Practical: Intro to linux

1. Getting around in Linux

Many of the tasks in in these assignments will be carried out using the terminal (command line) application. Any command listed from here on out which you will be expected to execute in the terminal will be on a separate line starting with $ and the command written in bold-face type. Your should not type $ (it is only there to indicate promt). For example:

```
$ cd /home/student
```

In this example you only write “cd /home/student” and press enter.

1.1 To get a thorough directory listing, type ll and press enter. Try it.

```
$ ll
```

1.2 To navigate to a specific directory, use cd (change directory). Try cd Programs from you home directory.

```
$ cd Programs
```

1.3 To navigate backwards one directory, type cd ..

```
$ cd ..
```

1.4 You can make a new directory with mkdir “directoryname”

```
$ mkdir directoryname
```

1.5 You can remove a file with rm filename. Or a directory with rm -rf directoryname. Be careful with rm -rf. If you make a mistake or typo, you can end up with deleting something you weren’t supposed to.

```
$ rm -rf directoryname
```

1.6 You can move a file with mv filename path/filename. Or copy it with cp filename path/filename. To copy a directory you need the -r flag, ie. cp -r my_directory /home/student/my_directory2. First make a directory named my_directory, then copy it to the folder named my_directory2.

```
$ mkdir my_directory
$ mkdir my_directory2
$ cp -r my_directory my_directory2
```

1.7 The directories containing all your exercises and practicals are located on a shared disk between the VMs. To attain a local copy for yourself, you need to copy the shared folders over to your VM.

```
$ cp -r /shared/Practical ~/.
$ cp -r /shared/Exercises ~/.
```

1.8 No matter which directory you are in, you can type cd to get back to your home directory. There is also an auto-complete function in linux, so instead of typing cd Programs, you can type cd Pro and press the Tab-button, which makes linux write the rest of the directory name for you, if it is the only one that starts with “Pro”. This is very handy if you are working with long and complicated file names/directory names. First go your home directory, then try the auto completion.

```
$ cd
$ cd Pro and press the Tab-button
```

1.7 Lastly, the directories containing all your exercises and practicals are located on a shared disk between the VMs. To attain a local copy for yourself, you need to copy the shared folders over to your VM. (Notice the use of the `~` sign. `~`, or “tilde” as it is pronounced, refers to your home directory)

```
$ cp -r /shared/Practical ~/.
$ cp -r /shared/Exercises ~/.
```

Remember, with any basic linux command/program, you can always type “command” –help to get a listing of available parameters to change the default behavior of this program.

2. Using an editor to edit and view files

A lot of the work done in the terminal environment is editing text files. vim (VI improved) is an excellent editor to get the job done.

2.1 To edit a file, type

```
$ vi myfile.txt
```

Press i to enter insert mode. Now you can write any text you like. Save your changes by first exiting insert-mode using esc, then typing :w Quit the editor with :q You can quit and exit at the same by typing :wq
(Just remember that vi reads in the whole file at once, which can be slow for really big files. If you want to look at a big file without really making any changes, the program less will open it in increments)

2.2 Try viewing the file your just made using the program less

```
$ less myfile.txt
```

3. Working with pipes and redirects

Sometimes it can be beneficial to store some of the information or logs produced by a tool or the operating system. A tool that prints results to stdout, which is the text that prints to your screen, can be redirected to a file instead. To achieve this we use the “greater than” - sign `>`, which is called a redirect.

Try ls > output.txt to save the output from ls to a file so you can view it later. Open the file output.txt to view its content.

```
$  ls > output.txt
```

The opposite “less than” – sign `<` can be used to redirect from a file to use it as input.

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

wc -l counts lines from what ever you input. How many lines are in your file output.txt?

You can also chain programs by letting the output from one program feed into the input of the next. This can be achieved using pipes `|`

```
$ ls | wc -l > lines.txt
```

lines.txt will contain a number. What does this number indicate?

4. Performing basic data manipulation.

This last assignment will teach you the basics of how to efficiently manipulate text files in linux. This knowledge is essential for a bioinformatician, as often you will need to manipulate text files in some manner, either to count lines, cut and paste certain columns or sort your data. And you do not want to spend 30 minutes writing a script, or open 1 GB of data in Microsoft Excel.

Luckily, most distributions of Linux (like Ubuntu, the one on your virtual machine) contain a small set of programs that can solve many of these problems

In ~/practical/1/interpro you will find 300 annotation files from an interproscan5 run against the Norwegian Moose feces metagenomic dataset. These files contain annotations for all the genes present in all contigs over 3000bp in length in tab separated format (tsv). Interproscan5 is part of our in house pipeline Meta-pipe, which analyses metagenomic samples. A description of the columns in these files can be found here: https://github.com/ebi-pf-team/interproscan/wiki/OutputFormats

Here is an example of some linux programs piped together to output the top 10 most common Gene Ontology terms in the sample:

```
cat * | cut -f 14 | grep GO | tr '|' '\n' | sort | uniq -c | sort -nr | head
```

4.1 This looks pretty cryptic at first glance. Try executing just `cat *` in the ~/practical/1/interpro directory and see what happens. (To stop a program, hold left ctrl and press C). cat basically prints the contents of a file, and with the asterisk (*) at the end, it will print all 300 files in sequence.

4.2 The pipe (|) tells the system to forward the output from one program to the next, instead of outputting it to the screen. Try executing `cat * | cut -f 14` Do you understand what happens? Try adding one pipe and one program with parameters and see if you understand what happens in each stage. Remember, you can always run program –help to get a description of usage, (ex: cut –help).

4.3 Try mixing up the above programs to answer these questions:
How many protein accessions total are there in these files? (Hint: wc -l counts lines)
How many unique accessions? (Hint: Always use sort with uniq)
Can you produce a top 10 list of interpro accessions (column 12)?

If you want to write any of your results to a file, use a redirect (>) which prints output to a file instead of the screen. Example:

```
cat * | head > results
```

Some other useful one-liners:

Convert a fastq file to fasta:

```
awk 'NR % 4 == 1 {print ">" $0 } NR % 4 == 2 {print $0}' my.fastq > my.fasta
```

Count sequences in a fasta file:

```
grep -c '>' my.fasta
```

5. Introduction to Anaconda

Before you start this practical, there is a small presentation / intro
Anaconda is a package and environment manager for Linux, Windows and MacOS. Installing software can be awfully complicated, especially in Linux, but Anaconda makes this so much easier.
On your machine there is already several environments configured. This is because throughout this course you will be using tons of different software with different, and sometimes conflicting dependencies.

5.1 To get an overview of the packages installed In the current environment, type:

```
$ conda list
```

This command will print a list of installed packages in the current environment where the first column is the name of the package, second is the version and third the build (often indicating what programming language it is made for)

5.2 To get an overview of environments, type:

```
$ conda env list
```

This command will print the names of all the available environments as well as the path to the environment data. You can change environment by typing conda activate “environment”. Try

```
$ conda activate Plotting
```

If you run

```
$ conda env list
```

now, you will see that the asterisk is now shown after the Plotting environment in the listing, indicating that this environment is active, and also it should say (Plotting) somewhere before your promt sign in the terminal.

5.3 We have left out one of the programs you need to complete some stages of this practical on purpose. This is an assembler called Megahit. Make sure you are in the base environment (notice the star after the name) by typing

```
$ conda deactivate
```

Install Megahit by typing

```
$ conda install megahit
```

Afterwards, type

```
$ conda list
```

and see if you can find megahit in the list of installed packages.

5.4 Lastly, to get some more experience with environments, you are going to install Qiime2 yourself into its own environment. In order to achieve this we will download a yaml file from the qiime2 repositories that describe all the dependencies required by qiime and also the conda-channels they can be found in. Navigate to your home folder using:

```
cd
```

Then download the yaml file which describes the Qiime2 conda package and dependencies:

```
wget https://data.qiime2.org/distro/core/qiime2-2018.11-py35-linux-conda.yml
```

Use less to view this file and see if you understand what it contains

```
less qiime2-2018.11-py35-linux-conda.yml
```

Create a new conda environment and install Qiime2 using this yaml file:

```
conda env create -n qiime2-2018.11 --file qiime2-2018.11-py35-linux-conda.yml
```