass data between them.

the output in UNIX is called *standard output*, as opposite to *standard error* output which is a way for the script to print any warnings or errors on the screen that won't be passed to the next step.

For example, a common expression in bioinformatics that you may want to write is something like this:

```
read data.tsv file [then] select third column [then] select lines that say
"sample2" [then] count how many lines there are
```

This would translate in command line to this expression:

```
$ cat data.tsv | cut -f 3 | grep "sample2"| wc -l
```

What is the standard **input** and standard **output** of the command **`grep`** in the example above?

Try it yourself, write a command that reads poem.txt file and prints only the lines that include the word "house" (use **`cat`** and **`grep`**)

Redirecting input and output. The output from programs is usually written to the screen, while their input usually comes from the keyboard (if no file arguments are given or nothing is *piped in*). To redirect standard output to a file instead of the screen, we use the **>** operator:

`$ echo hello`	displays "hello" on the screen
`$ echo hello > hello.txt`	writes "hello" to the file
`$ cat hello.txt`	displays contents of file; you should see "hello"

In this case, the contents of the file hello.txt will be overwritten if the file already exists. If instead we want to append the output of the echo command to the file, we can use the >> operator:

`$ echo hello >> hello.txt`	appends "hello" to the end of the file
`$ cat hello.txt`	displays contents of file

Standard input can also be redirected using the < operator, so that input is read from a file instead of the keyboard:

```
$ wc -l < hello.txt
```

There are many ways to do the same thing in UNIX, you may have noticed that the above example is equivalent of the one that you have already learned:

```
$ cat hello.txt | wc -l
```

You can combine input redirection with output redirection, but be careful not to use the same filename in both places. For example:

```
$ wc -l < hello.txt > hello_counts.txt
```

Exercise 3: writing and executing a shell script

You now know the basics of the **vi** editor from the first exercise. Try to copy/paste the below simple bash program into a file and execute it. The hello world bash script:

```
#!/bin/bash
echo "Hello World"
```

Copy the above 2 lines and save them to the file "hello.sh". The first line tells the Unix how to run the program. In this case it says that the file is a shell command and it should run it using 'bash' shell (but it could be very well a Python or R script).

Make your file executable by typing:

```
$ chmod +x hello.sh
```

Every file and folder in UNIX system has a set of permissions which says who can read it, and who can write to it. One of the permissions states if the file can be executed. This command sets the permission so that you (or whoever have permissions to read the file) can also run it.

And now, to run it, you can simply type:

```
$ ./hello.sh
```

What has been displayed?

Note the "**./**" at the start of the command. The "**.**" tells the system that the file is in the current directory (for comparison the "**..**" meant the directory above).

Exercise 4: download FASTA file and count the number of proteins

For this exercise, go to the **data** folder using the **cd** command. This is where you should but all data files that we download during this course.
Now we will download a FASTA file. For this we will use a command called **curl**

```
$ curl https://stringdb-static.org/download/protein.sequences.v11.0/9606.protein.sequences.v11.0.fa.gz -o human_proteome.fa.gz
```

The file will download in the current directory named human_proteome.fa.gz. Unzip it using a command **gunzip**

```
$ gunzip human_proteome.fa.gz
```

How did the name of the file change when you unzipped it?

Count the number of proteins in the file using commands you have learned. In a FASTA format file the first line of each protein record starts with the "**>**" character.

Exercise 5: sort file on column

Download the protein abundance data (https://pax-db.org/downloads/4.1/datasets/9606/9606-BRAIN-integrated.txt) with curl, like you did in the previous exercise and name the file "abundance_data.tsv"

Explore the downloaded file using the **`less`** command. Note which column holds the abundance numbers.

Run the following command:

```
$ sort -k3nr abundance_data.tsv | less
```

Open manual of the **`sort`** command, look for option "**`k`**" and "**`n`**", "**`r`**". What do they mean in the context of the command above?

Remove the "**`n`**" parameter and run the sort command again. Explain what has happened.

Exercise 6: write a simple bash script

Often, you would like to run the same command with different parameters. As an exercise, write a simple bash script that will output numbers from 1 to 100. Use a for loop.

```
#!/bin/bash
for i in {1..100}
do
   echo $i
done
```

Save the above code to a file (e.g. **`script.sh`**), make the file executable (**`+x`** flag) and run it.

What is the output?

Exercise 7: iterating over files

Download the compressed folder containing a selection of proteomes with curl, and extract it using the following **`tar`** command. URL with the proteomes:
https://raw.githubusercontent.com/meringlab/Bio334/master/01_unix/exercises/data/proteomes.tar.gz

`tar` command:

```
$ tar xzvf proteomes.tar.gz
```

 while **`gzip`** compresses one file, **`tar`** on the other hand is used to compress the whole directory structure and all the files in it. The arcane string **`xzvf`** translates to **X**tract **Z**ipped **V**isual **F**ile, but is more commonly referred to as "Extract Ze Vucking Files!".

Enter the proteomes directory (**`cd`** command)

 Using the commands you have learned print out the number of files in the directory.

 By utilizing the same concept of **`for`** loop from the previous exercise, can you try to create the shell script that iterates over all fasta files in the directory, print their names and the number of sequences in each file?

 `for filename in *.fa`

 You can put more than one command between **`do`** and **`done`**, one line would print the file name, the other will print the sequence count.

 Exercise 8: exploring a FASTA format file

Dictyostelium discoideum (www.dictybase.org) is an interesting social amoeba and a well-studied model organism. The whole genome sequence is already available here: https://raw.githubusercontent.com/meringlab/Bio334/master/01_unix/exercises/data/dd.tar.gz

Download the file with **`curl`** and extract it with **`tar`**. The FASTA format is widely used in sequence distribution, see the description at: http://en.wikipedia.org/wiki/FASTA_format.

 Explore the file using **`less`**. Use **`grep`** to find out how many chromosomes are present in the file. Use **`grep`** ( **`-v`**) to only print out the genomic sequence. How large is the genome?

Now look at the RNA-seq data sample stored in the rnaseq.fastq file. This file includes qualities for each nucleotide (see FASTQ description at http://en.wikipedia.org/wiki/FASTQ_format).

 How many reads are in the file?

 Exercise 9: searching for short sequences in the *Dictyostelium discoideum* genome

 Is the sequence "AAAAAGAGATACAT" present in the DD genome (`dd/dd.fasta`)?

use **grep** to find out

In the virtual environment we have installed a software called **bowtie2**. Bowtie2 is a short-sequence read aligner (e.g. 150 nucleotides long). It aligns the sequences to the reference genome, it's fast, memory efficient and supports mismatches.

* You can also use **bowtie2**. First build the index of the DD reference genome. The format is: "`bowtie2-build <fasta_file> <custom_index_name>`". In the dd folder, you could use:

```
$ bowtie2-build dd.fasta dd
```

Once the index is created (you do this only once for each reference genome, i.e. each FASTA file), you align one read (sequence) to the genome by typing:

```
$ bowtie2 -x dd -ac AAAAAGAGATACAT > dd.sam
```

Explore the SAM results file with **less -S**. The parameter "-S" prevents line wraps, so you can see one alignment per line.

Does **bowtie2** find more alignments compared to **grep**? Why could that be?

Exercise 10: alignment of RNA-seq sample reads to the *Dictyostelium discoideum* genome

To align the RNA-seq reads in the rnaseq.fastq file, you first need to index the *Dictyostelium discoideum* genome. Use bowtie2-build to create an index with name **dd** (if you didn't already build the index in the previous exercise):

```
$ bowtie2-build fasta_file custom_index_name
```

After you created the index, you can align the reads by typing:

```
$ bowtie2 -x dd -U rnaseq.fastq > dd.sam
```

The results are returned in SAM format and stored to the dd.sam file. How many reads align?

Bonus exercise: Running UNIX on your machine.

As mentioned in the beginning of these exercises, UNIX is a foundation of most of the modern operating systems. You are probably using one now. Try to do the above exercises on your computer.

- Linux: If you are using LINUX you are already proficient in using the command line. You can skip this exercise.
- macOS: Run "terminal" to access the command line.

- Windows: Windows doesn't have built-in UNIX like terminal, but you can install software called Cygwin, which is Linux environment for Windows.
- Android and iOS: although both run a version of UNIX underneath the GUI, the command line can't be accessed without rooting (hacking into) the device and severely compromising its security, so don't do it, seriously… don't. There are however many apps that similarly to windows install a UNIX environment on your handheld. Simply search for "terminal" in your respective app store.

 How many exercises were you able to do? Did you notice any differences?

System information, processes and other useful commands

uname -a	display system information
man *command*	display manual page of command
df -h	list mounted disks with available space
du -h *path*	show space usage
top	display running processes
kill -9 *pid*	kill process

File and folder manipulation, compression

pwd	display current folder
ls -l *path*	list files and folders
cd *path*	change folder to path
cd ~	change folder to home folder
mkdir *name*	make folder
rmdir *name*	remove folder
cp *source dest*	copy file/folder and all its contents
less *filename*	display file content
wc *filename*	count number of lines in file
head *filename*	shows first few lines of file
tail *filename*	shows last few lines of file
gzip *filename*	compress file with gzip (adds .gz extension)
gunzip *filename*	decompress and remove .gz extension
tar xfvz *filename.tar.gz*	decompress files from tar.gz archive
tar zcvf *archive.tar.gz folder_to_compress*	creates archive.tar.gz
unzip filename.zip	unzip archive

Network and file transfer

curl URL -o filename	download URL to filename
ssh *username@host*	remote login to host with username

"vi" editor

$ vi *filename*	start editing file with vi
i	switch to "insert" mode
ESC	switch to "command" mode
:w	save
:q	quit
:x	save and quit
/<pattern>	search for pattern, <n> gives you the next match
:q!	quit without saving changes