

NGS Introduction Course

Practical one, Linux and NGS tools

Jeroen Laros, Michiel van Galen

Tuesday, 1 April 2014



# 1 Linux

This first section will cover the basics you need to know to work on a Linux system. Most NGS tools depend on this knowledge. If you are already familiar with the command line interface you can safely continue to section two. If not we highly recommend to first complete this first section.

Hints:

- Keep in mind that bash is case sensitive.
- Whenever “course” is used, replace it with your user name.
- Auto completion can be done by pressing the TAB key.

## 1.1 Connecting to a machine

We will do the practical on a virtual machine in the LUMC. Everything is already prepared and ready to use. To connect we use ssh.

PUTTY?

```
$ ssh course@0.0.0.0
```

## 1.2 Files and directories

Find out where we are in the directory tree and list the content of directories. Try the following commands:

```
$ pwd
$ ls
$ ls .
$ ls ..
$ ls /home
$ ls /home/course
```

Now, create a directory and a text file using **nano**.

```
$ mkdir test_dir
$ cd test_dir
$ pwd
$ nano file1.txt
```

We have now opened the **nano** text editor and are editing the content of the file **file1.txt**. Try to create a FASTA formatted file, something like:

```
>header1
ATGCGT
>header2
GTCAAA
```

Hints for using **nano**:

- When done press **CTRL-X**.
- When asked to save changes press **y**.
- Press **enter** to confirm the name of the file.

Examine the directory and its content in detail. Use the **-l** and **-h** flags of **ls** for long listing and human readable output formatting respectively.

```
$ ls -lh
```

You can see the file size, who created it, and when it was created.

### 1.3 Redirecting

Create an other text file named **file2.txt** in the same directory. Make sure to put some text in this file as well.

Check the contents of the directory. If everything went well you should see the two files you created.

You can see the contents of a file with **cat**. Be careful with large files, it can take a very long time to print them to the screen.

```
$ cat file1.txt
$ cat file2.txt
$ cat file1.txt file2.txt
```

You might want to concatenate both files and write the result to a new file. We can redirect the output from **cat** to a file with the “greater than” sign (**>**).

```
$ cat file1.txt file2.txt > concat.txt
$ cat concat.txt
```

A way to quickly figure out how many lines a file contains is to use **wc** (word count) with the **-l** option.

```
$ wc -l concat.txt
```

With the **head** command we can print any number of lines from the top of the file to the terminal. Use the **-n** flag to indicate the number of lines you want to print.

```
$ head -n 2 file1.txt
```

Try different values for **n**. The command **tail** works in a similar way but instead of printing the first lines, it will print the last ones. Try some **tail** commands.

Some commands accept wildcards as input. The star (**\***) for example. Examine the following commands.

```
$ cat *
$ wc -l *
```

Files and directories can be removed with the **rm** command.

```
$ rm file1.txt
$ ls
```

You can see that the file is gone. Now we want to get rid of the whole directory. Before we can do so we need to leave the current directory, then we can recursively remove the directory using **rm** with the **-r** flag.

```
$ cd ..
$ ls
$ rm -r test_dir
$ ls
```

The directory **test\_dir** and all of its content is now removed.

## 1.4 Piping

To chain processes we can use the pipe (**|**) symbol. The file **/etc/services** contains a list of internet network services, the content is not important, but it is a large text file that serves our purposes for the following examples.

First, we use **cat** to show the content of this file, then we use a pipe to connect this output to the input of **less**. In this way, we will be able to conveniently scroll through the lines.

```
$ cat /etc/services
$ cat /etc/services | less
```

Hints for using **less**:

- Scroll down and up using the arrow keys.
- Press **q** when you are done.

Try to redirect output from **ls** to **less**.

The command **grep** searches for a given pattern, it will only print the lines matching this pattern. It requires input that we can provide by piping the output from **cat** to **grep**.

Suppose we want to have information about the **ssh** protocol.

```
$ cat /etc/services | grep ssh
```

We can save the lines of interest to a file in the same way we concatenated files earlier with the **>** symbol.

```
$ cat /etc/services | grep ssh > ssh.txt
$ cat ssh.txt
```

The **ps** command gives a list of current processes.

```
$ ps
```

To quickly count how many processes are running we pipe the output to **wc**.

```
$ ps | wc -l
```

## 2 NGS analysis

We continue this practical on how to use a selection of tools to analyze a small human short read dataset. The data will be inspected for quality, aligned, checked for variants and annotated.

### 2.1 Get the input

First we must fetch the course data for the practicals. We have precompiled a small human test set of short reads in an online repository. Use the next command to pull everything from that repository for you to work with.

How this exactly works is not important for now but if things work as expected, you should have a folder called ngs-intro-course at the location you executed the command. Inside that folder you can find an “exercise” folder, providing everything you need to continue.

- Get the data:

```
$ git clone https://git.lumc.nl/humgen/ngs-intro-course.git
```

### 2.2 Quality control with FastQC

Every tool which is system wide available can simply be run by typing the name of the tool. Try typing part of the tool name and hit TAB twice. It will automatically auto-complete the name or give suggestions if it's not sure yet.

Most tools will show some documentation if you start them without any options. If this doesn't happen you have to usually add the -h which stands for help.

- Navigate into the ngs-intro-course/exercise folder
- Run a FastQC with the help function:

```
$ fastqc -h
```

Now we know how to run tools let's start by looking at the quality of the data by using FastQC. You can see in the synopsis of FastQC that it only requires an input file to work. There are other arguments, yet these are optional.

- Remember the Linux commands you learned earlier to check your output
- Run a FastQC analysis:

```
$ fastqc short_reads.fq
```

This will create a folder with data and plots of the analysis. Have a look at the fastq\_data.txt file to get an idea of the metrics of this set of reads.

## 2.3 Alignment with Bowtie

We skip the clipping process and continue with the alignment. Run Bowtie, same way as you did with FastQC to see what options are available.

- Run Bowtie and show help:

```
$ bowtie -h
```

You will see lots of options. Don't be overwhelmed, for now we will run a default analysis. However, feel free to check out what you could tweak. Be careful though, running tools in default or with adjusted parameters can greatly influence the results.

Running Bowtie requires at least two arguments. The first is the reference index we want to use. In this case we use a precompiled human mtDNA (16kb) reference. The reference is part of what we downloaded and available at `./bowtie_index/chrM` for you to use.

The index is followed by the fastq file. If we omit any further options the output is printed to the screen. Since we rather want an output we also give a name for the output as last option.

For downstream purpose we want one more option, namely the output to be in SAM format. We achieve this by adding the `-S` parameter. Note these optional arguments come first before the others. (As you can see in the help)

- Run a Bowtie analysis on the data by aligning to the mtDNA index:

```
$ bowtie -S ./ref/chrM course.fq course.sam
```

## 2.4 From SAM to sorted BAM

Now we have a SAM file with alignment information. You can look at the file using `"less course.sam"`. Many tools can work with SAM format, yet even more tools accept BAM format. The content is the same, only BAM is binary using 80% less disk space! Let's convert the file.

The first option we need to give Samtools is which function to use, in our case we use `"view"`. After that we give `"-Sb"`, basically telling Samtools that our input is SAM and we would like BAM output. Finally we redirect the output to a new output file using the greater than sign.

For many purposes it is convenient, and often even required, to have your BAM file sorted. This allows tools only have to go through the file only once. Think of an unsorted BAM file as a novel with unsorted pages, not really useful. To sort the file, we simply call Samtools sort. The first argument is the input file, followed by the output file name.

Note that we omit the `.bam` extension for the output file name. This will be done by Samtools.

- Convert the SAM file to BAM,
- Then sort the BAM file:

```
$ samtools view -Sb course.sam > course.bam
$ samtools sort course.bam course-sorted
```

## 2.5 Variant calling with Samtools

We have our reads aligned and nicely stowed in a sorted BAM file. This is usually the point from where the analysis forks of into different downstream analyses. It is a good idea to store and backup your BAM files for the projects you work on.

Let's continue with calling variants. Again, we make use of the Samtools bundle, yet this time we call mpileup. This converts the BAM into a per position summary of calls in text format. The size of this can get quite big so instead of saving the mpileup file we immediately feed it into bcftools using a pipe. We will not go into what the “-guf” flag means anymore. If you like, you can “samtools mpileup” to see what the individual characters mean in this sense.

As explained, the mpileup data will be fed directly into bcftools. At this point bcftools will go over the pileup format, picks out the variants and describes them into a bcf file. (A binary format of the vcf format.) In the next step we will convert the bcf file to a readable vcf file while at the same time we apply a filter step from vcfutils.

- Use mpileup and bcftools to generate a bcf file
- Convert and filter the bcf into a vcf file

```
$ samtools mpileup -guf ./bowtie_index/chrM.fa course_sorted.bam \  
| bcftools view -cgb -> course.bcf
```

```
$ bcftools view course.bcf | vcfutils.pl varFilter > course.vcf
```

Feel free to have a look at the vcf file to see how this reports the variants. (Use “less course.vcf”)

## 2.6 Annotation with Ensembl VEP

The variants in the vcf file are not really informative without any context. To annotate them, various tools are available. In the next exercise you will use VEP to annotate the variants in our vcf file.

- Run the VEP with the suggested options:

```
$ /usr/local/variant-effect-predictor/variant-effect-predictor -71/variant-effect-predictor \  
-i course.vcf -o course.vep --database
```

Inspect the output and notice there is multiple lines per variant and also lots of non informative information. Let's try to condense the output and at the same time expand annotation for interesting variants only. Manage this using the next parameters.

- Try some of these parameters and figure out what they do:
- --check\_existing --canonical --coding\_only --force\_overwrite

Again, take a look at the output and notice the effect of the given options. There are many more parameters to play with. Feel free to run VEP and give them a try to see the effect on the output you get.

- Play around a bit more with maybe some of these options:
- --sift=b --polyphen=b --regulatory --protein --domains --refseq

This is the end of the practical. We hope you now got an idea what it takes to work with a Linux terminal and NGS tools.

Thank you for participating!