PATHOGEN MULTIOMICS AND BIOINFORMATICS – RECIFE

Recife PE - 2025

COURSE DOCUMENTATION

February 10th – 14th, 2025

FIOCRUZ Pernambuco
LISBOA UNIVERSIDADE DE LISBOA
FACULDADE DE FARMÁCIA Universidade de Lisboa
imed Research Institute for Medicines

PATHOGEN MULTIOMICS AND BIOINFORMATICS – Recife 2025

:: Course Overview ::

Course Director:

- João Perdigão, PhD, Research Institute for Medicines (iMed.ULisboa), Faculty of Pharmacy of the University of Lisbon (PORTUGAL)

  E- mail: jperdigao@ff.ulisboa.pt

Instructors:

- João Perdigão, PhD, Research Institute for Medicines (iMed.ULisboa), Faculty of Pharmacy of the University of Lisbon (PORTUGAL)

E- mail: jperdigao@ff.ulisboa.pt

- Túlio Campos, PhD, Institute Aggeu Magalhães - Fiocruz (BRAZIL)

E- mail: tulio.campos@fiocruz.br

- Marcelo Paiva, PhD, Institute Aggeu Magalhães - Fiocruz (BRAZIL)

E- mail: marcelo.paiva@fiocruz.br

Course Contents

The course is structured in six distinct modules (Total Hours: 20) that covers basic Next Generation Sequencing analysis workflows:

Course Schedule

	10th Februray	11th Februray	12th Februray	13th Februray	14th Februray
	Monday	Tuesday	Wednesday	Thursday	Friday
09:00					
10:00					
11:00					
12:00					
13:00	**Module 1** Mapping	**Module 2** De novo Assembly	**Module 3** Pathogen ToolBox	**Module 5** Viral Genomic Surveillance and Metagenomics	**Module 6** RNA-Seq and Transcriptomics
14:00					
15:00			**Module 4** Phylogenetics		
16:00					
17:00					

Course documentation includes an introduction the topics covered in the course as well as exercises/tutorials. Moreover, further documentation concerning the installation of the computing system, overview and basic Linux commands is also provided.

:: Index ::

:: Course Computing System Overview ::

Introduction

This course will make use of a Virtual Operating System based on CentOS 7. This is a free, enterprise-class, community-supported Linux distribution that is sourced from Red Hat Enterprise Linux (RHEL).

For this course we will use a Virtual CentOS image that will run on Oracle VirtualBox, a free virtualization platform that runs across multiple systems. Every computer made available for this course is already pre-installed with Virtual Box already configured with CentOS 7. You just need to find the shortcut on your desktop, double-click it, then select CentOS Virtual Machine (VM) from the VM list on the left and click Start. This will open a new window and CentOS with all course files and bioinformatic software necessary for the course will soon start.

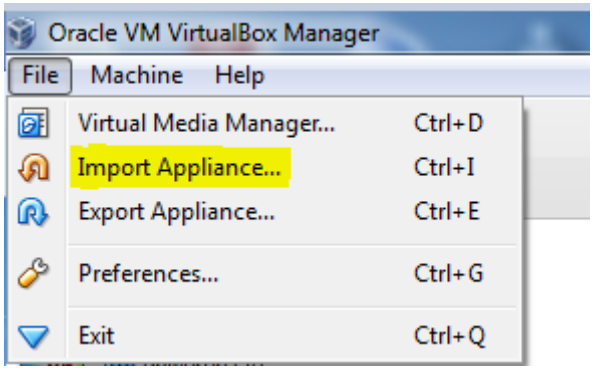
The image of this VM is also available (approx. size 40Gb) if you wish to take it and carry out some of the course analysis at home. Just ask the course instructors for the link to download this image. The following section explains how to import this image to VirtualBox on your computer.

Importing the OS Virtual disk into VirtualBox

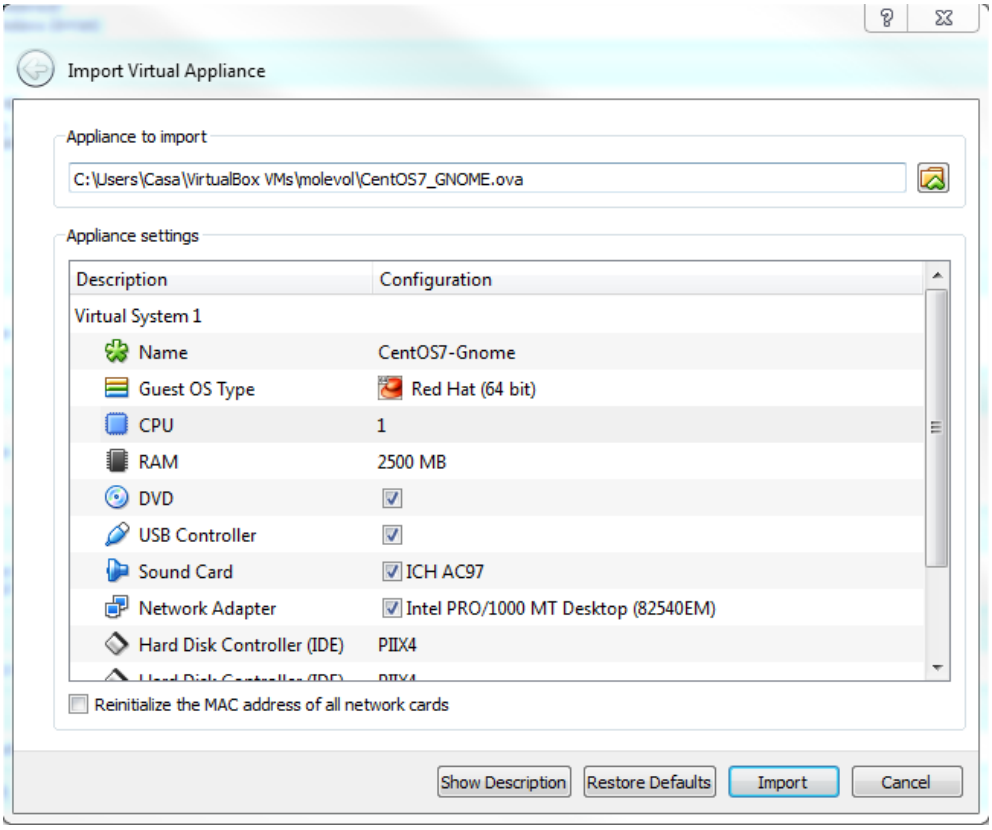
There are two main methods for importing the course VM to your VirtualBox installation: using an image disk file (vdi or vmdk file) or using an exported Open Virtualization Format Archive (OVF or OVA file). You will therefore need to check which type of file was made available to you and proceed accordingly.

Import from a OVA/OVF Archive:

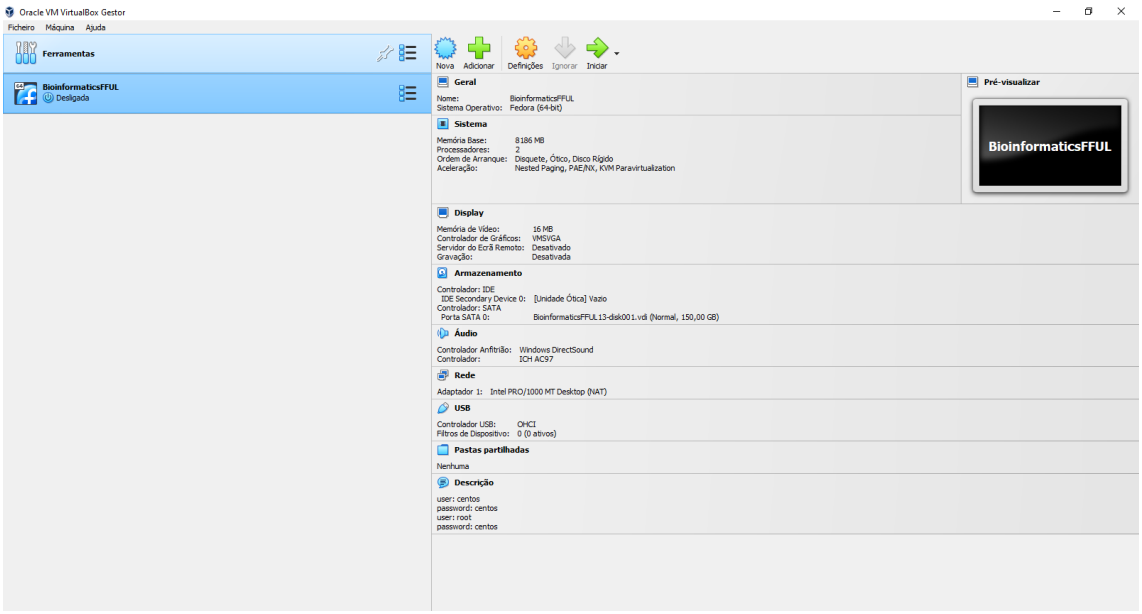
To import a VM in OVA/OVF format go to *File* > *Import Appliance*…



Then, select the OVA/OVF file from its location on your computer and click on *Import*.

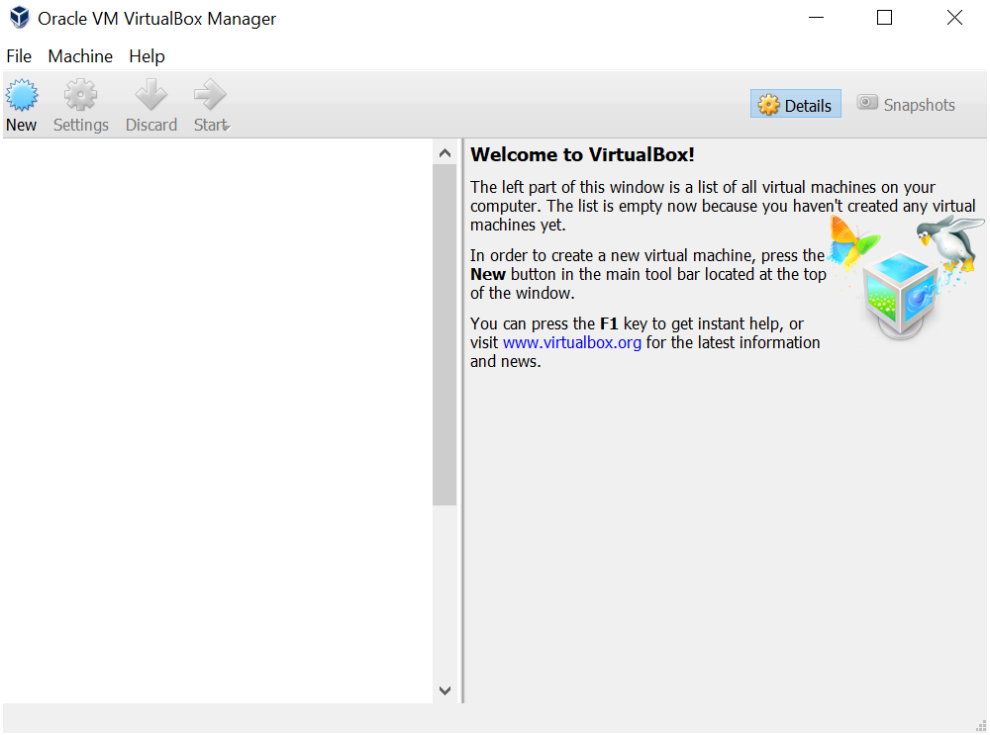


It may take some time to finalize the importation process.

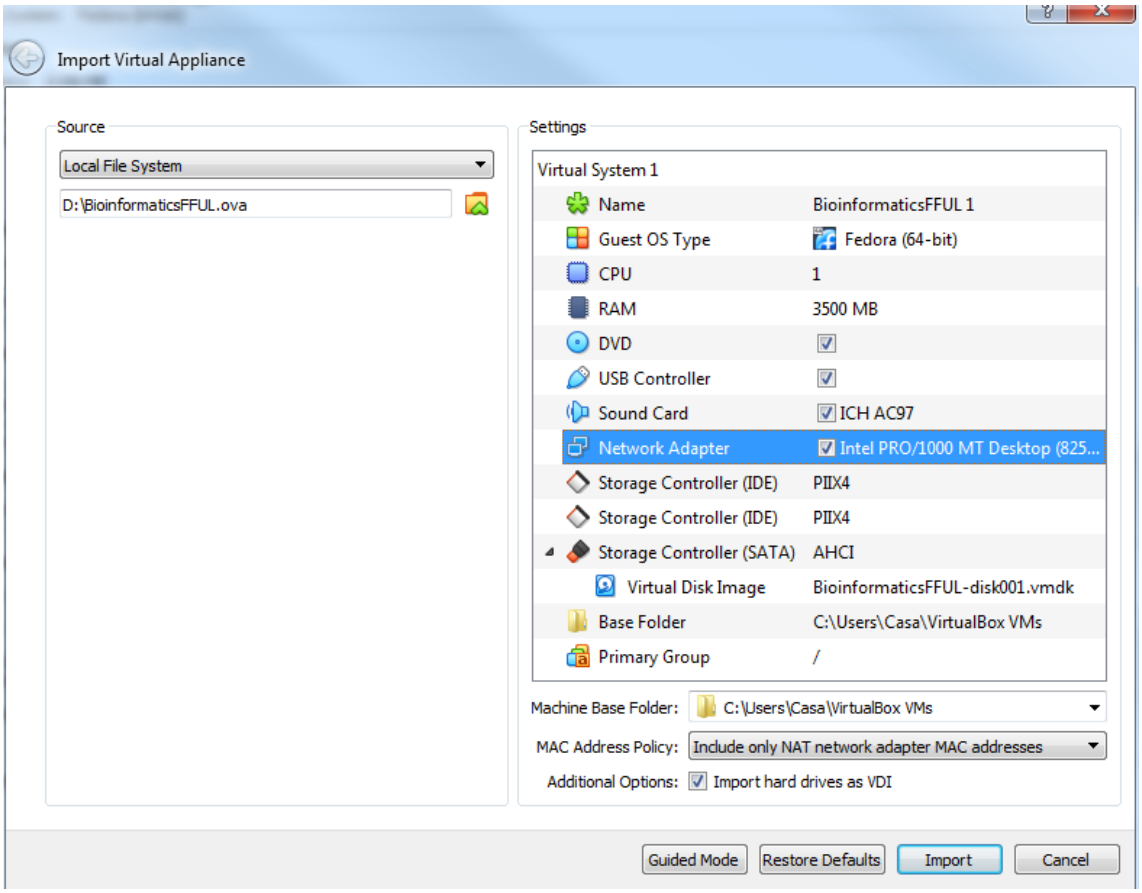


After the Importation process this VM will appear on VirtualBox VM list. To start it just select it and click *Start*.

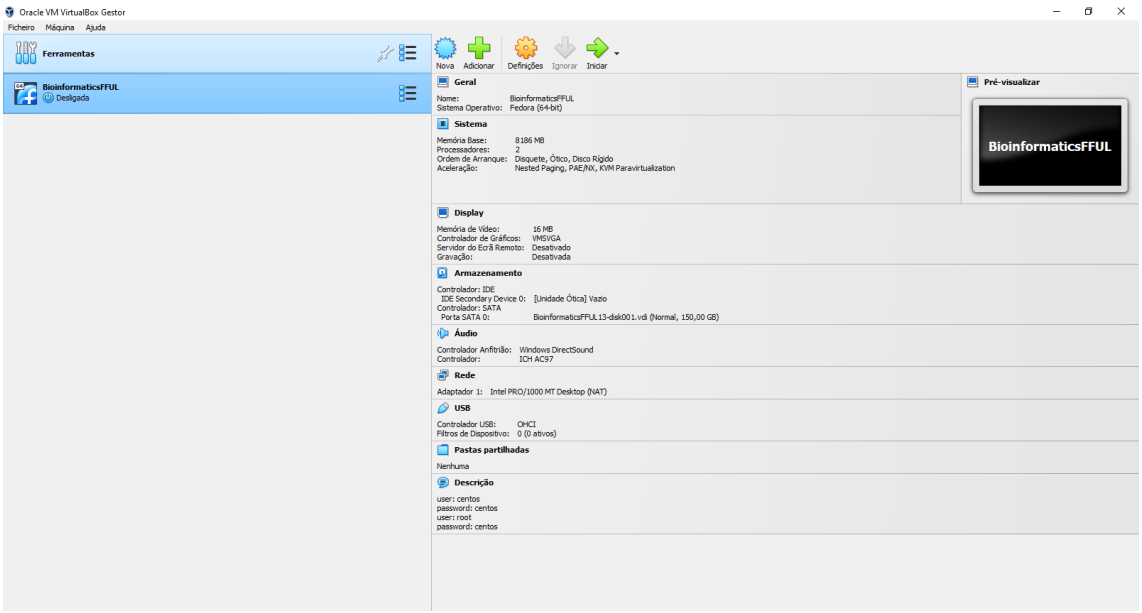
Import from a vdi or vmdk disk file:



Start VirtualBox and click *New*.



You can give it the name of your choice (e.g. BionformaticsFFUL), select Guest OS type as Fedora (64-bit). For memory size allocation it will depend on your system. For this course we recommend a minimum of 3000 Mb. After clicking on *Import* this VM will appear on VirtualBox VM list. To start it just select it and click *Start*.



About the Course VM

As mentioned above the course VM runs on a CentOS Stream Linux distribution. It comes with a GNOME Graphical User Interface (GUI), which is a free and open-source desktop environment that runs on Linux distributions.

This CentOS installation is configured to log in automatically with user centos. The user passwords are the following:

User: centos

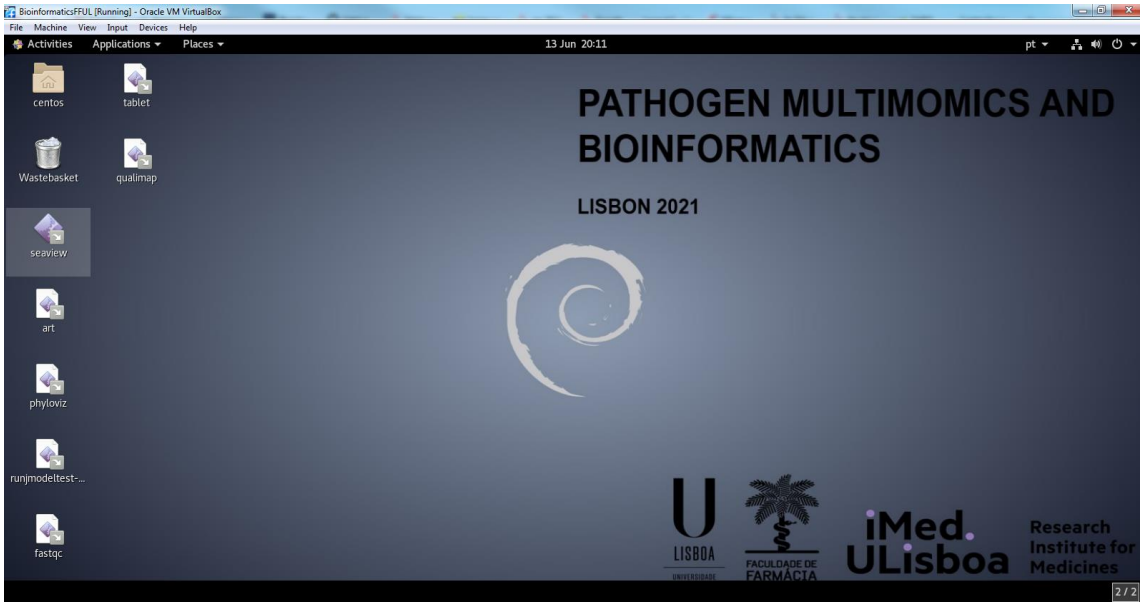
Password: centos

User: root

Password: centos

root is the super-user and it won't be necessary to use this account during the course. It may eventually become necessary if you deploy this VM onto your own computer and wish to undertake some system changes.

After logging in you will see the following desktop or a similar one:



Notice the desktop is already populated with some of the software that you will use over the course. Also, there are direct shortcuts to:

- your *home* directory folder (centos user *home* directory in this case) (Path: /home/centos/).
- open up a terminal/console window. Every time you open a new window from this shortcut it will open on your home directory.

You will find the course files within your home directory in the following folders/directories:

- **Module1** – Module 1 working directory with some files already present;
- **Module2** - Module 2 working directory with some files already present;
- **Module3** - Module 3 working directory with some files already present;
- **Module4** - Module 4 working directory with some files already present;
- **Module5** - Module 5 working directory with some files already present;
- **Module6**- Module 6 working directory with some files already present;
- **course_files** – Contains the FASTQ files to be used over the course;

Under the menu *Applications* in the desktop top bar you will find links to additional software and tools already installed (*e.g.*, Libre Office [an open-source alternative to Microsoft Office] or System Monitor).

:: Linux Command-line Basics ::

What is Linux?

During the 70's programmers from the Bell laboratories (AT&T) developed a new Operating System (UNIX) to overcome the existing limitations of the Operating Systems and Programming Languages of that time. Although initially used only for internal use at AT&T, universities and research centers became increasingly interested in this new OS with AT&T making available its source code. This enabled this new OS to expand and new extensions and tools to de developed by the scientific community.

However, AT&T started to restrict the licensing of UNIX and in order to protect the free software a new foundation called Free Software Foundation was created along with a new licence: General Public Licence (GPL). And from this new foundation a new UNIX-based OS was born: GNU (which means GNU is Not UNIX). In the 90's, Linus Torvalds inspired by the GNU project developed a new OS kernel: Linux, fully compatible with UNIX and GNU environments. Although quite rudimentary and incomplete, at its beginning Linux became increasingly popular, reaching one hundred new users in its first week. Over the last decades since its creation it has become a very robust and sophisticated OS and is currently at the same level of other commercial OSs, if not beyond.

Why on earth should I use Linux and the command-line?

First, it's free! And, being an open-source OS, each user can be a contributor to Linux development thereby increasing its reliability and stability since any error or bug can be readily corrected and incorporated in future releases.

Two of the most important Linux features are that this is a multitask and multiuser OS. Multiple users can be logged in simultaneously and can start multiple tasks/programs – called processes. Furthermore, each user is prevented from interfering with another user's work.

Presently, many Linux distributions already come bundled with graphical desktop environments (e.g. GNOME or KDE) but the command-line provides the user with a more flexible and enhanced control of the system. In the beginning it may be hard to know the commands but as the user gets familiar and learns how to use the command-line, it is possible to achieve a performance that is superior to the graphical interfaces.

Do I have to memorize all the commands?

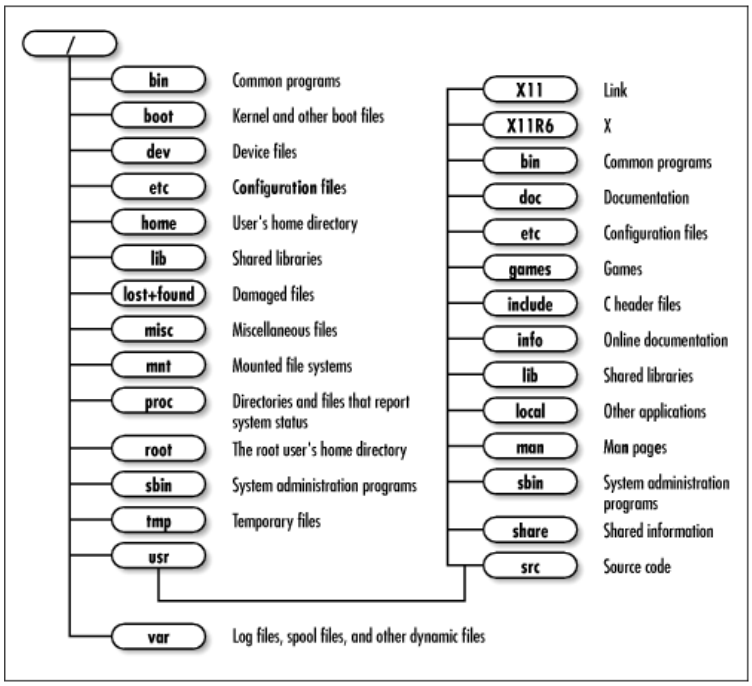
Actually no! Some of the commands are essential while others you will probably never use them. Linux comes with a plethora of basic commands and programs that you won't even know they are there. You'll end up by naturally assimilate this "new language" as you use these commands and if you don't recall any specific command, just look it up on the textbooks, online on webpages or the extensive community support forums or, in this document – that's what those are for!

Filesystem

As in windows or MacOS, files are organized in folders or directories where each directory can have multiple sub-directories and so on. The main directory (/) is called **root** and contains all directories and folders in the system. Each user only has access to its home directory unless if granted access to other directories or has administrative privileges. Each user home directory is located on /home/[user login] (in this course VM: /home/centos).

There is a special user: the **super-user** called **root**, because it has access to the root directory.

Take a look at an example of a Filesystem hierarchy:



The command-line

You can access to the command-line by opening a terminal window using the shortcut on your desktop. It is called command-line because the user must write commands in the form of text. Once you open the command-line interface you will see an initial message looking like this:

```
[centos@localhost ~]$
```

This is called the prompt and it tells you that the system is waiting for commands. It also tells you that you are logged in as user *centos* at (@) the *localhost* (the machine name you are using) and you are at your home directory (~). This is important because you can use the command-line to switch to another user and to connect to another machine.

Basic commands

You can start by obtaining a list of the files and sub-directories that exist in your current directory by typing:

```
$ ls
```

```
[centos@localhost ~]$ ls
commands.txt          Downloads   Music             Public       test.sh
course_documentation  files_copy  NC000962_3.fasta  R            Videos
course_files          Module1     NC000962_3.gbk    RAST_output
Desktop               Module2     other             Templates
Documents             Module3     Pictures          test.Rexec
[centos@localhost ~]$
```

Or if you want to obtain a more detailed list (long) list you can type:

```
$ ls -l
```

```
[centos@localhost ~]$ ls -l
total 17544
-rw-rw-r--. 1 centos centos     5429 Sep  1 18:51 commands.txt
drwxrwxr-x. 2 centos centos        6 Sep 11 10:22 course_documentation
drwx------. 2 centos centos     4096 Aug 19 18:45 course_files
drwxr-xr-x. 2 centos centos     4096 Sep  7 11:26 Desktop
drwxr-xr-x. 2 centos centos        6 Nov  1  2014 Documents
drwxr-xr-x. 5 centos centos     4096 Sep 11 10:26 Downloads
drwxrwxr-x. 2 centos centos     4096 Aug 19 18:29 files_copy
drwxrwxr-x. 3 centos centos       61 Sep  8 14:14 Module1
drwxrwxr-x. 3 centos centos     4096 Sep 11 10:23 Module2
drwxrwxr-x. 6 centos centos       66 Sep 14 16:39 Module3
drwxr-xr-x. 2 centos centos        6 Nov  1  2014 Music
-rw-r--r--. 1 centos centos  4474567 Jul  6 13:25 NC000962_3.fasta
-rw-r--r--. 1 centos centos 13443692 Jul 23  2013 NC000962_3.gbk
drwxrwxr-x. 5 centos centos       35 Sep 11 10:31 other
drwxr-xr-x. 2 centos centos     4096 Sep  7 11:36 Pictures
drwxr-xr-x. 2 centos centos        6 Nov  1  2014 Public
drwxrwxr-x. 3 centos centos       44 Aug 16 23:38 R
drwxrwxr-x. 2 centos centos        6 Sep  8 17:54 RAST_output
drwxr-xr-x. 2 centos centos        6 Nov  1  2014 Templates
-rwxrwxr-x. 1 centos centos       80 Aug 17 02:20 test.Rexec
-rwxrwxrwx. 1 centos centos       80 Aug 17 02:21 test.sh
drwxr-xr-x. 2 centos centos        6 Nov  1  2014 Videos
[centos@localhost ~]$
```

Important note: the commands are case-sensitive so Ls or LS will not work. Keep this always in mind!!

This more exhaustive list gives a number of details that we will not cover here but notice the modification date and time and size in bytes. Notice that NC000962_3.gbk file has a size of 13443692 bytes. It seems a lot but it is only 13.4 Mb!

Now, how do we find the full path of the directory where we are working? Just type:

```
$ pwd
```

It means print working directory outputs this:

```
[centos@localhost ~]$ pwd
/home/centos
```

What if I want to change directory:

```
$ cd Module1
```

cd means change directory and you can use the full path of a directory to change to a distant directory from where you are (e.g. cd /home/centos/Module1/vcfs). When you don't use the full path, cd assumes that the directory you are specifying exists in the present working directory. Now type pwd. What do you see?

```
$ pwd
```

```
[centos@localhost Module1]$ pwd
/home/centos/Module1
```

To go back type:

```
$ cd ..
```

On the command-line, two dots (..) means the directory above while one dot (.) means the current directory. That is why if you type `cd .` it will not get you anywhere but the present directory.

File manipulation

Let's say we wish to manipulate files and directories. To create a new directory type:

```
$ mkdir test_dir
```

To remove it:

```
$ rmdir test_dir
```

If the directory contains files rmdir will not work,you have to type:

```
$ rm -r test_dir
```

Which is more drastic. Try this and use the cd or ls command to go inside and see your directory.

And what about files?

You can copy (cp command) or move (mv command) files easily from the command-line:

```
$ cp NC000962_3.fasta Documents
```

This command copied NC000962_3.fasta file to your Documents directory. To avoid errors youl would write it in the following format:

```
$ cp ./NC000962_3.fasta ./Documents/
```

That way it is more intelligible that you are copying it to a folder. But here you are not specifying if you want the copied file to remain with the same name in the destination folder, the prompt assumes that it should stick with the same name. Otherwise:

```
$ cp ./NC000962_3.fasta ./Documents/test.fasta
```

The mv command works approximately in the same way, except that it deletes the original file.

To remove a file just type:

```
$ rm ./Documents/test.fasta
```

Can we visualize text files on the command-line? Sure, let's use the cat command (from your home directory, to go there from anywhere type cd ~) and type:

```
$ cat NC000962_3.fasta
```

Did you see it all? Probably not. By the way, cat takes multiple files and can concatenate those, hence the name cat. You can use CTRL+S and CTRL+Q to stop the scroll or, scroll up using SHIFT+PgUp or SHIFT+PgDn when the listing stops.

To see a file gradually with stoppings, type:

```
$ more NC000962_3.fasta
```

If you press ENTER or SPACE it will scroll down, to exit just press q.

Now let's stop for a moment and introduce three other characters:

- >, redirects the output to a file and erases a previously existing file if it has the same name;
- >>, redirects the output but appends it to the end of an existing file;

- |, named pipe character, does exactly that, pipes or channels the output of a command to another.

We can, for example, do:

```
$ ls -l > list.txt
```

This created a new file (list.txt). Let's read the content:

```
$ more list.txt
```

Is it familiar?

And if you do this:

```
$ ls -l >> list.txt
$ more list.txt
```

What now?

Wild-cards

To end this basic Linux tutorial. We'll introduce wild-cards and regular expressions. Wild-cards are special characters that enable you to call many files recursively. These are:

Character	Meaning
*	Any sequence of one or more characters
?	Any single character
[]	A sequence of one or more characters containing the characters within brackets

Let's use wild-cards to list files that only start with NC:

```
$ ls -l NC*
```

Or if we only want fast files:

```
$ ls -l *.fasta
```

You can use wild-cards on file operations as well:

```
$ cp ./NC* ./Documents/
```

What happened? Do you notice you have copied both files to the documents directories?

Let's delete those:

```
$ rm ./Documents/NC*
```

You just deleted two files with a single command. Be aware of the risks of using wild-cards as you can easily delete thousands of files with a single command. But, notice the power that comes with the command-line.

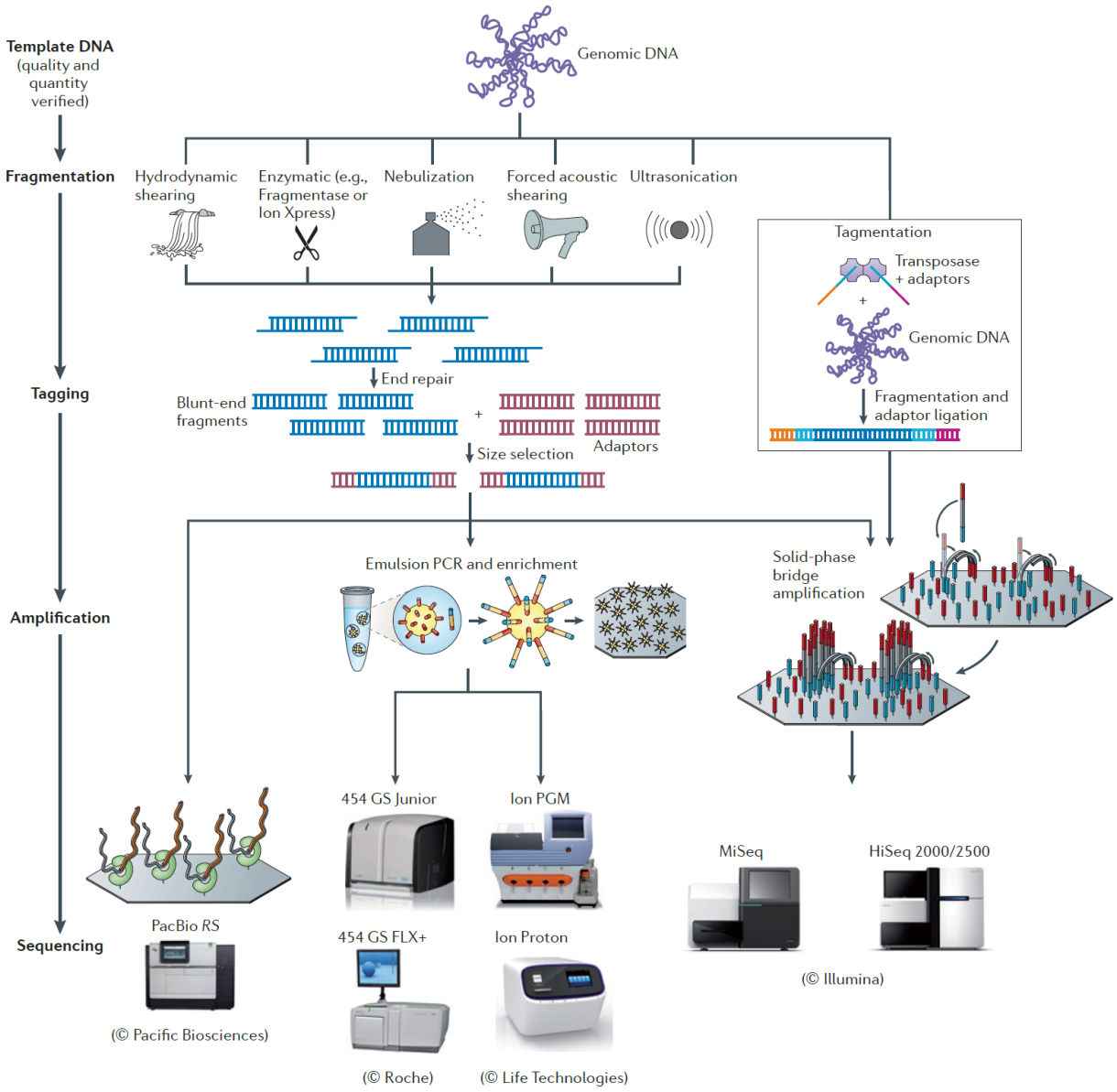
Hope this basic introduction is helpful for the following course Modules.

:: Mapping Sequence Data ::

Introduction

A wide array of Next Generation Sequencing platforms are available nowadays and, depending on the platform, a single machine can output a total of 6000 Gbp in approximately 40h (the equivalent to approximately 937 human diploid genomes). Moreover, this data is produced in the form of short or long **sequencing reads**, also depending on the platform [1]. Given the ability of these machines to produce millions of reads from a single organism, the process of arranging these to find overlaps and delineate contigs (genome assembly) is computationally demanding. Long read lengths are more adequate for genome assembly whereas the assembly of a genome using short read lengths may prove challenging particularly due to repetitive regions. This challenge can be partially overcomed using a reference genome for the organism being studied, in which the sequencing reads are aligned with this reference genome. This procedure is often called **reference assembly** or **mapping**. This designation contrasts with the procedure in which sequencing reads are assembly without a reference genome – ***de novo* assembly.** An important aspect for mapping that should be taken in account is that the reference genome should be a high quality well assembled genome [1].

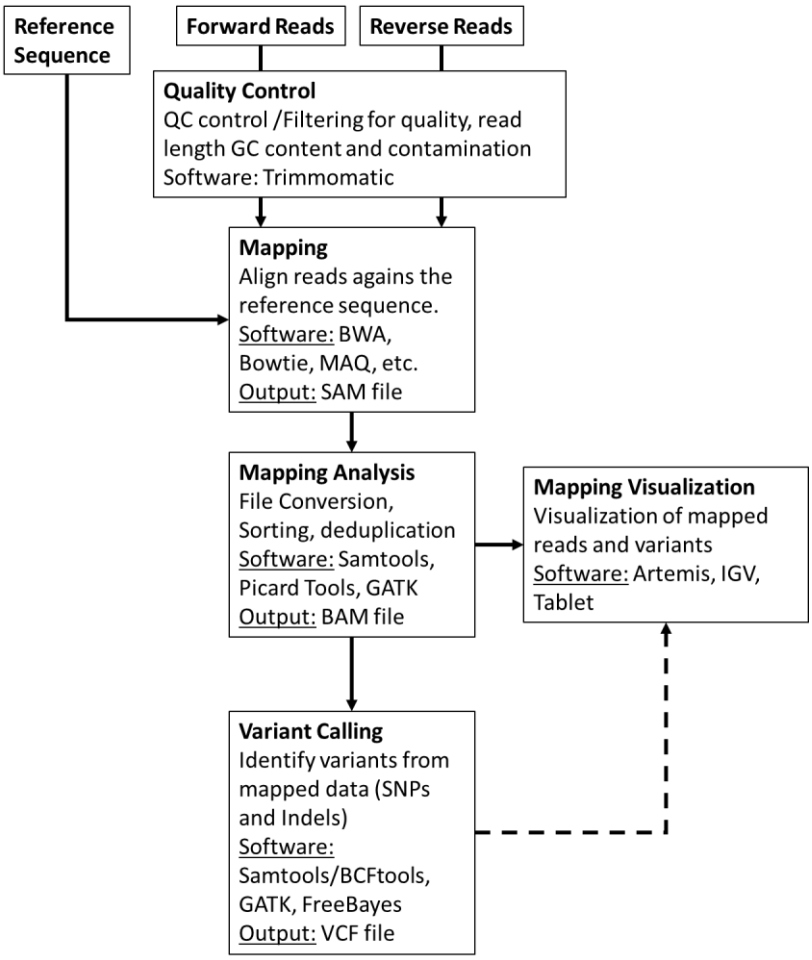
This module will address the basic bioinformatic analytical steps underlying mapping/reference assembly of NGS data. We will mainly focus on the analysis of sequence data obtained from Illumina platforms, that presently dominates 80-90% of the market, sequenced in paired-end mode. **What does "paired-end mode" mean?** It is important to first start by understanding the process of Illumina "sequencing-by-synthesis" methodology: starting with genomic DNA, it is randomly sheared and an appropriate length is selected after fractioning by agarose gel electrophoresis, followed by adapter ligation and solid phase PCR amplification (bridge amplification) on a sequencing flow cell [1]. Afertwards, sequencing by synthesis is carried out using reversible fluorescently-labelled terminator nucleotides where each fragment is sequenced on both ends producing two mate reads for each fragment (hence, **paired-end sequencing**).



(Loman *et al*, 2012)

The ensuing computational or in silico analysis picks up from the FASTQ files produced after adapter removal. The two main approaches at this point are as mentioned, **Mapping** and ***De novo* Assembly**. In this module we will focus on **Mapping** sequence reads obtained from *M. tuberculosis* clinical isolates collected in Portugal. By mapping sequence reads to a reference genome it will enable the downstream identification of Single Nucleotide Polymorphisms (SNPs), insertions and deletions (indels) and copy number variants (CNVs) that exist between the two organisms. Comparative Genomics underpinned on mapping analysis is also possible if the reference sequence remains the same.

The general workflow for this analysis consists in mapping the reads against a reference sequence using a mapper/aligner software to produce a mapping file (SAM/BAM) that can be further analysed and sorted using programs such as Picard Tools/Genome Analysis Toolkit or Samtools. Afterwards, it is possible to visualize the alignments using appropriate software with graphical interface and perform additional downstream analysis such as variant calling.



First, let's have a quick look at the FASTQ file format (Exercise 1).

Exercise 1 – Understanding the FASTQ file format

Let's have a look at the files made available for this course. Under the home directory there is a another directory called fastq_files.

Let´s access this folder: open up a terminal and type:

```
$ cd course_files
```

You can list its contents by typing ls (list command) which should give you the list of files and sub-directories within. You'll find a fasta file (NC000962_3.fasta) for the M. tuberculosis H37Rv reference genome (GenBank Acc. NC000962.3) and ten compressed fastq files for five different M. tuberculosis clinical isolates:

- PT000033: PT000033_1.fastq.gz and PT000033_2.fastq.gz
- PT000049: PT000049_1.fastq.gz and PT000049_2.fastq.gz
- PT000050: PT000050_1.fastq.gz and PT000050_2.fastq.gz
- PT000271: PT000271_1.fastq.gz and PT000271_2.fastq.gz
- PT000279: PT000279_1.fastq.gz and PT000279_2.fastq.gz

Some of the programs we will use ahead can deal directly with fastq compressed files but let us decompress the PT000033 files while keeping the compressed files unchanged:

```
$ gzip -dc PT000033_1.fastq.gz > PT000033_1.fastq

$ gzip -dc PT000033_2.fastq.gz > PT000033_2.fastq

# Next, let's copy these to the Module1 directory so that these are
available for the next parts but let's change first to the Module 1
directory:

$ cd ../Module1

$ cp ../course_files/PT000033_*.fastq .
```

The -d option indicates that it is a decompressing operation and -c option redirects the output to the file we indicate after >.

Let's use this strain for the following exercises. By the way, you can learn more about these strains on htttp://cplp-tb.ff.ulisboa.pt.

```
$ more NC000962_3.fasta
```

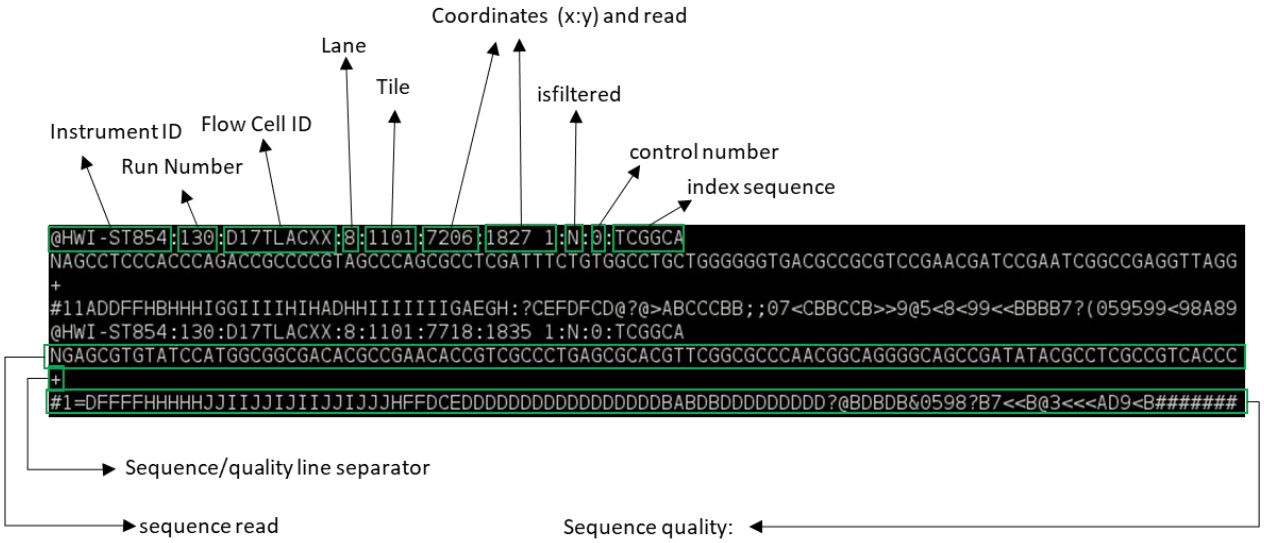
Now let's look at the file content and format. Let's start by looking at the reference fasta file.

Now let us compare the FASTA format with the FASTQ format:

```
$ more PT000033_1.fastq
```

What are the differences?

The FASTQ format:



ASCII_BASE=33 Illumina, Ion Torrent, PacBio and Sanger

Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII
0	1.00000	33 !	11	0.07943	44 ,	22	0.00631	55 7	33	0.00050	66 B
1	0.79433	34 "	12	0.06310	45 -	23	0.00501	56 8	34	0.00040	67 C
2	0.63096	35 #	13	0.05012	46 .	24	0.00398	57 9	35	0.00032	68 D
3	0.50119	36 $	14	0.03981	47 /	25	0.00316	58 :	36	0.00025	69 E
4	0.39811	37 %	15	0.03162	48 0	26	0.00251	59 ;	37	0.00020	70 F
5	0.31623	38 &	16	0.02512	49 1	27	0.00200	60 <	38	0.00016	71 G
6	0.25119	39 '	17	0.01995	50 2	28	0.00158	61 =	39	0.00013	72 H
7	0.19953	40 (	18	0.01585	51 3	29	0.00126	62 >	40	0.00010	73 I
8	0.15849	41)	19	0.01259	52 4	30	0.00100	63 ?	41	0.00008	74 J
9	0.12589	42 *	20	0.01000	53 5	31	0.00079	64 @	42	0.00006	75 K
10	0.10000	43 +	21	0.00794	54 6	32	0.00063	65 A			

Let's take a look at the beginning and end of the bottom sequence in the figure above. What are the Phred33 Q scores compared to the rest of the sequence?

How we deal with this problem will be topic of **the next exercise**: **Read Quality Control**!!

Exercise 2 – Read Quality Control

Before introducing sequencing reads into an analytical pipeline it is highly important to check the overall quality of the reads through the evaluation of the percentage of low quality bases, contamination or length. The impact of low quality or base calling errors can be minimised in downstream applications by applying some filters where low quality reads or bases are removed if these do not meet thresholds defined by the user [2, 3].

Let's start by analysing the reads from the previous exercise (PT000033). These have been put in the Module1 directory. To assess the quality of these reads we will use a Java written software named FastQC that provides a user-friendly way to carry out some quality control checks on raw sequence data.

To start, double-click on the FastQC shortcut available on the desktop. This will open the FastQC graphical interface. Alternatively, you can start the FastQC from the command prompt by typing:

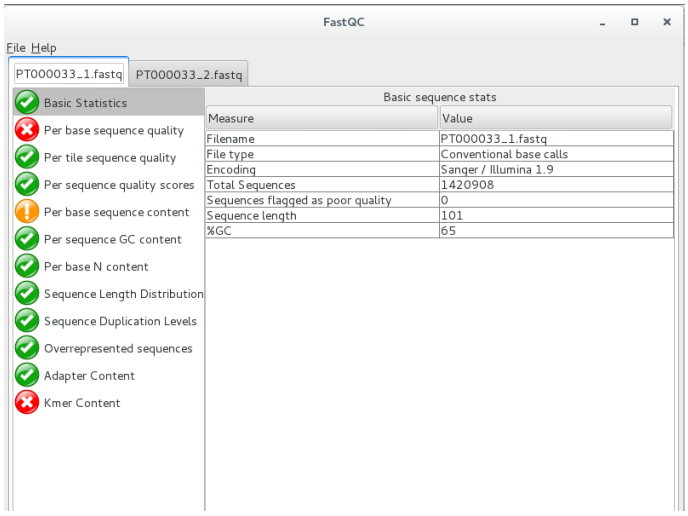
```
$ fastqc
```

Go to File>Open and then select the two reads for PT000033 strain present in the Module1 directory. This will start the analysis. After analysis is completed you can examine the data for each read file showing up on different tabs.

Go ahead and take a look at the different parameters that are evaluated. Normal QC tests will show up with a green tick, slightly abnormal with an exclamation mark in orange, and abnormal results with a cross highlighted in red. What problems seem to be affecting these reads?

Let's have a look at the QC parameters:

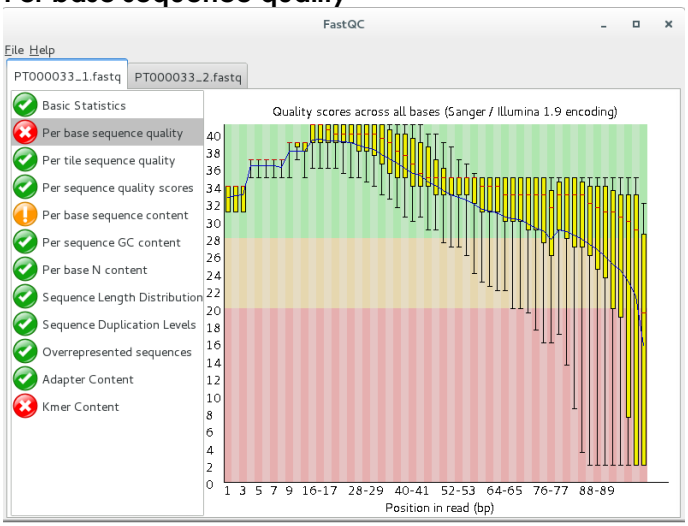
Basic Statistics



The Basic Statistics module generates some simple and self-explanatory composition statistics for the file analysed:

- Filename
- File type
- Encoding: Says which ASCII encoding of quality values was found in this file.
- Total Sequences
- Filtered Sequences
- Sequence Length
- %GC

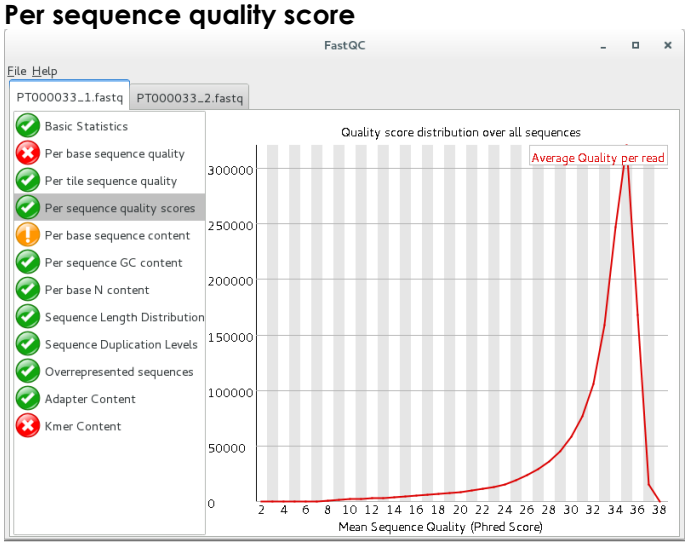
Per base sequence quality



Overview of the range of quality values across all bases at each position in the FastQ file. In general sequencing chemistry degrades with increasing read length and for long runs you may find that the general quality of the run falls to a level where a warning or error is triggered.

Warning: the lower quartile for any base <10, or median for any base < 25.Failure: lower quartile for any base is less than 5 or if the median for any base is less than 20. Phred33 scores below 20 are considered to be an indicator of poor quality.

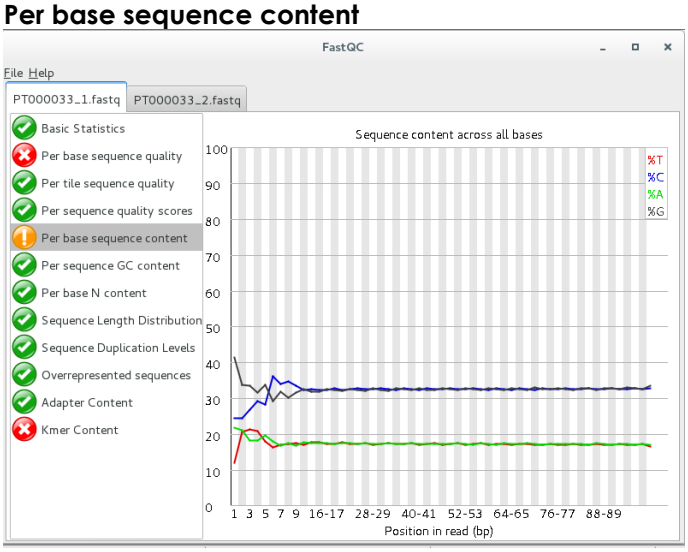
Per sequence quality score



The per sequence quality score report allows you to see if a subset of your sequences have universally low quality values. It is often the case that a subset of sequences will have universally poor quality, often because they are poorly imaged, however these should represent only a small percentage of the total sequences.

A warning is raised if the most frequently observed mean quality is below 27 - this equates to a 0.2% error rate. An error is raised if the most frequently observed mean quality is below 20 - this equates to a 1% error rate.

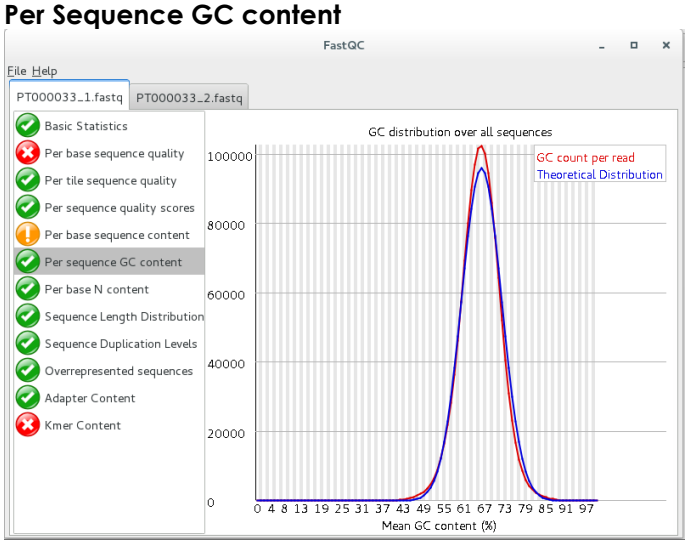
Per base sequence content



Per Base Sequence Content plots out the proportion of each base position in a file for which each of the four normal DNA bases has been called.

This module issues a warning if the difference between A and T, or G and C is greater than 10% in any position. This module will fail if the difference between A and T, or G and C is greater than 20% in any position.

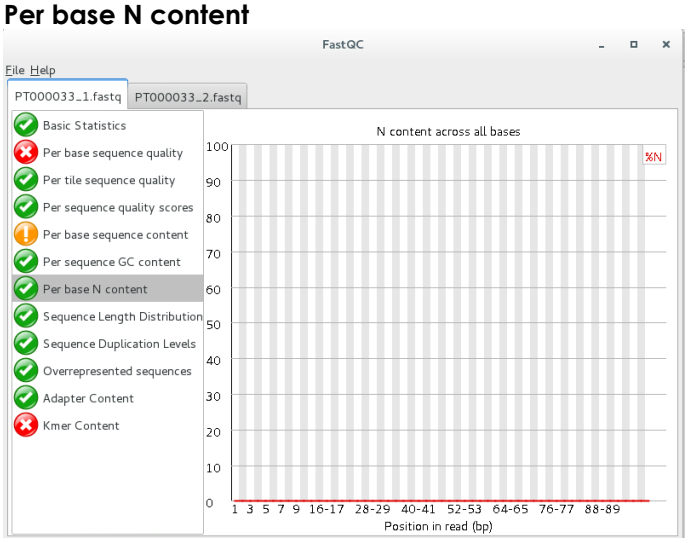
Per Sequence GC content



This module measures the GC content across the whole length of each sequence in a file and compares it to a modelled normal distribution of GC content.An unusually shaped distribution could indicate a contaminated library or some other kinds of biased subset. A normal distribution which is shifted indicates some systematic bias which is independent of base position. If there is a systematic bias which creates a shifted normal distribution then this won't be flagged as an error by the module since it doesn't know what your genome's GC content should be.

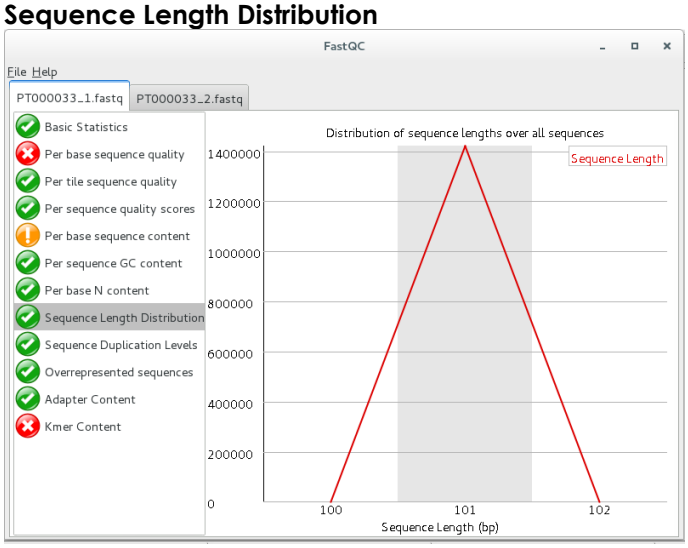
A warning is raised if the sum of the deviations from the normal distribution represents more than 15% of the reads. This module will indicate a failure if the sum of the deviations from the normal distribution represents more than 30% of the reads.

Per base N content



This module plots out the percentage of base calls at each position for which an N (uncalled base) was called. If the N proportion rises above a few percent it suggests that the analysis pipeline was unable to interpret the data well enough to make valid base calls. This module raises a warning if any position shows an N content of >5%. This module will raise an error if any position shows an N content of >20%.

Sequence Length Distribution

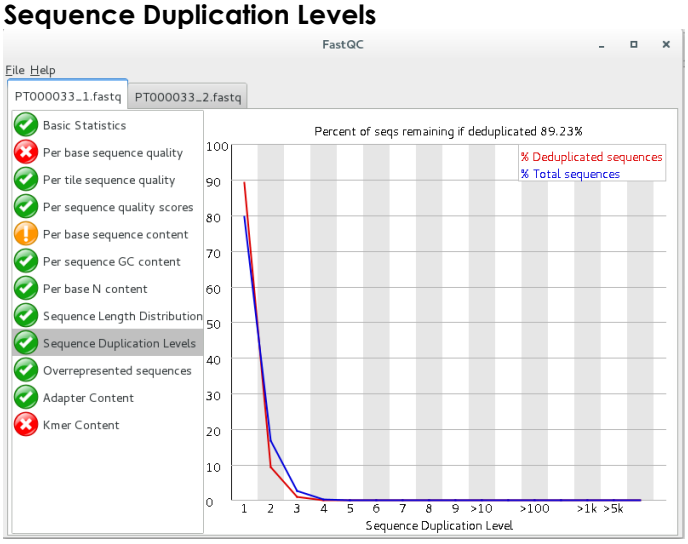


This module generates a graph showing the distribution of fragment sizes in the file which was analysed. In many cases this will produce a simple graph showing a peak only at one size, but for variable length FastQ files this will show the relative amounts of each different size of sequence fragment.

This module will raise a warning if all sequences are not the same length. This module will raise an error if any of the sequences have zero length.

For some sequencing platforms, it is entirely normal to have different read lengths so warnings here can be ignored.

Sequence Duplication Levels

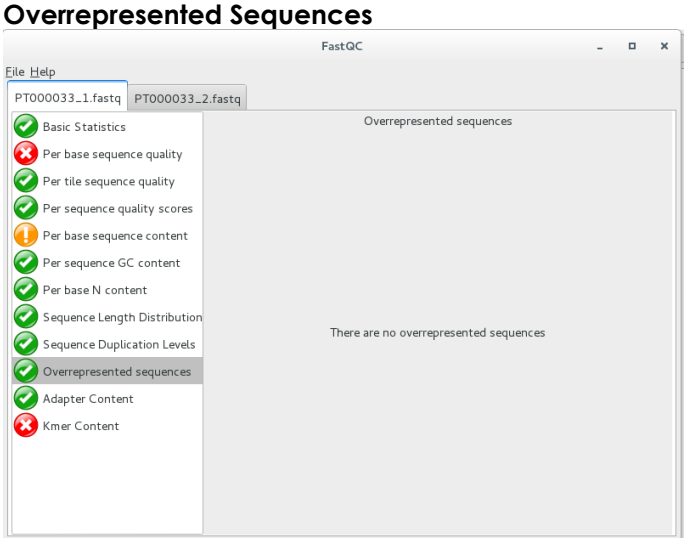


In a diverse library most sequences will occur only once in the final set. A low level of duplication may indicate a very high level of coverage of the target sequence, but a high level of duplication is more likely to indicate some kind of enrichment bias (eg PCR over amplification).

This module counts the degree of duplication for every sequence in a library and creates a plot showing the relative number of sequences with different degrees of duplication.

This module will issue a warning if non-unique sequences make up more than 20% of the total. This module will issue a error if non-unique sequences make up more than 50% of the total.

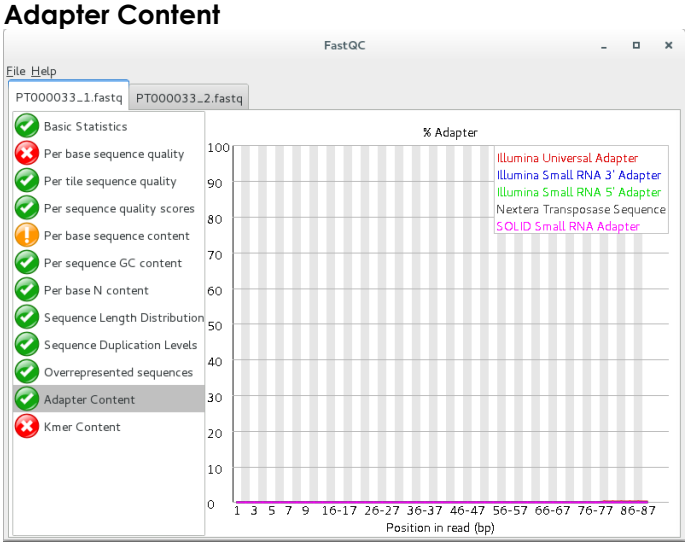
Overrepresented Sequences



A normal high-throughput library will contain a diverse set of sequences, with no individual sequence making up a tiny fraction of the whole. Finding that a single sequence is very overrepresented in the set either means that it is highly biologically significant, or indicates that the library is contaminated, or not as diverse as you expected. For each overrepresented sequence the program will look for matches in a database of common contaminants and will report the best hit it finds. It's also worth pointing out that many adapter sequences are very similar to each other so you may get a hit reported which isn't technically correct.

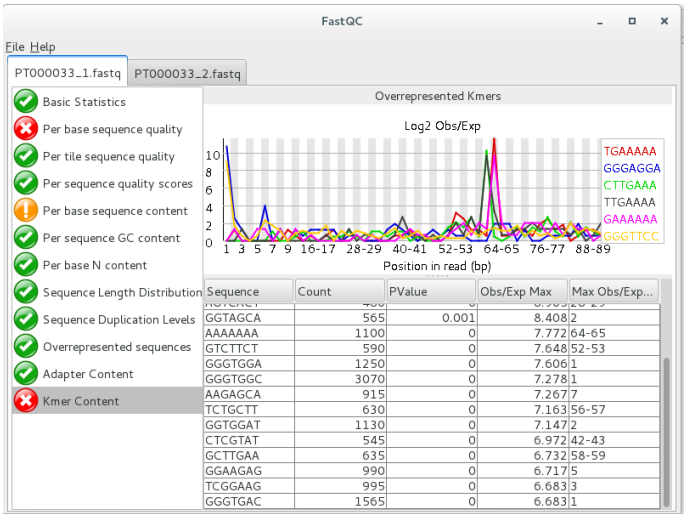
This module will issue a warning if any sequence is found to represent more than 0.1% of the total. This module will issue an error if any sequence is found to represent more than 1% of the total.

Adapter Content



One obvious class of sequences which you might want to analyse are adapter sequences. It is useful to know if your library contains a significant amount of adapter in order to be able to assess whether you need to adapter trim or not. Although the Kmer analysis can theoretically spot this kind of contamination it isn't always clear. This module therefore does a specific search for a set of separately defined Kmers and will give you a view of the total proportion of your library which contain these Kmers. A results trace will always be generated for all of the sequences present in the adapter config file so you can see the adapter content of your library, even if it's low. This module will issue a warning if any sequence is present in more than 5% of all reads. This module will issue a warning if any sequence is present in more than 10% of all reads.

Kmer Content



The analysis of overrepresented sequences will spot an increase in any exactly duplicated sequences, but there are a different subset of problems where it will not work: if you have very long sequences with poor sequence quality then random sequencing; and if you have a partial sequence which is appearing at a variety of places within your sequence then this won't be seen either by the per base content plot or the duplicate sequence analysis. The Kmer module starts from the assumption that any small fragment of sequence should not have a positional bias in its appearance within a diverse library. Any Kmers with positionally biased enrichment are reported. The top 6 most biased Kmer are additionally plotted to show their distribution.

For your reference: it is possible to save the report for each analysis. Just go to File>Save Report… and you will be able to save an html file containing the analysis displayed in FastQC.

Adapted from https://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3 Analysis Modules/.

Exercise 3 – Sequence Read Trimming

Improving the quality of the read files is an essential step that prevents the emergence of errors in downstream steps such as erroneously mapped reads or incorrectly called variants [3]. We will be using a Java written command-line tool that is designed to trim and crop Illumina FASTQ files and can also be used to remove adapters: **Trimmomatic** [2]. This tool also has the advantage of parallelization across multiple cores thereby increasing its execution speed in multi-processor machines (multithreading). Trimmomatic can operate on Single-end and Paired-end mode. We will be using the paired-end mode where the software will keep the correspondence between read mates.

Learn more about Trimmomatic:

- Bolger, A. M., Lohse, M., & Usadel, B. (2014). Trimmomatic: A flexible trimmer for Illumina Sequence Data. *Bioinformatics*, btu170.
- http://www.usadellab.org/cms/?page=trimmomatic

The Trimmomatic options are:

- ILLUMINACLIP: Cut adapter and other illumina-specific sequences from the read.
- SLIDINGWINDOW: Perform a sliding window trimming, cutting once the average quality within the window falls below a threshold.
- LEADING: Cut bases off the start of a read, if below a threshold quality
- TRAILING: Cut bases off the end of a read, if below a threshold quality
- CROP: Cut the read to a specified length
- HEADCROP: Cut the specified number of bases from the start of the read
- MINLEN: Drop the read if it is below a specified length
- TOPHRED33: Convert quality scores to Phred-33
- TOPHRED64: Convert quality scores to Phred-64

Let's get started, from your home directory go to the Module1 directory:

```
$ cd Module1

## Now let's start trimming PT000033 reads:

$ trimmomatic PE -phred33 PT000033_1.fastq PT000033_2.fastq
PT000033_1_trimmed_paired.fastq PT000033_1_trimmed_unpaired.fastq
PT000033_2_trimmed_paired.fastq PT000033_2_trimmed_unpaired.fastq
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:20 MINLEN:36
```


What files did the software generated? Rerun the FastQC analysis for the paired-end files (*_paired.fastq) and take a look…

Exercise 4 – Mapping

Time to start mapping! To align sequence reads to a reference genome we must first choose one of the many short-read aligner software that are currently available. These softwares are based on alignment algorithms that can cope with the millions of reads produced for a single organism by NGS platforms and, are more efficient in doing this than traditional aligning algorithms. Different algorithms exist for this task as well as different software implementations [4, 5]. For a thorough review on mapping algorithms and sequence read alignment you can check the following articles:

- Mielczarek M, Szyda J. Review of alignment and SNP calling algorithms for next-generation sequencing data. *J Appl Genet*. 2016; **57**: 71-79.
- Li H, Homer N. A survey of sequence alignment algorithms for next-generation sequencing. *Brief Bioinform*. 2010; **11**: 473-483.

Most well-known aligners include MAQ, Bowtie, BWA, etc. The output of most of these aligners is a file containing the alignment in the SAM/BAM format which is the most widely used format to store NGS mapped reads [6]. We will look at this format ahead. In this course we will use one of the most popular aligners – Burrows-Wheeler Aligner (BWA) [7]. BWA comprehends different three alignment algorithms: BWA-backtrack (aln and sampe/samse); BWA-SV; and BWA-MEM. For 70bp or longer Illumina, 454, Ion Torrent and Sanger reads, assembly contigs and BAC sequences, BWA-MEM is usually the preferred algorithm. For short sequences, BWA-backtrack may be better. BWA-SW may have better sensitivity when alignment gaps are frequent.

In this exercise we will use the BWA-MEM algorithm. We will now map the reads from strain PT000033 to the *M.tuberculosis* H37Rv genome. Let's start by open a terminal window and from your home directory go to Module1 directory:

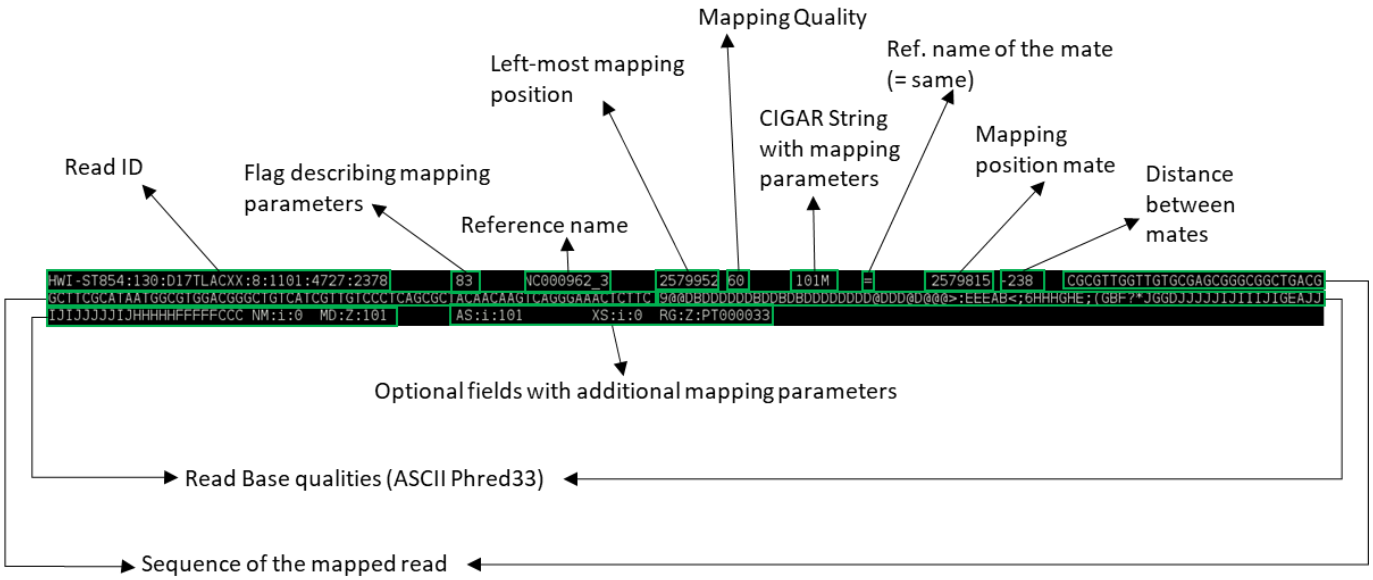
```
$ cd Module1

## First, we have to index the reference sequence (NC00962_3.fasta)
to allow efficient access by BWA upon mapping:

$ bwa index NC000962_3.fasta
```

```
## Next, we will map the trimmed paired reads obtained in the
previous exercise. We can also map the unpaired reads but we would
need to merge the resulting three SAM/BAM files. In this exercide
we will continue with the paired reads only:


$ bwa mem -R "@RG\tID:PT000033\tSM:PT000033\tPL:Illumina" -M
NC000962_3.fasta PT000033_1_trimmed_paired.fastq
PT000033_2_trimmed_paired.fastq > PT000033.sam


##The -R and -M option are included for compatibility with Picard
tools and GATK during variant calling. Although we will not use
these, the SAM file produced can be subsequently analysed by these
programs if necessary. The -R option specifies the read group
header line and the -M option is grants compatibility with Picard
tools.
```

The commands above should have generated a SAM file. Let's look at its content:

```
$ more PT000033.sam
```

This seems rather complex but it is in fact a flexible format to store the coordinates of mapped and unmapped reads, associated Phred33 Q scores and chromosome. See the figure below to have an idea of what it represents (the image below represents a single line):



You can also use the built-in word-count program in Linux to check the number of lines in this SAM file:

```
$ wc -l PT000033.sam
```

Now that we have a SAM file we can convert it to its compressed binary format (which cannot be read as a text file but can be read with Samtools view). We also need to sort the BAM file by coordinates to allow efficient access and analysis and index this BAM file. The BAM index file (*.bai) is required by some visualization and downstream analysis software, as it allows rapid access to the BAM file.

```
## We will again index the reference sequence, this time with
Samtools:

$ samtools faidx NC000962_3.fasta

## We can then convert the SAM file to a BAM file:

$ samtools view -bt NC000962_3.fasta.fai PT000033.sam >
PT000033.bam

## Sorting:

$ samtools sort -o PT000033.sorted.bam PT000033.bam

## Finally, we will index the sorted bam file (if posteriorly you
change the name of the BAM file it is necessary to re-index this
same file):

$ samtools index PT000033.sorted.bam
```

You can produce some basic mapping statistics by using samtools flagstat command:

```
$ samtools flagstat PT000033.sorted.bam

## This will output the result to standard output (screen), or you
can alternatively save it in a new file (PT000033_stats.txt):

$ samtools flagstat PT000033.sorted.bam > PT000033_stats.txt
```

At this point we have checked the quality of reads (QC), trimmed low-quality reads and bases and mapped the sequence reads to a reference genome. We can now proceed to variant calling but let's first visualize the mapped reads onto a reference chromosome.

Exercise 5 – Visualization of Mapped Data

There are several programs to visualize genomic data, including annotated genes and mapped reads: Artemis, Tablet, IGB, etc [8-10]. Artemis and Tablet are available in the Linux Operating System Image made available for this course. However, we will be using Artemis in this exercise. Artemis is a powerful visualization and annotation tool with multiple functionality, having the advantage of being a Java written platform and can therefore be used on Windows, MacOS or GNU/Linux platforms [9].

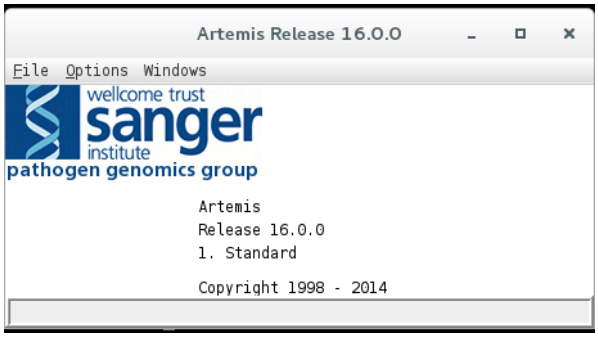
To use artemis we must first open a genome, this can be either in the format of a fasta file, such as the file containing the reference genome we used in the previous exercise or for example a GenBank file (*gb or *gbk). The latter has the advantage of containing the sequence data along with the annotation for genes and open reading frames (ORFs). It is also possible to open a FASTA file and subsequently read in entries from a GFF3 file (annotation). All these file types can be easily retrieved from GenBank (see send to file on: https://www.ncbi.nlm.nih.gov/nuccore/NC_000962; don't forget to check the Show sequence on the Display options).

On the Module1 directory the NC000962_3.gbk file is available. Please open it with Artemis.

To start Artemis either double-click on the shortcut on the desktop or open up a terminal window and type:

```
$ art
```

This will open an Artemis window:



Go to File>Open… > then select the NC000962_3.gbk and click OPEN. You can skip the warnings.

A new window will open:

The Artemis window has three main views. The first contains the plus and minus DNA strands along with their respective three reading frames. You can see genes annotated on these reading frames. The middle view is similar but zooming at the nucleotide level along with respective amino acids at each reading frame. The bottom view shows each

annotated feature, starting and ending genomic positions and, additional information on the gene function or similarity, etc., if available.

Before reading more data in, navigate around, you can use the horizontal scroll bar on the top view to move along the genome, and you can use the vertical scroll bar to zoom in and out. To move to regions more efficiently, click on Goto>Navigator… to open the navigator window. Try to search for a gene of your interest (e.g. *rpsL*). After selecting, note that you can view the gene in FASTA format by right-clicking on it and then View>Bases>Bases of Selection as FASTA. Similarly by doing the same thing but by starting with Create you can save the FASTA directly to a file of your choice.

You can also add additional plots by, e.g., going to Graphs>GC Content (%).

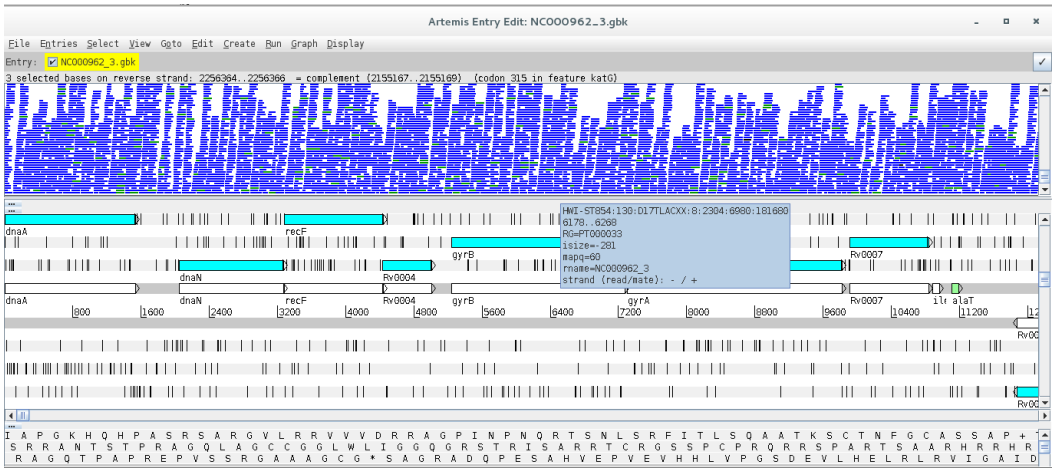
These are just some of the functionalities.

Tip: To select bases or amino acids inside a feature click the region of interest while pressing Alt.

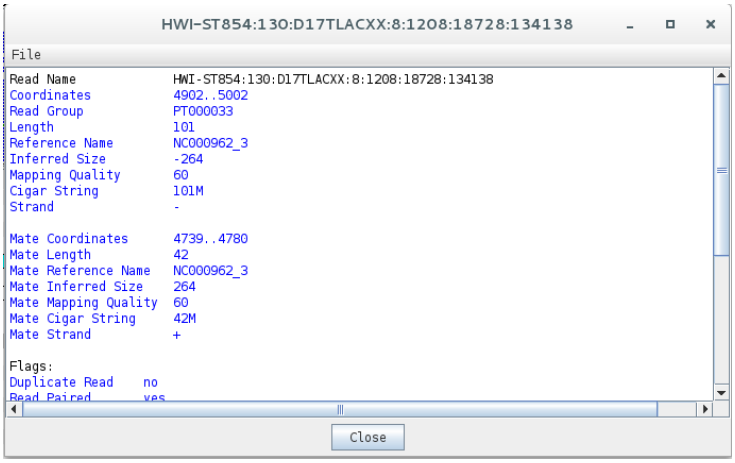
Next, it is time to look at your alignment file. Go to File>Read BAM/VCF and open the sorted BAM file you created in the previous exercise for the PT000033 strain.

(**Attention:** if the BAM file is not indexed and the bai file is not in same directory Artemis won't be able to open it)

Something like this should appear:



Now, there is a new view representing the mapping location of every successfully mapped read. Moreover, you can for example select any read right-click and then click the Show Details of the read. A new window will appear with data on the sequence read:

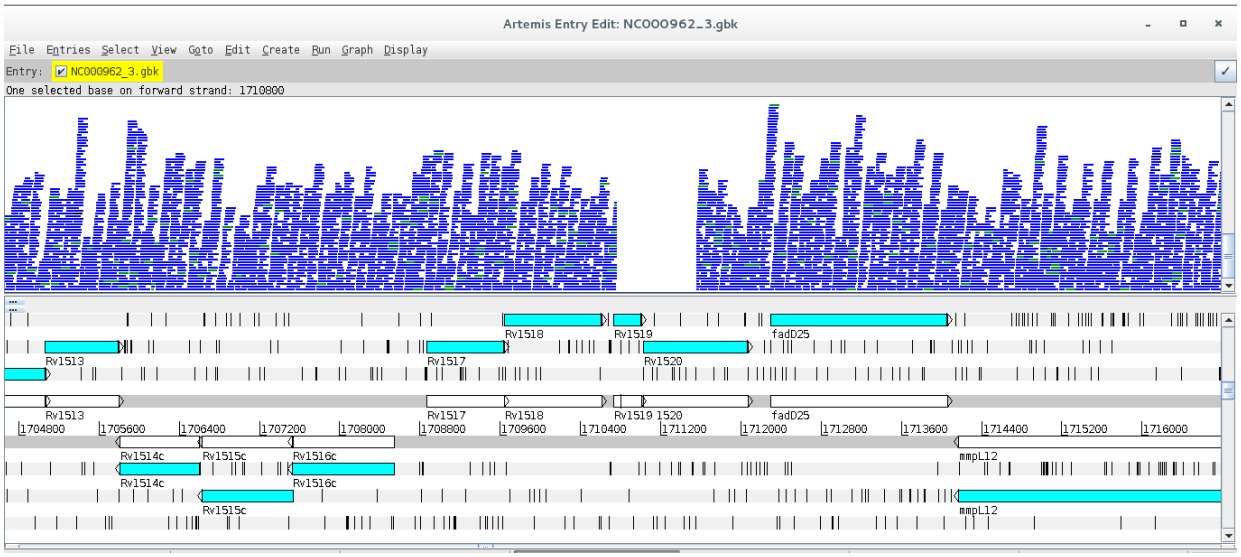


Does this information remind you of anything? Perhaps the data contained in the SAM file…

Another useful option is to look at coverage plots. Right click on the stacked view and select Views>Coverage. What happened?

Differences in coverage:

Let's again select the Stack view from the Views>Coverage. Next, using the Navigator window let's go the region around nucleotide 1710800. What is happening here?

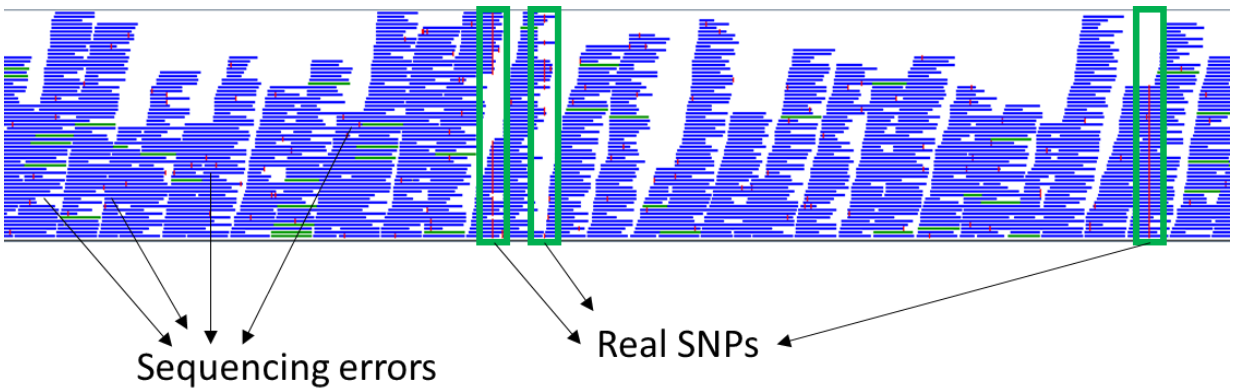


Now go to the article by Gagneux et al 2006 entitled "Variable Host-Pathogen Compatibility" and check Supplementary Table 4?

What can you conclude from this?

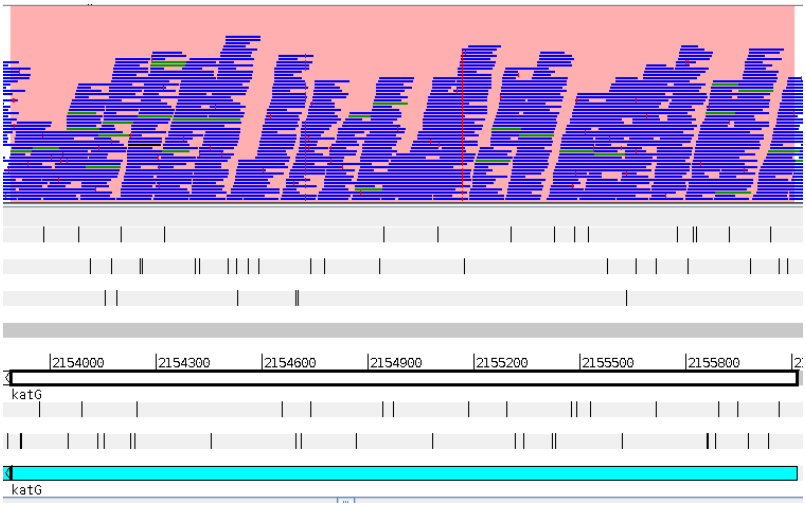
What about Single Nucleotide Polymorphisms?

Another important aspect is to look for SNPs. Return to the stack view and to the beginning of the genome. Now on the stack view righ-click Show>SNP Marks. Each SNP is highlighted by a red marking on the respective read. Take a look at it. Why are there so many red markings? How to distinguish those from the real SNPs?



Note that real SNPs consistently show up in almost all reads, whereas sequencing errors appear dispersed.

This strain is in fact monorresistant to Isoniazid (INH). To uncover the molecular basis of resistance in this isolate we can start by looking at the main gene associated with INH resistance: *katG*. Using the Navigator go to the katG gene. How many real SNPs can you detect and which one is associated with resistance?



Now, isn't there an easier way to detect and visualize SNP variants? Yes: **Variant Calling**.

Exercise 6 – Variant Calling

Different software packages are also available for variant calling. Three main programs are available: samtools/bcftools mpileup, GATK and FreeBayes [6, 11]. In this practical we will use samtools mpileup in combination with Bcftools for filtering variants. A good approach is also to use multiple callers and combining the data to produce more robust datasets. At this point, the parameters used by variant calling software play a crucial role in correct variant identification [3].

Open a terminal and type:

```
$ bcftools mpileup
```

The complete parameter listing for samtools mpilup should appear:

```
Usage: samtools mpileup [options] in1.bam [in2.bam [...]]

Input options:
  -6, --illumina1.3+      quality is in the Illumina-1.3+ encoding
  -A, --count-orphans     do not discard anomalous read pairs
  -b, --bam-list FILE     list of input BAM filenames, one per line
  -B, --no-BAQ            disable BAQ (per-Base Alignment Quality)
  -C, --adjust-MQ INT     adjust mapping quality; recommended:50, disable:0 [0]
  -d, --max-depth INT     max per-BAM depth; avoids excessive memory usage [250]
  -E, --redo-BAQ          recalculate BAQ on the fly, ignore existing BQs
  -f, --fasta-ref FILE    faidx indexed reference sequence file
  -G, --exclude-RG FILE   exclude read groups listed in FILE
  -l, --positions FILE    skip unlisted positions (chr pos) or regions (BED)
  -q, --min-MQ INT        skip alignments with mapQ smaller than INT [0]
  -Q, --min-BQ INT        skip bases with baseQ/BAQ smaller than INT [13]
  -r, --region REG        region in which pileup is generated
  -R, --ignore-RG         ignore RG tags (one BAM = one sample)
  --rf, --incl-flags STR|INT  required flags: skip reads with mask bits unset []
  --ff, --excl-flags STR|INT  filter flags: skip reads with mask bits set
                                [UNMAP,SECONDARY,QCFAIL,DUP]
  -x, --ignore-overlaps   disable read-pair overlap detection

Output options:
  -o, --output FILE       write output to FILE [standard output]
  -g, --BCF               generate genotype likelihoods in BCF format
  -v, --VCF               generate genotype likelihoods in VCF format

Output options for mpileup format (without -g/-v):
  -O, --output-BP         output base positions on reads
  -s, --output-MQ         output mapping quality

Output options for genotype likelihoods (when -g/-v is used):
  -t, --output-tags LIST  optional tags to output:
                DP,AD,ADF,ADR,SP,INFO/AD,INFO/ADF,INFO/ADR []
  -u, --uncompressed      generate uncompressed VCF/BCF output

SNP/INDEL genotype likelihoods options (effective with -g/-v):
  -e, --ext-prob INT      Phred-scaled gap extension seq error probability [20]
  -F, --gap-frac FLOAT    minimum fraction of gapped reads [0.002]
  -h, --tandem-qual INT   coefficient for homopolymer errors [100]
  -I, --skip-indels       do not perform indel calling
  -L, --max-idepth INT    maximum per-sample depth for INDEL calling [250]
  -m, --min-ireads INT    minimum number gapped reads for indel candidates [1]
  -o, --open-prob INT     Phred-scaled gap open seq error probability [40]
  -p, --per-sample-mF     apply -m and -F per-sample for increased sensitivity
  -P, --platforms STR     comma separated list of platforms for indels [all]
      --input-fmt-option OPT[=VAL]
                Specify a single input file format option in the form
                of OPTION or OPTION=VALUE
      --reference FILE
                Reference sequence FASTA FILE [null]

Notes: Assuming diploid individuals.
```

Traditionally, the samtools mpileup command would be used to generate VCF, BCF or pileup for one or multiple BAM files and the calling process would be completed by Bcftools [12]. More recently the mipelup function was transferred to bcftools to avoid incompatibility issued between samtools/bcftools version. It is still possible to use samtools in this way but we need to check for compatibility between versions. Therefore, we have to pipe (' | ') the output from bcftools [or samtools] mpileup to bcftools call. On a terminal window, type:

```
$ bcftools call
```

… to list bcftools call parameters and options:

```
Usage:  bcftools call [options] <in.vcf.gz>

File format options:
      --no-version              do not append version and command line to the header
  -o, --output <file>           write output to a file [standard output]
  -O, --output-type <b|u|z|v>   output type: 'b' compressed BCF; 'u' uncompressed BCF; 'z' compressed VCF; 'v' uncompressed VCF [v]
      --ploidy <assembly>[?]    predefined ploidy, 'list' to print available settings, append '?' for details
      --ploidy-file <file>      space/tab-delimited list of CHROM,FROM,TO,SEX,PLOIDY
  -r, --regions <region>        restrict to comma-separated list of regions
  -R, --regions-file <file>     restrict to regions listed in a file
  -s, --samples <list>          list of samples to include [all samples]
  -S, --samples-file <file>     PED file or a file with an optional column with sex (see man page for details) [all samples]
  -t, --targets <region>        similar to -r but streams rather than index-jumps
  -T, --targets-file <file>     similar to -R but streams rather than index-jumps
      --threads <int>           number of extra output compression threads [0]

Input/output options:
  -A, --keep-alts               keep all possible alternate alleles at variant sites
  -f, --format-fields <list>    output format fields: GQ,GP (lowercase allowed) []
  -F, --prior-freqs <AN,AC>     use prior allele frequencies
  -g, --gvcf <int>,[...]        group non-variant sites into gVCF blocks by minimum per-sample DP
  -i, --insert-missed           output also sites missed by mpileup but present in -T
  -M, --keep-masked-ref         keep sites with masked reference allele (REF=N)
  -V, --skip-variants <type>    skip indels/snps
  -v, --variants-only           output variant sites only

Consensus/variant calling options:
  -c, --consensus-caller        the original calling method (conflicts with -m)
  -C, --constrain <str>         one of: alleles, trio (see manual)
  -m, --multiallelic-caller     alternative model for multiallelic and rare-variant calling (conflicts with -c)
  -n, --novel-rate <float>,[...] likelihood of novel mutation for constrained trio calling, see man page for details [1e-8,1e-9,1e-9]
  -p, --pval-threshold <float>  variant if P(ref|D)<FLOAT with -c [0.5]
  -P, --prior <float>           mutation rate (use bigger for greater sensitivity), use with -m [1.1e-3]
```

Let's start the variant calling process, type the following commands in the Module1 directory:

```
$ bcftools mpileup -Q 23 -d 2000 -f NC000962_3.fasta
PT000033.sorted.bam | bcftools call -O v -vm -o PT000033.raw.vcf

## Notice that you have set the -Q option to skip aligned bases
with quality below 23 and set the maximum read depth to 2000.

$ vcfutils.pl varFilter -d 10 PT000033.raw.vcf > PT000033.filt.vcf

## Here, we used the vcfutils.pl perl script from BCFtools to
filter the raw vcf file with the -d option to set the minimum read
depth of 10 to call a variant. Alternatively, a more complex command
can perform additional filtering:


$ bcftools view --include 'FMT/GT="1/1" && QUAL>=30 && INFO/DP>=10
&& (FMT/AO)/(FMT/DP)>=0.75' PT000033.raw.vcf > PT000033.filt.vcf
```

The PT000033.filt.vcf file contains the called SNPs and small INDELs. Large structural variants that are longer than the read length must be identified using other approaches and software.

But let's look at the VCF (Variant Call Format) format, type:

```
$ more PT000033.filt.vcf
```

The VCF file is composed of a number of header lines starting with the hash symbol ('#') containing metadata and other information, followed by 10 columns of data on called variants:

```
##fileformat=VCFv4.2
##FILTER=<ID=PASS,Description="All filters passed">
##samtoolsVersion=1.3+htslib-1.3
##samtoolsCommand=samtools mpileup -Q 23 -d 2000 -C 50 -ugf NC000962_3.fasta PT000033.sorted.bam
##reference=file://NC000962_3.fasta
##contig=<ID=NC000962_3,length=4411532>
##ALT=<ID=*,Description="Represents allele(s) other than observed.">
##INFO=<ID=INDEL,Number=0,Type=Flag,Description="Indicates that the variant is an INDEL.">
##INFO=<ID=IDV,Number=1,Type=Integer,Description="Maximum number of reads supporting an indel">
##INFO=<ID=IMF,Number=1,Type=Float,Description="Maximum fraction of reads supporting an indel">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Raw read depth">
##INFO=<ID=VDB,Number=1,Type=Float,Description="Variant Distance Bias for filtering splice-site artefacts in RNA-seq data (bigger is better)",Version="3">
##INFO=<ID=RPB,Number=1,Type=Float,Description="Mann-Whitney U test of Read Position Bias (bigger is better)">
##INFO=<ID=MQB,Number=1,Type=Float,Description="Mann-Whitney U test of Mapping Quality Bias (bigger is better)">
##INFO=<ID=BQB,Number=1,Type=Float,Description="Mann-Whitney U test of Base Quality Bias (bigger is better)">
##INFO=<ID=MQSB,Number=1,Type=Float,Description="Mann-Whitney U test of Mapping Quality vs Strand Bias (bigger is better)">
##INFO=<ID=SGB,Number=1,Type=Float,Description="Segregation based metric.">
##INFO=<ID=MQ0F,Number=1,Type=Float,Description="Fraction of MQ0 reads (smaller is better)">
##FORMAT=<ID=PL,Number=G,Type=Integer,Description="List of Phred-scaled genotype likelihoods">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##INFO=<ID=ICB,Number=1,Type=Float,Description="Inbreeding Coefficient Binomial test (bigger is better)">
##INFO=<ID=HOB,Number=1,Type=Float,Description="Bias in the number of HOMs number (smaller is better)">
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes for each ALT allele, in the same order as listed">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called genotypes">
##INFO=<ID=DP4,Number=4,Type=Integer,Description="Number of high-quality ref-forward , ref-reverse, alt-forward and alt-reverse bases">
##INFO=<ID=MQ,Number=1,Type=Integer,Description="Average mapping quality">
##bcftools_callVersion=1.5-26-g81c6dd3+htslib-1.5-14-ge1380c8
##bcftools_callCommand=call -O v -vm -o PT000033.raw.vcf; Date=Tue Aug 22 13:51:27 2017
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  PT000033
NC000962_3      1849    .       C       A       142     .       DP=12;VDB=0.00174423;SGB=-0.670168;MQ0F=0;AC=2;AN=2;DP4=0,0,0,10;MQ=45  GT:PL   1/1:172,30,0
NC000962_3      1977    .       A       G       154     .       DP=10;VDB=0.198441;SGB=-0.651104;MQSB=1;MQ0F=0;AC=2;AN=2;DP4=0,0,7,1;MQ=44     GT:PL   1/1:184,24,0
NC000962_3      3446    .       C       T       225     .       DP=54;VDB=0.0158223;SGB=-0.693147;MQSB=0.97197;MQ0F=0;AC=2;AN=2;DP4=0,0,28,19;MQ=45    GT:PL   1/1:255,141,0
NC000962_3      4013    .       T       C       225     .       DP=49;VDB=0.0969683;SGB=-0.693146;MQSB=0.0153297;MQ0F=0;AC=2;AN=2;DP4=0,0,25,19;MQ=45  GT:PL   1/1:255,132,0
NC000962_3      7362    .       G       C       225     .       DP=62;VDB=0.827911;SGB=-0.693147;MQSB=0.971079;MQ0F=0;AC=2;AN=2;DP4=0,0,30,23;MQ=44    GT:PL   1/1:255,160,0
NC000962_3      7585    .       G       C       225     .       DP=46;VDB=0.761957;SGB=-0.693146;MQSB=0.202323;MQ0F=0;AC=2;AN=2;DP4=0,0,14,28;MQ=44    GT:PL   1/1:255,126,0
NC000962_3      8405    .       C       A       225     .       DP=31;VDB=0.121957;SGB=-0.693079;MQSB=0.960561;MQ0F=0;AC=2;AN=2;DP4=0,0,14,15;MQ=45    GT:PL   1/1:255,87,0
NC000962_3      9304    .       G       A       225     .       DP=33;VDB=0.502902;SGB=-0.693054;MQSB=0.267776;MQ0F=0;AC=2;AN=2;DP4=0,0,19,9;MQ=45     GT:PL   1/1:255,84,0
NC000962_3      11879   .       A       G       225     .       DP=31;VDB=0.00639057;SGB=-0.693079;MQSB=0.840657;MQ0F=0;AC=2;AN=2;DP4=0,0,15,14;MQ=44  GT:PL   1/1:255,87,0
NC000962_3      12204   .       G       A       225     .       DP=22;VDB=0.246837;SGB=-0.692067;MQSB=0.984877;MQ0F=0;AC=2;AN=2;DP4=0,0,5,15;MQ=44     GT:PL   1/1:255,60,0
NC000962_3      14785   .       T       C       225     .       DP=37;VDB=0.400878;SGB=-0.693127;MQSB=0.976914;MQ0F=0;AC=2;AN=2;DP4=0,0,16,17;MQ=44    GT:PL   1/1:255,99,0
NC000962_3      15117   .       C       G       225     .       DP=36;VDB=0.22674;SGB=-0.693132;MQSB=0.997491;MQ0F=0;AC=2;AN=2;DP4=0,0,13,21;MQ=44     GT:PL   1/1:255,102,0
NC000962_3      21795   .       G       A       225     .       DP=28;VDB=0.40874;SGB=-0.692717;MQSB=0.95306;MQ0F=0;AC=2;AN=2;DP4=0,0,10,13;MQ=44     GT:PL   1/1:255,69,0
NC000962_3      24007   .       G       T       225     .       DP=36;VDB=0.921407;SGB=-0.693136;MQSB=0.99869;MQ0F=0;AC=2;AN=2;DP4=0,0,22,13;MQ=44     GT:PL   1/1:255,105,0
```

Column	Field	Description
1	CHROM	Chromosome name
2	POS	Left-most position of the variant
3	ID	Unique variant identifier
4	REF	Reference allele
5	ALT	Alternate allele
6	QUAL	Variant/Reference Quality
7	FILTER	Filters applied
8	INFO	Information related to the variant, separated by semi-colon
9	FORMAT	Format of the genotype fields, separated by colon (optional)
10	SAMPLE	Sample Genotypes and per-sample information (optional)

You can open the vcf file with any text editor or even import it into an Excel spreadsheet. You can right-click on the file and select Open with LibreOffice Calc and import it as a file containing fields separated by tabs.

We can also view the vcf file in Artemis, but first it is necessary to compress the vcf file and index with tabix (from BCFtools):

```
$ bgzip -c PT000033.filt.vcf > PT000033.filt.vcf.gz
$ tabix -p vcf PT000033.filt.vcf.gz
```

Some software only read compressed vcf files indexed with tabix. Indexation with tabix produces an index file (*.tbi) that should be in the same directory as the compressed vcf file (*.vcf.gz).

To open the vcf file in Artemis repeat the steps from Exercise 5 and load the NC000962.gbk files and read-in the PT000033.sorted.bam file. Now, go again to File>Read BAM/VCF and select the compressed vcf file (PT000033.filt.vcf.gz). Another view should appear with vertical lines highlighting the variants on the VCF file. It is generally recommended not to consider variants with QUAL score below 30. Recall the Phred33 table in Exercise 1. What is the error probability of QUAL 30 score?

You can apply different sorts of filters by right-clicking on the VCF view and selecting Filters.

Inside Module1 directory, there is a sub-directory called vcfs. This directory contains additional VCF files for you to use. You can add these vcf files for comparison purposes. Right-click on the VCF view and select Add VCF …

[Alternatively, you can generate these VCF files yourself by repeating the steps from the different exercises in this module using the FASTQ files in the course_files directory, see Exercise 1].

Exercise 7 – Variant Functional Annotation

The last, and optional, exercise of this module pertains the functional annotation of the VCF file. This last step is extremely useful as it will annotate each variant with the respective genetic impact. Not only we will be able to directly evaluate if each variant is synonymous or non-synonymous but it will be also possible to directly see the genetic mutation and affected gene.

We will use another Java written program – SnpEff - that requires some configuration that is ouside the scope of this module. However, the online documentation available at http://snpeff.sourceforge.net/SnpEff_manual.html is very comprehensive and contains detailed steps on how to download or build its database. SnpEff takes the uncompressed

VCF file as input and generates another VCF file annotated with additional information in the INFO field (ANN) [13]. We can do this by typing:

```
$ snpEff -no-downstream -no-upstream -v -c ~/snpEffdata/snpeff.config
NC000962_3 PT000033.filt.vcf > PT000033.ann.vcf

## With this command, we are specifying the path to SnpEff configuration
file using the -c option; the -v option turns on the verbose mode; and
the -no-downstream and -no-upstream options disable the annotation of
downstream and upstream variants. We selected this option as otherwise
the VCF file would be overpopulated with upstream and downstream modifier
variants as genes in bacteria are very close to each other.
```


This VCF file can also be loaded in Artemis although it has to be compressed with bgzip and indexed with tabix.

Let us look at this new VCF file by typing:

```
$ more PT000033.ann.vcf
```

Can you spot the difference?

What about the INH-associated mutation that we detected visually? Can you find it using grep (sort of a Linux/GNU built-in Find):

```
$ grep "katG" PT000033.ann.vcf
```

Tip: Try to open this newly annotated VCF file on Excel/LibreOffice Calc. Down the line, parsing this format in R will allow you to carry out more advanced statistical analysis.

Optional:

You can also use BCFTOOLS to extract data to a more tabular format, you just need to specify the fields and become better acquainted with the VCF format and the syntax for the bcftools query command. Check this out:

```
$ bcftools query -f '%CHROM\t%POS\t%REF\t%ALT\t%QUAL\t%DP\t%ANN\n'
PT000033.ann.vcf | sed 's/|/\t/g'

# Notice that you just used sed (similar to find and replace) to replace
the pipe symbols (|) from the ANN field with tabulations (\t).
```

References

1 Loman NJ, Constantinidou C, Chan JZ, et al. High-throughput bacterial genome sequencing: an embarrassment of choice, a world of opportunity. *Nat Rev Microbiol*. 2012; **10**: 599-606.

2 Bolger AM, Lohse M, Usadel B. Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics*. 2014; **30**: 2114-2120.

3 Olson ND, Lund SP, Colman RE, et al. Best practices for evaluating single nucleotide variant calling methods for microbial genomics. *Front Genet*. 2015; **6**: 235.

4 Li H, Homer N. A survey of sequence alignment algorithms for next-generation sequencing. *Brief Bioinform*. 2010; **11**: 473-483.

5 Mielczarek M, Szyda J. Review of alignment and SNP calling algorithms for next-generation sequencing data. *J Appl Genet*. 2016; **57**: 71-79.

6 Li H, Handsaker B, Wysoker A, et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*. 2009; **25**: 2078-2079.

7 Li H, Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*. 2009; **25**: 1754-1760.

8 Milne I, Bayer M, Stephen G, Cardle L, Marshall D. Tablet: Visualizing Next-Generation Sequence Assemblies and Mappings. *Methods Mol Biol*. 2016; **1374**: 253-268.

9 Carver T, Harris SR, Berriman M, Parkhill J, McQuillan JA. Artemis: an integrated platform for visualization and analysis of high-throughput sequence-based experimental data. *Bioinformatics*. 2012; **28**: 464-469.

10 Freese NH, Norris DC, Loraine AE. Integrated genome browser: visual analytics platform for genomics. *Bioinformatics*. 2016; **32**: 2089-2095.

11 McKenna A, Hanna M, Banks E, et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research*. 2010; **20**: 1297-1303.

12 Danecek P, Auton A, Abecasis G, et al. The variant call format and VCFtools. *Bioinformatics*. 2011; **27**: 2156-2158.

13 Cingolani P, Platts A, Wang le L, et al. A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3. *Fly (Austin)*. 2012; **6**: 80-92.

:: *De novo* Assembly ::

Introduction

This second module explores an alternative approach to reconstruct genomic data from NGS reads – ***de novo* Assembly**. Rather than using a mapping strategy to guide the assembly of sequence reads, de novo Assembly attempts to reconstruct genomes by exploring read overlapping and contiguity. This approach is more informative when we are dealing with a new species for which there is no reference genome or even a new strain of a well-known pathogen [1, 2].

However, the process of assembling reads into contigs can be challenging due to the huge number of reads produced from random sampling of the genome being studied. Additionally, high sequencing error rate, repeat regions/patterns and uneven sequencing depth are some of the problems that can influence the correct assembly of a genome. There are three major approaches for assembling reads: i) **overlap-and-extend**; ii) **string graph**; and iii) **de Bruijn graphs**. Overlap-and-extend methods iteratively attempt to first find read overlaps (where the suffix of a read is equal to the prefix of another read with alength that meets a defined threshold) and then extend the first read constructing a longer read (SSAKE, VCAKE and SHARCGS). The String Graph based assemblers constructs a string graph for every read in which each read is a vertex and there is an edge from a vertex to another if there is read overlap (Edena and BOA). These first two approaches are more susceptible to sequencing errors and can lead to more memory consumption [2, 3].

De Bruijn Graph-based algorithms are presently the most widely used approaches and are used in several software packages for genome assembly from NGS reads (e.g. Velvet, SOAPdenovo, SPAdes, etc) [4, 5]. Graphs are mathematical structures used to model pairwise relations between objects. In essence, each vertex represents a length-*k* substring (*k*-mer) in a read and there is a directed edge from vertex *u* to vertex *v* if *u* and *v* are consecutive *k*-mers in a read, i.e., the last *k*–1 nucleotides of the *k*-mer represented by *u* is the same as the first *k*–1 nucleotides of the *k*-mer represented by *v* [3].

Exercise 1 – De Bruijn Graphs

Before going to the computational part we can do a simple and quick hands-on exercise that covers the basics of de Bruijn Graph based assemblies. A concept that will be important to retain is the **concept of *k*-mer**. As explained above the k-mer is a substring

of a read and, a read can thus have multiple k-mers depending on the k-mer length (sometimes this is also referred as hash length or word length). For example, if you have a 50bp read it can only yield one 50bp k-mer, but it can yield two 49bp k-mers [4]. The caveat of this approach is that the k-mer length you choose must always be below the read-length of your data.

Let's consider the following reads:

ATGCTA
GCTAGC
TAGCAC
CTAGCA

We can use a simple overlap and extend approach but let's use the de Bruijn Graph approach to find *k*-mers. List all 3 bp k-mers from the reads above:

First, link the k-mers that only differ by k-1 nucleotides. Then, you can try to establish links by creating a path between the nodes to find your contig – each k-mer is a node. The two main rules are: **visit all nodes at least once** and **use the minimal path length.**

Write the final contig here:

Exercise 2 – *De novo* Assembly

Now that you have attempted to do de novo assembly by hand it is time to use real data. We'll continue with the PT000033 strain from Module 1 but, instead of mapping, we will do de novo assembly. Let's go to Module2 directory by typing (from your home directory) and run Trimmomatic for the files:

```
# Start by copying the files from the course_files directory:
$ cp ../course_files/PT000033_*.fastq.gz .


# Let us decompress the files:
$ gzip -d PT000033_1.fastq.gz
$ gzip -d PT000033_2.fastq.gz


# Now let's run Trimmomatic:
$ trimmomatic PE -phred33 PT000033_1.fastq PT000033_2.fastq
PT000033_1_trimmed_paired.fastq PT000033_1_trimmed_unpaired.fastq
PT000033_2_trimmed_paired.fastq PT000033_2_trimmed_unpaired.fastq
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:20 MINLEN:36
```

Inside this sub-directory (you can see its content using the ls command) you will find the fastq files used in Module 1, including the files produced by Trimmomatic. In this exercise we will use Velvet to assemble the PT000033 strain from sequence reads without using a reference genome [6].

You can also find in this sub-directory the reference genome used in Module 1. Although we will not use it in the assembly process, this genome file will be necessary in Exercise 3 for ordering the contigs.

This will be a simple approach to computational assembly using **Velvet**, which relies on two main programs: velveth and velvetg. velveth takes in a number of sequence files, produces a hash table, then outputs two files in an output directory (creating it if necessary), Sequences and Roadmaps, which are necessary to velvetg. The file formats supported by velveth include FASTA, FASTQ, compressed (gunzip) FASTA/FASTQ files, SAM/BAM, etc. Moreover, there are different read categories that can be specified depending on the data: -short (default), -shortPaired (for paired-end short read data), -short2 (to specify a second library), -shortPaired2 (idem), -long (for Sanger, 454 or even reference sequences) and -longPaired.

The second program, velvetg, is the core of Velvet where the de Bruijn graph is built and then manipulated.

Perhaps the most important parameter you must specify is the hash length or *k*-mer length. In fact, it cannot be inferred from the data such as other parameters that we are setting to auto or using default values. The two main rules for choosing the *k*-mer length are: it must be an odd number, to avoid palindromes (if you specify an even number Velvet will automatically decrease it); and, it must be inferior to the read length. Longer *k*-mers generally provide higher specificity but decrease coverage (and therefore sensitivity).

But, let's start assembling. Inside Module2 directory type:

```
$ velveth PT000033_41 41 -fastq -shortPaired
PT000033_1_trimmed_paired.fastq PT000033_2_trimmed_paired.fastq -
fastq -short PT000033_1_trimmed_unpaired.fastq
PT000033_2_trimmed_unpaired.fastq

# This first command creates the hash table in a new directory
PT000033_41 (it will contain the assembly using a k-mer length of
41)

$ velvetg PT000033_41 -exp_cov auto -cov_cutoff auto

# This second command produces the graph and assembles the genome.
You can find your files in the newly created PT000033_21 sub-
directory.
```

Notice that in the first command you specified two distinct libraries: a paired library present across two files using the -shortPaired option and two unpaired libraries using the -short option. We could have just used the paired library but in this way we do not loose good quality data.

Upon velvetg completion look at the last line summarising the results. Also, you can run the following command to find out the number of contigs and some additional statistics:

```
$ assemblathon_stats.pl ./PT000033_41/contigs.fa
```

This command will output additional statistics. The assemblathon_stats.pl (Keith Bradnam, UC Davis) is a Perl written script that calculates some statistics for assemblies and scaffolds (we will talk about this ahead). It was written for the Assemblathon contests to assess

state-of-the-art methods in the field of genome assembly [1]. You can look at some of these metrics and take note in the table below. The output will be similar to the one below and as we are only working with assembled contigs we can focus on the third section only:

```
                                 Number of contigs       268
                     Number of contigs in scaffolds         0
                 Number of contigs not in scaffolds       268
                            Total size of contigs   4343956
                                   Longest contig   2764650
                                  Shortest contig        61
                        Number of contigs > 1K nt        53  19.8%
                       Number of contigs > 10K nt         3   1.1%
                      Number of contigs > 100K nt         2   0.7%
                        Number of contigs > 1M nt         2   0.7%
                       Number of contigs > 10M nt         0   0.0%
                                 Mean contig size     16209
                               Median contig size       127
                               N50 contig length   2764650
                                L50 contig count         1
                                       contig %A     17.25
                                       contig %C     32.44
                                       contig %G     32.91
                                       contig %T     17.33
                                       contig %N      0.06
                                contig %non-ACGTN      0.00
                     Number of contig non-ACGTN nt         0
```

What do these parameters mean?

- Contigs: number of contigs? Which is better, more or less contigs?
- N50: Length of the contig that contains the middle nucleotide when the contigs are ordered by size.
- Longest contig: length of the longest contig;
- Total size: Sum of all contigs lengths.

How many contigs are larger than 10 Kb? _____

Statistic	***K*-mer**		
	41	**49**	**55**
Contigs			
Total Size			
Longest Contig			
Mean Contig Size			
N50			

Repeat this exercise using a *k*-mer length of 49 and 55 (**don't forget to change the directory name: PT000033_41 to PT000033_49 or PT000033_55**).

Which is the best *k*-mer length? _____

Upon completion of these commands you should have obtained your contigs. Each of the three assemblies are located in the respective folders. Let's look at the assembly:

```
$ cd PT000033_49

# This will get you inside the PT000033_49 subdirectory, assuming
you are inside the Module2 directory. Recall that this is the
assembly carried out using a k-mer length of 31bp.

# Then:

$ more contigs.fa
```

In which format are the contigs' sequences? _____

Besides the contigs file you will find other files containing the sequences, graphs and other files necessary if you want to perform a faster re-assembly of your data using different parameters.

But now that you have the contigs we can order these and/or join them in **Exercise 3**.

Optional:

Graph Visualization:

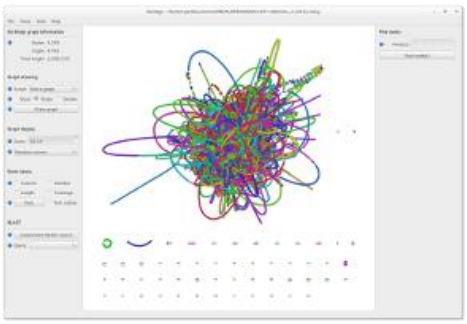
Under the assembly directories you can find the file LastGraph. It contains the final Graph of your assembly. It is possible to visualize and navigate this graph using the software Bandage. This is a program that can be installed on Windows, MacOS or Linux machines and it creates interactive visualizations of assembly graphs [7].

To open Bandage double-click on its shortcut at the desktop or, from any terminal window just type:

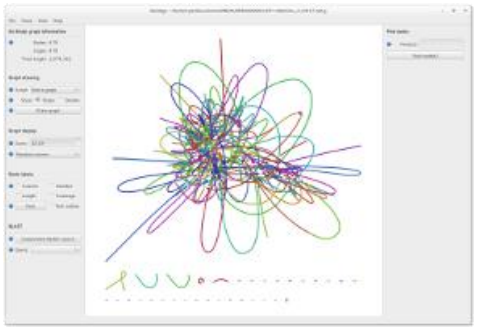
```
$ Bandage
```

This will start the Bandage window, go to *File* > *Load Graph* and select the LastGraph file from an assembly at your choice to open the Graph of your assembly. To visualize this Graph click on *Draw Graph*. Since this is a bacterial genome you can draw the entire graph. You can compare the different assemblies using this approach.

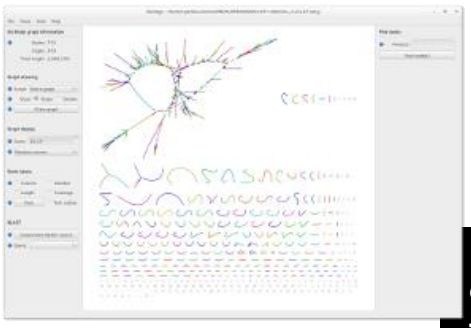
The graph for your assembly will show the different nodes and be similar to this:



K-mer of 21. This value is too small, resulting in short contigs and many connections, giving a dense tangled graph.



K-mer of 77. This is a good balance, giving long contigs that are well connected.



K-mer of 127. This value is too large resulting in the graph breaking into many discontinuous pieces.

Other software for assembly statistics/QC:

There are more complete alternatives to evaluate the assembly quality, namely, using BUSCo or QUAST. QUAST for example compares against a reference genome, provided that one exists. Busco offers another alternative by checking the degree of representation of a set of core genes in each assembly.

You can run QUAST in this VM by:

```
$ cd PT000033_41


$ quast.py -o quast_qc -r ../NC000962_3.fasta contigs.fa


# Inside the newly created quast_qc directory look at the report
files:


$ cat quast_qc/report.tsv
```

For BUSCO analysis in the same directory:

```
$ conda activate busco


$ busco -i contigs.fa -l actinobacteria_phylum_odb10 -o
busco_analysis -m genome


$ conda deactivate
```


Additional notes:

The assembly exercise carried out in this exercise is a simplistic approach. In a real situation it is necessary to carefully estimate several parameters such as the expected coverage cut-off, expected coverage, k-mer length, insert size, etc. VelvetOptimiser script (https://github.com/tseemann/VelvetOptimiser) deals with these problems and carries out several Velvet assemblies while simultaneously adjusting for several of these parameters. You can input a range of K-mer lengths .

A different approach is implemented in SPAdes by using multi k-mer assemblies where you specify several *k*-mer lengths [5]. SPAdes takes as input paired-end reads, mate-pairs and single (unpaired) reads in FASTA and FASTQ. Besides SPAdes other assemblers implement a multi K-mer strategy (IDBA, Megahit, GATB-Pipeline) which always performs better than single K-mer strategies. The downside, and the main reason to why we do not used this strategy in this module, are the longer run times and the fact that these are more resource-demanding softwares.

Exercise 4 - Genome Annotation

In the last exercise of this module we will attempt to find features along the scaffold created earlier and add some information concerning it to genome. This is a process called annotation and, it is necessary if we want to take full advantage of having the genome sequence and move towards functional genomics [2]. This annotation process is made with biologically relevant information that can range from gene models, functional data (including gene ontology or "Kyoto Encyclopedia of Genes and Genomes", KEGG, pathways) to epigenetic or microRNA modifications. Generally, this procedure is limited to the annotation of protein coding sequences and may include the annotation of ribosomal and transfer RNAs.

We can take three different approaches to annotation: web-based tools (RAST, NCBI annotation); command-line tools that perform *de novo* gene discovery (Prokka and DIYA); and, transfer of annotation data from a reference genome such as the one we used in the previous exercise (RATT, BG-7) [8, 13, 14].

Moreover, annotation will imply a different format other than FASTA. While the latter only stores the sequence, it is possible to have a second file in the GFF (General Feature File) format containing the annotation. Alternatively, we can combine these data on a GenBank file. You can look up online the structure of both formats.

Prokka Annotation:

In this exercise we will take the first two approaches and we will start with annotation with Prokka, a software tool to annotate bacterial, archaeal and viral genomes quickly and produce standards-compliant output files [13]. Prokka uses a variety of databases when trying to assign function to the predicted CDS features. It takes a hierarchial approach to make it fast. The initial core databases are derived from UniProtKB.

To start the annotation let's type:

```
$ cd /home/centos/Module2/PT000033_49


$ prokka --outdir ./PT000033_prokka --prefix PT000033 contigs.fa


# Alternatively, for a monomorphic species such as Mycobacterium
tuberculosis you can opt by transferring the annotation from M.
tuberculosis H37Rv reference strain using the --proteins option:


$ cp ~/course_files/NC000962_3.gbk .


$ prokka --outdir ./PT000033_prokka --prefix PT000033 --proteins
NC000962_3.gbk contigs.fa
```


Here, you have carried out a simple annotation procedure using Prokka's core database. With the --outdir option you specified the creation of a new output directory and the --prefix option allow you to set the prefix name of your files. You can look at other Prokka options by simply typing:

```
$ prokka
```

Meanwhile the annotation process will take about 16 minutes, but when finished you can find the new PT000033_prokka directory. Let's go inside:

```
$ cd PT000033_prokka
```

Prokka will have outputted the following files:

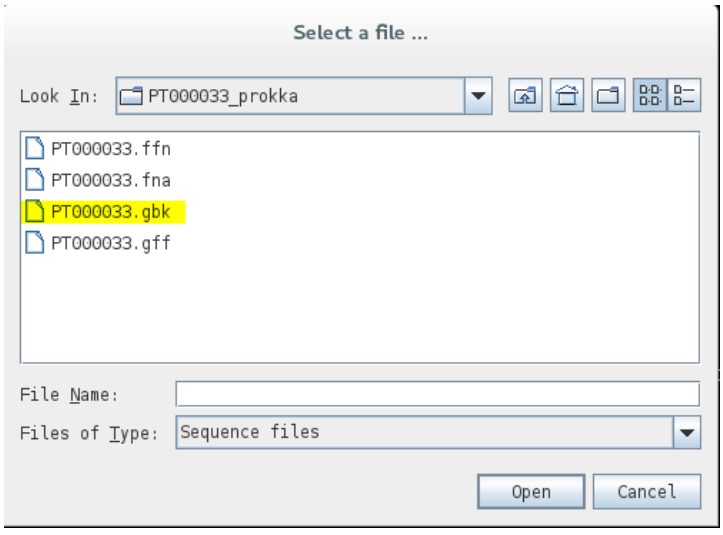
Extension	Description
.gff	This is the master annotation in GFF3 format, containing both sequences and annotations. It can be viewed directly in Artemis or IGV.
.gbk	This is a standard Genbank file derived from the master .gff. If the input to prokka was a multi-FASTA, then this will be a multi-Genbank, with one record for each sequence.
.fna	Nucleotide FASTA file of the input contig sequences.
.faa	Protein FASTA file of the translated CDS sequences.
.ffn	Nucleotide FASTA file of all the prediction transcripts (CDS, rRNA, tRNA, tmRNA, misc_RNA)

.sqn	An ASN1 format "Sequin" file for submission to Genbank. It needs to be edited to set the correct taxonomy, authors, related publication etc.
.fsa	Nucleotide FASTA file of the input contig sequences, used by "tbl2asn" to create the .sqn file. It is mostly the same as the .fna file, but with extra Sequin tags in the sequence description lines.
.tbl	Feature Table file, used by "tbl2asn" to create the .sqn file.
.err	Unacceptable annotations - the NCBI discrepancy report.
.log	Contains all the output that Prokka produced during its run. This is a record of what settings you used, even if the --quiet option was enabled.
.txt	Statistics relating to the annotated features found.
.tsv	Tab-separated file of all features: locus_tag,ftype,gene,EC_number,product

Next, open up Artemis:

```
$ art
```

And go to File > Open and open the PT000033.gbk file:



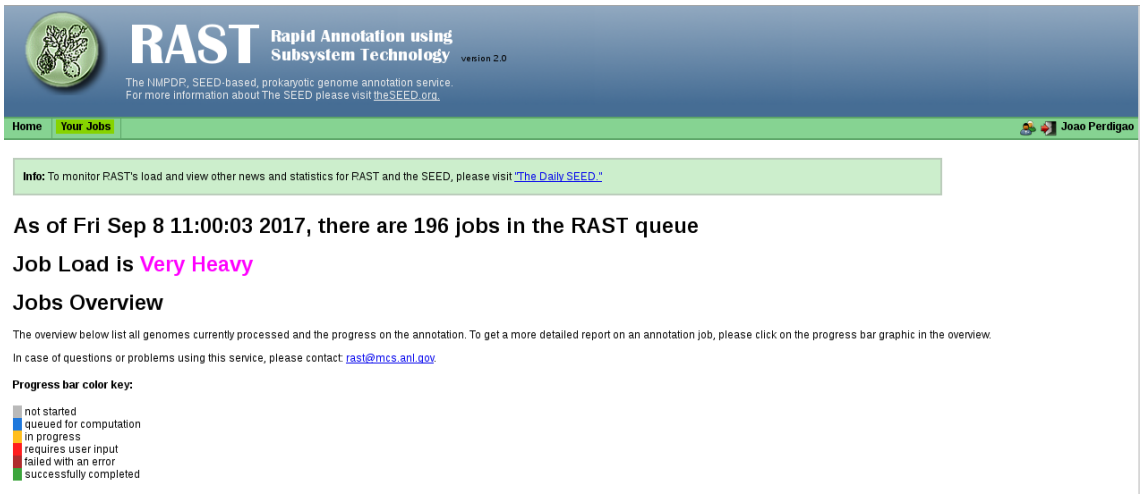
You can see that your scaffold from the previous exercise is now annotated:

As a side note, to improve Prokka annotations it is possible to use user-define databases from closely related genomes.

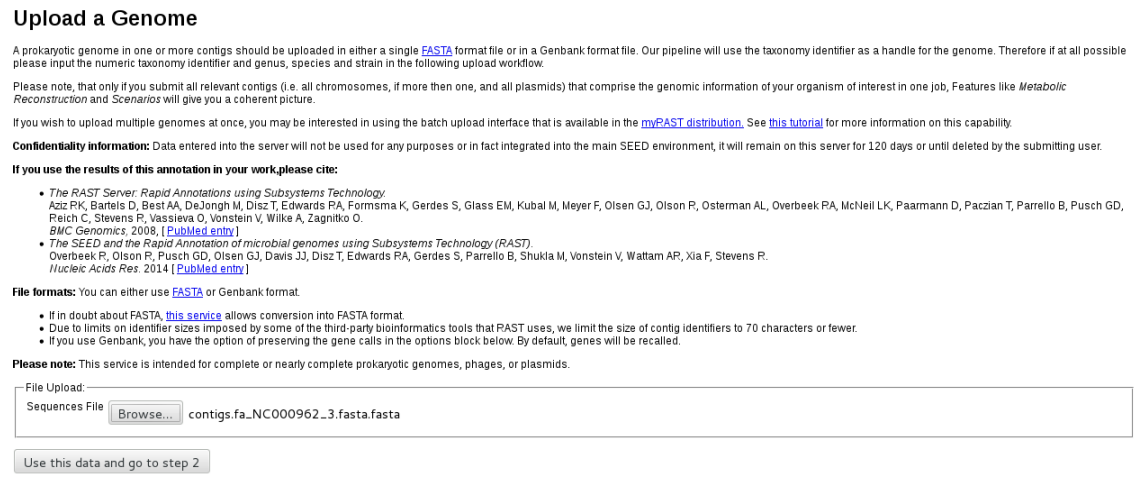
RAST Annotation:

RAST (Rapid Annotation using Subsystem Technology) is a fully-automated service for annotating complete or nearly complete bacterial and archaeal genomes available at http://rast.nmpdr.org/. RAST is designed to rapidly call and annotate the genes of a complete or essentially complete prokaryotic genome [14]. RAST, Rapid Annotations based on Subsystem Technology, uses a "Highest Confidence First" assignment propagation strategy based on a more sophisticated database based on manually curated subsystems and subsystem-based protein families that automatically guarantees a high degree of assignment consistency. RAST returns an analysis of the genes and subsystems in your genome, as supported by comparative and other forms of evidence. Despite being an on-line tool, it is possible to use RAST on the command-line through the myRAST toolkit, therefore allowing the integration of RAST in scripts and pipelines [14].

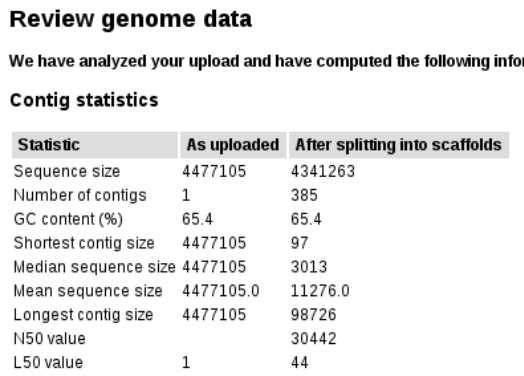
To use RAST you need to set up a free account.



Once you have registered to RAST and your account is active, log in to your RAST homepage and go to *Your Jobs* > *Upload New Job*.



On this next screen select the scaffold file(contigs.fa_NC000962_3.fasta.fasta). On the next screen you can review some sequence statistics before proceeding:



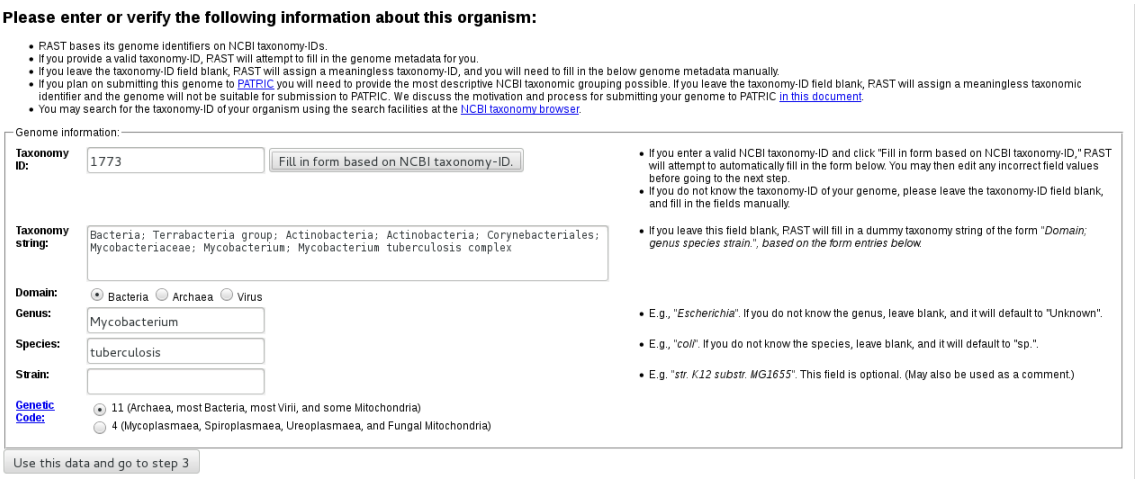
Review genome data

We have analyzed your upload and have computed the following information.

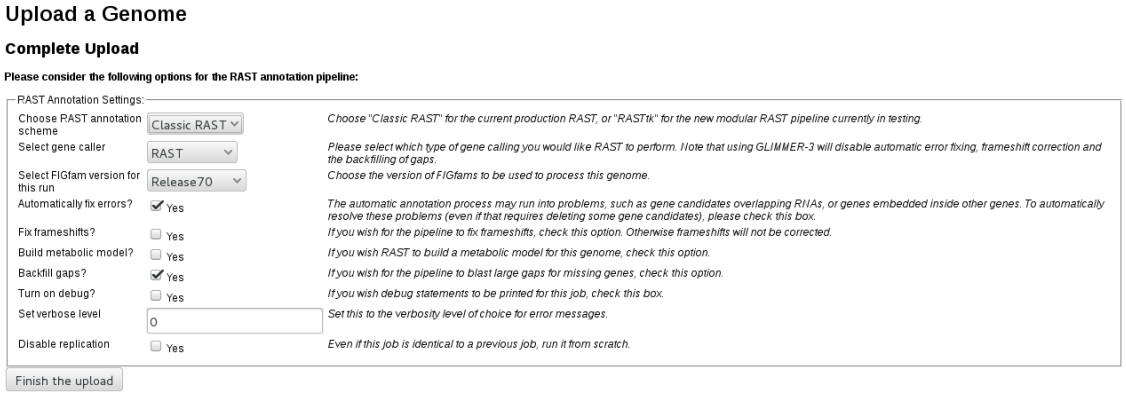
Contig statistics

Statistic	As uploaded	After splitting into scaffolds
Sequence size	4477105	4341263
Number of contigs	1	385
GC content (%)	65.4	65.4
Shortest contig size	4477105	97
Median sequence size	4477105	3013
Mean sequence size	4477105.0	11276.0
Longest contig size	4477105	98726
N50 value		30442
L50 value	1	44

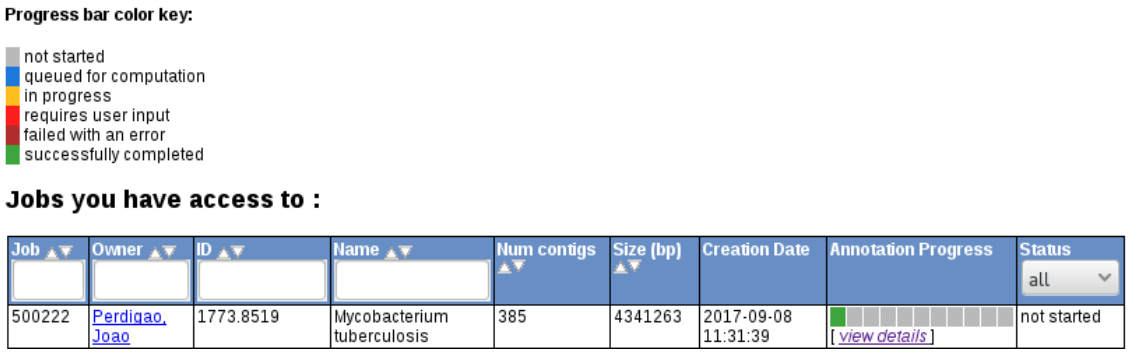
Below, in this same page you will be asked to introduce some information concerning the organism. Since we are working with Mycobacterium tuberculosis you can enter NCBI Taxonomy Id 1773 and click on the "Fill in form based on NCBI Taxonomy-ID" button as it will fill the form automatically:



In the last step you can chose some RAST annotation parameters. We will go with the default parameters:



Genome annotation using RAST may take half a day, an entire day or longer depending on server availability. Upon completion you can go to the Jobs Overview page where you can find your submitted jobs:



If your job is already completed you can click on *view details*, which will give you access to RAST output files. These files include, GenBank, GFF, FASTA, EMBL and even spradsheets (Excel and tsv formats) containing the list of features, along with genome coordinates, orientation, gene ID, gene product, nucleotide and protein sequences, etc.

A RAST annotated PT000033 genome is available for you under the RAST_output sub-directory in Module2 directory. Open a terminal and type:

```
$ cd ./Module2/RAST_output
# Then start artemis:
$ art
```

Then open the GenBank file as you did with the Prokka annotation file. Is this annotation better?

Also, try to open the Excel spreadsheet with LibreOffice Calc in the Virtual Machine. LibreOffice Calc is an Open-Source alternative to Microsoft Excel. You can easily extract gene information from your strain using this file as well (and maybe even reading it into R!).

References

1 Baker U, Tomson G, Some M, et al. 'How to know what you need to do': a cross-country comparison of maternal health guidelines in Burkina Faso, Ghana and Tanzania. *Implement Sci*. 2012; **7**: 31.

2 Ekblom R, Wolf JB. A field guide to whole-genome sequencing, assembly and annotation. *Evol Appl*. 2014; **7**: 1026-1042.

3 Chin FY, Leung HC, Yiu SM. Sequence assembly using next generation sequencing data--challenges and solutions. *Sci China Life Sci*. 2014; **57**: 1140-1148.

4 Compeau PE, Pevzner PA, Tesler G. How to apply de Bruijn graphs to genome assembly. *Nat Biotechnol*. 2011; **29**: 987-991.

5 Bankevich A, Nurk S, Antipov D, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J Comput Biol*. 2012; **19**: 455-477.

6 Zerbino DR, Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*. 2008; **18**: 821-829.

7 Wick RR, Schultz MB, Zobel J, Holt KE. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*. 2015; **31**: 3350-3352.

8 Swain MT, Tsai IJ, Assefa SA, Newbold C, Berriman M, Otto TD. A post-assembly genome-improvement toolkit (PAGIT) to obtain annotated genomes from contigs. *Nat Protoc*. 2012; **7**: 1260-1284.

9 Carver TJ, Rutherford KM, Berriman M, Rajandream MA, Barrell BG, Parkhill J. ACT: the Artemis Comparison Tool. *Bioinformatics*. 2005; **21**: 3422-3423.

10 Darling AE, Mau B, Perna NT. progressiveMauve: multiple genome alignment with gene gain, loss and rearrangement. *PLoS ONE*. 2010; **5**: e11147.

11 Galardini M, Biondi EG, Bazzicalupo M, Mengoni A. CONTIGuator: a bacterial genomes finishing tool for structural insights on draft genomes. *Source Code Biol Med*. 2011; **6**: 11.

12 Alonge M, Lebeigle L, Kirsche M, et al. Automated assembly scaffolding using RagTag elevates a new tomato system for high-throughput genome editing. *Genome Biol*. 2022; **23**: 258.

13 Seemann T. Prokka: rapid prokaryotic genome annotation. *Bioinformatics*. 2014; **30**: 2068-2069.

14 Overbeek R, Olson R, Pusch GD, et al. The SEED and the Rapid Annotation of microbial genomes using Subsystems Technology (RAST). *Nucleic Acids Res*. 2014; **42**: D206-214.

:: Pathogen Toolbox and Additional Software Tools for Genomics::

Introduction

As NGS sequencing is becoming simpler and cheaper, increasing volumes of WGS data for bacterial pathogens such as *M. tuberculosis* are being generated . However, a significant part of the existing bottleneck lies within the analytical stage and the challenge for clinical and public health laboratories in accessing, extracting and interpreting the relevant and meaningful information. To bridge this gap, over the last years several tools have been developed and described aiming at an audience with limited bioinformatic skills [1, 2]. These resources are, foremost, dedicated to the prediction of drug resistance from sequence data although other tools aiming at a genotypic characterization have been described as well. While some of these tools have also been deployed on an online server thus enabling easy access using a Guided User Interface (GUI) and remote analytical processing. Some mere examples of such tools for TB:

- **TBprofiler** – enables drug resistance prediction, (sub)lineage assignament from raw read data. Features an online version and a standalone, locally deployable, command-line version [3];
- **PhyResSE** – uses the same philosophy as TBprofiler and also reports strain lineage and antibiotic resistance [4];
- **Mykrobe Predictor** – also reporting drug resistance and lineage from reads, this software runs locally [5];
- **SpoTyping and Spolpred** – these two tools report the spoligotyping profile from read data. Spotyping can query SITVIT database for SIT/Clade assignement for the obtained profile [6, 7];

The downside of these tools is the limitation, to different degrees, of customizing the pipeline and the analytical workflow in case there is the need to perform additional steps or different parameters. Also, the caveat to the use of these resources also lies in the need for the critical interpretation of the results that are obtained.

In addition, other tools of more general use are also available for other pathogens, envisaging the detection of plasmid replicons, drug resistance genes and even for the analysis of the gene content across the core/pan genomes. This enables rapid extraction of important genotypic data that can be of clinical or epidemiological importance.

This module will cover the use of selected available tools as examples for the plethora of tools already made available and maintained by the scientific community. Some of these tools are presently implemented or being across diagnosis or public health microbiology laboratories worldwide.

Part I – *Mycobacterium tuberculosis* related software

Exercise 1 – Extracting spoligotypes

This first exercise will be focused on obtaining spoligotyping information from raw sequence data. Two available tools Spolpred and Spotyping can be used for this end, both rely on the identification of specific regions called spacer regions in the Direct Repeat locus. Basically it an in silico version of what is carried out in the lab using the classical methodology: 43 probes will be used to check the presence/absence of these regions as originally described by Kamerbeek et al in 1997 ! However, herein, these will be "*in silico* probes" and since the sequence length of these probes is 25 nt this can even be applied to early protocols of NGS sequencing generating 36 bp reads.

As mentioned, both Spotyping and Spolpred screen the query sequence reads against a probe database. The screening method differs with Spotyping being based on BLAST search and therefore faster than Spolpred. Both methods are valid and for the sake of speediness we will proceed with Spotyping.

Spotyping is written in Python programming language. However, it requires Python 2.7 to operate correctly. Let's check our python version:

```
$ python -V
```

Which Python version do we have?

Probably you noticed that we have Python 3.7 installed and, while this is a more updated version there are some differences. To facilitate, he Spotyping.py script has been modified with the following line of text right in the beginning:

#!/usr/bin/python2

Python 2.7 is still installed but the command is now python2 and is in /usr/bin/. Regardless, this first line added to the Spotyping script automatically tells the system which interpreter is required for the script. So just type:

```
$ SpoTyping.py
```


You can notice that it outputs the usage, to know more:

```
$ SpoTyping.py -h
```

Take some time to read the different options.

Usage: SpoTyping.py [options] FASTQ_1/FASTA FASTQ_2(optional)

Options:

--version show program's version number and exit

-h, --help show this help message and exit

--seq Set this if input is a fasta file that contains only a complete genomic sequence or assembled contigs from an isolate [Default is off]

-s SWIFT, --swift=SWIFT swift mode, either "on" or "off" [Default: on]

-m MIN, --min=MIN minimum number of error-free hits to support presence of a spacer [Default: 5]

-r MIN_RELAX, --rmin=MIN_RELAX minimum number of 1-error-tolerant hits to support presence of a spacer [Default: 6]

-O OUTDIR, --outdir=OUTDIR output directory [Default: running directory]

-o OUTPUT, --output=OUTPUT basename of output files generated [Default: SpoTyping]

--noQuery Suppress the SITVIT database query [Default is off]

-d, --debug enable debug mode, keeping all intermediate files for checking [Default is off]

Now, from your home directory go to Module4 directory:

```
$ cd Module3
```

It is enough to use one of the paired-end files. Copy the first one to this directory…

```
$ cp ../course_files/PT000033_1.fastq.gz .
```

Done? Now, let's start *in silico* spoligotyping! Type:

```
$ SpoTyping.py PT000033_1.fastq.gz -o PT000033_1.out
```

It should take a minute or so to complete. Once it does check the files in the directory. There is a log file, the output file and an excel spreadsheet with the results from SITVIT query.

To see the profile just type:

```
$ more PT000033_1.out
```

Now have a look at the spreadsheet and at the log file. What do you see?

In the end, what SIT and clade does this strain belong to? Is it expected in Portugal?

Feel free to try with other strains in the course files directory. What do you find?

Exercise 2 – Resistance Prediction: Mykrobe Predictor

Next exercise! Let's go for resistance prediction! We will start with Mykrobe Predictor.

```
$ conda activate mykrobe

$ mykrobe predict PT000033 tb -1 PT000033_1.fastq.gz > PT33.csv

$ conda deactivate
```

This will run the Mykrobe Predictor pipeline using the tb database. Notice the tb argument. The latter is of special importance as Mykrobe Predictor also does AMR prediction for *Staphylococcus aureus*, which uses the staph parameter instead. In this exercise we will use only one set of reads and as soon as the pipeline completes the results will be in the PT33.csv file which can be opened with LibreOffice Calc in this virtual machine. That option is available if you right click the file. Also, there is an option that you can explore if you are interested which is to output the results to a JSON file which provides extra information in a hierarchically structured fashion. Can you find out what is the argument for that?

Well, once it completes you can check the results. In the meanwhile, within the course_files directory there are three compressed FASTQ files from three strains that we haven't covered yet:

- PT000040 (PT000040_1.fastq.gz)
- PT000286 (PT000286_1.fastq.gz)
- ERR1034816 (ERR1034816_1.fastq.gz)

Copy the respective FASTQ files for the working directory of this module and process these with the Mycrobe Predictor pipeline:

```
$ conda activate mykrobe

$ mykrobe predict PT000286 tb -1 PT000286_1.fastq.gz > PT286.csv

$ mykrobe predict PT000040 tb -1 PT000040_1.fastq.gz > PT40.csv

$ mykrobe predict ERR1034816 tb -1 ERR1034816_1.fastq.gz > ERR1034816.csv

$ conda deactivate
```


Once it completes you can check the results in the CSV files!

Exercise 3 – Resistance Prediction: TB Profiler

Let's switch to an alternative tool! We will use TB-Profiler which also reports drug resistance and (sub)lineage. The command-line version of TB profiler has different functionalities depending if you want to collate the results of previous runs, carry out the complete pipeline or, execute the pipeline for VCF files. We will use the profile function which carries out all the pipeline including mapping and variant calling. It is also possible to skip the mapping stage if you already possess a BAM file. In that case you use the BAM file as input instead of raw reads. Take a moment to look at the help page by typing:

```
$ conda activate tb-profiler
$ tb-profiler profile -h
```

Something like this will show up:

```
usage: tb-profiler profile [-h] [--platform {Illumina,minION}] [--read1 READ1]
                           [--read2 READ2] [--bam BAM] [--prefix PREFIX]
                           [--db DB] [--external_db EXTERNAL_DB]
                           [--mapper {bwa,minimap2,bowtie2}]
                           [--caller {BCFtools,GATK}] [--call_whole_genome]
```

```
                [--min_depth MIN_DEPTH] [--af AF]
                [--reporting_af REPORTING_AF] [--threads THREADS]
                [--dir DIR] [--call_method {low,high,optimise}]
                [--txt] [--csv] [--pdf] [--html]
                [--add_columns ADD_COLUMNS] [--meta META]
                [--webserver] [--verbose {0,1,2}] [--plot_cov]
                [--min_gene_frac MIN_GENE_FRAC]
                [--format {classic,new,tex}]
```

optional arguments:

```
 -h, --help            show this help message and exit
 --platform {Illumina,minION}, -m {Illumina,minION}
                       NGS Platform used to generate data (default: Illumina)
 --read1 READ1, -1 READ1
                       First read file (default: None)
 --read2 READ2, -2 READ2
                       Second read file (default: None)
 --bam BAM, -a BAM     BAM file. Make sure it has been generated using the
                       H37Rv genome (GCA_000195955.2) (default: None)
 --prefix PREFIX, -p PREFIX
                       Sample prefix for all results generated (default:
                       tbprofiler)
 --db DB               Mutation panel name (default: tbdb)
 --external_db EXTERNAL_DB
                       Path to db files prefix (overrides "--db" parameter)
                       (default: None)
 --mapper {bwa,minimap2,bowtie2}
                       Mapping tool to use. If you are using minION data it
                       will default to minimap2 (default: bwa)
 --caller {BCFtools,GATK}
                       Variant calling tool to use. (default: GATK)
 --call_whole_genome   Call variants on whole genome (Useful if you need to
                       use whole genome variants later) (default: False)
 --min_depth MIN_DEPTH
                       Minimum depth required to call variant. Bases with
                       depth below this cutoff will be marked as missing
                       (default: 10)
 --af AF               Minimum allele frequency to call variants (default:
                       0.1)
 --reporting_af REPORTING_AF
                       Minimum allele frequency to call variants (default:
                       0.1)
```

```
--threads THREADS, -t THREADS
                      Threads to use (default: 1)
--dir DIR, -d DIR     Storage directory (default: .)
--call_method {low,high,optimise}
                      Level of quality stringency used by BCFtools to call
                      variants. High will enable BAQ and set min baseQ to
                      13. Low will diable BAQ and set min baseQ to 0.
                      Optimise will look at the coverage and decide whether
                      to set low or high. Low stringency setting will
                      greatly decrease the number of missing calls for low
                      coverage isolates (default: low)
--txt                 Add text output (default: False)
--csv                 Add CSV output (default: False)
--pdf                 Add PDF output. This requires pdflatex to be installed
                      (default: False)
--html                Add HTML output (default: False)
--add_columns ADD_COLUMNS
                      Add additional columns found in the mutation database
                      to the text and pdf results (default: None)
--meta META           Add meta data from a CSV file to the results. The CSV
                      file must contain a column labelled "id" with the same
                      value as the prefix argument (default: None)
--webserver           Webserver root (default: False)
--verbose {0,1,2}, -v {0,1,2}
                      Verbosity increases from 0 to 2 (default: 0)
--plot_cov            Not currently used (default: False)
--min_gene_frac MIN_GENE_FRAC
                      Not currently used (default: 0.9)
--format {classic,new,tex}, -f {classic,new,tex}
                      Not currently used (default: classic)
```


But, time to start profiling. Let's start with the PT000033 reads. Type the following:

```
$ tb-profiler profile -1 PT000033_1.fastq.gz -p PT000033 --txt --csv
```

You could eventually use a second FASTQ file containing the read-pais or, as mentioned, use the -a argument and a BAM file. Notice the -p argument, it defines the prefix for the result files. By default, TB Profiler outputs results in a JSON format, you can add the --txt and –csv argument so that the program also outputs a TXT and CSV result files, depending on your preference and subsequent file utilization.

In the meanwhile, as TB Profiler runs, look at the commands. Noticing anything familiar?

As soon as TB Profiler completes you can check the results outputted to the results folder.

Before you go and look at them let's also run the TB profiler pipeline for the other strains as it will take some time:

```
$ tb-profiler profile -1 PT000040_1.fastq.gz -p PT000040 --txt –csv

$ tb-profiler profile -1 PT000286_1.fastq.gz -p PT000286 --txt –csv

$ tb-profiler profile -1 ERR1034816_1.fastq.gz -p ERR1034816 --txt –
csv

$ conda deactivate
```


Now that you have run reads from different clinical isolates it is time to compare the results. Please fill in the following table:

Strain	Platform/Software			
	TB Profiler	**Mykrobe**	**PhyResSE**	**Obs.**
PT000033				
PT000286				
PT000040				
ERR1034816				

Tip: For the PT strains you can check the phenotype at http://cplp-tb.ff.ulisboa.pt.

As an alternative to the command-line versions you can also check PhyResSE and TGS-Tb which also perform AMR prediction for TB. These two platforms are web-based. You can open the browser and upload the data. Warning: this analysis takes some time, but if you choose to do it try first PhyResSE as TGS-TB is now relying on TB Profiler and will likely show you the same results you have obtained here.

What are your conclusions?

Part II – Antimicrobial Resistance and BLAST-based software for gene detection

Now that we have introduced some specific tools for M. tuberculosis let's switch to more general tools. In the following exercises we will be using draft genome assemblies as input (before contig ordering and scaffolding) that you can obtain by following the exercises and steps in Module 2. So, the input will be a simple multifasta file containing multiple contigs. These exercises can also be completed using finished chromosome sequences that, e.g., can be found available in public databases.

Also, the assemblies used in this exercise can be obtained with any assembler provided your input is in the fasta format. That is the sole requirement!

Exercise 4 – Extracting resistomic information from genome assemblies

In this first exercise we will aim at finding the genetic determinants for resistance in genome assemblies from Klebsiella pneumoniae. Inside the Module 4 directory there is a sub-directory called Kp_assemblies, inside you can find four fasta files each from a different strain (Kp1-Kp4).

There are different tools available for detection and characterization of the genetic determinants, some implemented in online servers with a graphical interface. By now you should be accustomed with these command-line tools which allow for a more precise parametrization and can be deployed into programmatic use in a pipeline or to large datasets.

Now, all the tools we will now use rely, in one way or another, on BLAST tools. The VM you are using has all BLAST tools installed and allow you to run these locally. As such, these tools end-up by acting as BLAST parsers with some additional features.

The most important caveat to take in account here: these tools are only as good as the databases that are associated. What does it mean? It is useless to be looking for OXA-181 carbapenemase coding genes or the mrk gene clusters if the database does not have those. Also, the annotation of the genes in the database will allow you to integrate additional.

Enough with the talk and let's do some practical work. We will start by using AMRFinderPlus, which implements blastx that takes nucleotide sequences as input and searches across a protein database [8]. This will be the way to go if you need to identify specific alleles (e.g. KPC-2 vs KPC-3) of different resistance genes which only differ in the amino-acid sequence. Doing blastn (nucleotide blast) may also work but it may lead to the wrong identification if synonymous mutations are present. The downside is that this tool is somewhat slower, but it draws all of its database from the NCBI curated database of AMR determinants .

Let's try!

```
$ cd ~/Module3

$ cd Kp_assemblies

# You can try the following:

$ amrfinder --nucleotide Kp1.fasta --plus

# This will output to the screen. It is best if you redirect to a file:

$ amrfinder --nucleotide Kp1.fasta --plus > Kp1_amrfinder.tsv

$ amrfinder --nucleotide Kp2.fasta --plus > Kp2_amrfinder.tsv

$ amrfinder --nucleotide Kp3.fasta --plus > Kp3_amrfinder.tsv

$ amrfinder --nucleotide Kp4.fasta --plus > Kp4_amrfinder.tsv

$ amrfinder --nucleotide Kp5.fasta --plus > Kp5_amrfinder.tsv

# Notice the .tsv extension, it just informs you that this is a tab
separated value file.
```

Now, why is it called AMRFinderPlus with the "Plus"? Plus what? In addition to resistance genes if you add the --plus argument (which I hope you did) it will also screen for metal resistance genes, resistance to biocides or a limited number of known virulence genes.

You can examine the output files using the cat or the more commands or you can open those in LibreOffice Calc. Just right-click and select open with LibreOffice Calc.

Look at the files. Do all bear genes capable of hydrolysing carbapenems? And Extended Spectrum Beta-Lactamases?

Notice there are columns for the identity and coverage. What would be good cut-offs here?

Optional:

Check, for example, if there are resistance genes mapping to the same contig in Kp?

What are the consequences?

Exercise 5 – *K. pneumoniae* capsule, O antigen and MLST

Next, we will use another tool that although capable of detecting resistance genes it is specifically developed for Klebsiella and the determination of the Sequence Type (MLST) capsular locus and O antigen locus.

```
# Try the following

$ kleborate-runner.py --all -a Kp1.fasta

# As in the previous example, this will output to the screen. With the
following command you can do all files at once and redirect to the same
file:

$ kleborate-runner.py --all -a *.fasta -o kleborate_results.tsv

# Notice the .tsv extension, this is also a tab separated value file.
```

Let's examine the output of Kleborate [9, 10]. Besides species identification and assembly statistics, the output contains:

- ST
- Virulence and resistance scores
- Presence/absence of specific virulence genes
- K Locus type
- Antigen O type
- AMR genes

You should look at the online Kleborate Github/Wiki page to check any doubts you might have considering the output (https://github.com/katholt/Kleborate/wiki).

Do all these strains belong to the same ST? Do you see any difference regarding drug resistance genes across strains that belong to the same ST? Does that reflect at K and O *loci*?

Exercise 6 – Searching for different plasmids and virulence factors.

The final exercise for this module will make use of another tool called Abricate (author: Torsten Seemann) which already comes with a diversity of databases (https://github.com/tseemann/abricate).

```
$ cd ../Ec_assemblies

# Try the following

$ abricate
```

You will get something like:

```
Synopsis:
    Find and collate amplicons in assembled contigs
Author:
    Torsten Seemann <torsten.seemann@gmail.com>
Usage:
        % abricate --list
        % abricate [options] <contigs. {fasta,gbk,embl}[.gz]> > out.tab
        % abricate --summary <out1.tab> <out2.tab> <out3.tab> ... > summary.tab
Options:
  --help          This help.
  --debug         Verbose debug output (default '0').
  --quiet         Quiet mode, no stderr output (default '0').
  --version       Print version and exit.
  --setupdb       Format all the BLAST databases (default '0').
  --list          List included databases (default '0').
  --check         Check dependencies are installed (default '0').
  --summary       Summarize multiple reports into a table (default '0').
  --datadir [X]   Location of database folders (default '/home/centos/miniconda3/bin/../db').
  --db [X]        Database to use (default 'resfinder').
  --noheader      Suppress column header row (default '0').
  --csv           Output CSV instead of TSV (default '0').
  --minid [n.n]   Minimum DNA %identity (default '75').
  --mincov [n.n] Minimum DNA %coverage (default '0').
  --nopath        Strip filename paths from FILE column (default '0').
Documentation:
  https://github.com/tseemann/abricate
```

Notice that the default database comes is the resfinder one and, as such, if you don't select for a different one you will get the resistance genes.

To find out which other databases are available do the following:

```
$ abricate --list
```

This will show the following:

DATABASE	SEQUENCES	DATE
argannot	1749	2018-Jul-16
card	2220	2018-Jul-16

ecoh	597	2018-Jul-16
ncbi	4324	2018-Jul-16
plasmidfinder	263	2018-Jul-16
resfinder	2280	2018-Jul-16
vfdb	2597	2018-Jul-16

Notice that these databases that come with abricate are somewhat outdated and in a real-life situation you should check if these databases have newer versions.

Another neat aspect of abricate is that it allows for you to add additional databases. You just need to have the the sequences that need to be searched, formatted in a fasta format, and you can include those in the software by following online instructions.

We will test this on another set of assemblies. Inside the Module4 directory, besides the Kp_assmblies there is also a Ec_assemblies sub-directory. The latter contains two *E. coli* assemblies. We will screen for plasmids and virulence factors using the plasmidfinder and vfdb databases respectively.

```
$ abricate --db plasmidfinder Ec1.fasta > Ec1_plasmidfinder.tsv

$ abricate --db plasmidfinder Ec2.fasta > Ec2_plasmidfinder.tsv

$ abricate --db vfdb Ec1.fasta > Ec1_vfdb.tsv

$ abricate --db vfdb Ec2.fasta > Ec2_vfdb.tsv

## Instead, if you prefer csv files you can try adding the --csv option.
```

Abricate offers another neat function to compare across result files:

```
$ abricate --summary Ec1_plasmidfinder.tsv Ec2_plasmidfinder.tsv >
plasmids_summary.tsv

$ abricate --summary Ec1_vfdb.tsv Ec2_vfdb.tsv > vfdb_summary.tsv


## Beware that the values in the cells is the coverage, you lose the
percentage identity. You might want to set initial thresholds using the
--minid and --mincov options. The drawback is that for plasmids this can
vary…

Let's test for virulence factors:

$ abricate --db vfdb --minid 90 --mincov 60 Ec1.fasta >
Ec1_vfdb_filtered.tsv

$ abricate --db vfdb --minid 90 --mincov 60 Ec2.fasta >
Ec2_vfdb_filtered.tsv

$ abricate --summary Ec1_vfdb_filtered.tsv Ec2_vfdb_filtered.tsv >
vfdb_summary_filtered.tsv
```


Tips: Notice that the -cd option controls the percentage of isolates a gene must be in to be in the core set (by default 99%). You can tweak this a bit but also be mindful that you need to have a larger dataset to notice any differences.

Optional Exercise:

Exercise 7 – Pan/Core genome and Comparative Gene Content Analysis

This optional exercise uses the Roary pipeline for calculation of the pan genome across a set of samples. As input Roary takes GFF3 files which are annotation files (you may recall these from Module 2), ideally produced by Prokka [11-13].

Take a look at roary's options:

```
Usage:   roary [options] *.gff

Options: -p INT    number of threads [1]
         -o STR    clusters output filename [clustered_proteins]
         -f STR    output directory [.]
```

```
        -e          create a multiFASTA alignment of core genes using
PRANK
        -n          fast core gene alignment with MAFFT, use with -e
        -i          minimum percentage identity for blastp [95]
        -cd FLOAT   percentage of isolates a gene must be in to be core
[99]
        -qc         generate QC report with Kraken
        -k STR      path to Kraken database for QC, use with -qc
        -a          check dependancies and print versions
        -b STR      blastp executable [blastp]
        -c STR      mcl executable [mcl]
        -d STR      mcxdeblast executable [mcxdeblast]
        -g INT      maximum number of clusters [50000]
        -m STR      makeblastdb executable [makeblastdb]
        -r          create R plots, requires R and ggplot2
        -s          dont split paralogs
        -t INT      translation table [11]
        -ap         allow paralogs in core alignment
        -z          dont delete intermediate files
        -v          verbose output to STDOUT
        -w          print version and exit
        -y          add gene inference information to spreadsheet,
doesnt work with -e
        -iv STR     Change the MCL inflation value [1.5]
        -h          this help message
```

You may have noticed that multiple options are available and besides the analysis of the pan-genome, roary can also produce a core genome alignment. Depending on the approach which can be useful for the phylogenetic analysis module in this course.

We will use the GFF3 files produced by Prokka for Kp2, Kp4 and Kp5. Why? These represent ST15 strains detected in Portugal, but each has a different resistance enzymes. So, what genes are common between strains and which are not?

```
$ cd ~/Module4/Kp_gff

$ conda activate roary

$ roary -v -f Kp_pan *.gff

# The -v switches on the verbose mode so you can keep track of what is
going on.

# Alternatively, you can produce a core genome analysis but this will take
longer (~1h) and will not be suitable to do in teaching sessions as it
will align all genes individually:

$ roary -e --mafft -cd 99 -f Kp_core -r -v *.gff

# A folder containing the results is provided for the previous command.

$ conda deactivate
```


You can look at the roary's documentation online to check the output files and what these contain. Let's look at the summary_statistics.txt and the gene_presence_absence.csv files. This latter one can be opened in LibreOffice Calc for easier visualization.

Anything interesting? What are the total genes across this toy dataset and how many genes are core genes (99-100% presence)?

In the Kp_core folder [core genome analysis] there is a *core_gene_alignment.aln* file which can be used for phylogenetic purposes and comparison of clonality between isolates. You can reduce this alignment to the segregating sites using snp-sites:

```
# Let's switch to another directory containing more GFF3 files with the
output of a Roary core genome analysis:

$ cd ~/Module4/Kp_gff2/Kp_core

Look at the core_gene_alignment.aln

$ nano core_gene_alignment.aln

# press CTRL+X to exit.

# Now try:

$ snp-sites -o core_gene_alignment_segregant.aln core_gene_alignment.aln

# This will output the core_gene_alignament_segregant.aln file. You can
see that it is considerably smaller.
```


References

1 van Heusden P, Mashologu Z, Lose T, Warren R, Christoffels A. The COMBAT-TB Workbench: Making Powerful Mycobacterium tuberculosis Bioinformatics Accessible. *mSphere*. 2022; **7**: e0099121.

2 Groschel MI, Owens M, Freschi L, et al. GenTB: A user-friendly genome-based predictor for tuberculosis resistance powered by machine learning. *Genome Med*. 2021; **13**: 138.

3 Coll F, McNerney R, Preston MD, et al. Rapid determination of anti-tuberculosis drug resistance from whole-genome sequences. *Genome Med*. 2015; **7**: 51.

4 Feuerriegel S, Schleusener V, Beckert P, et al. PhyResSE: a Web Tool Delineating Mycobacterium tuberculosis Antibiotic Resistance and Lineage from Whole-Genome Sequencing Data. *J Clin Microbiol*. 2015; **53**: 1908-1914.

5 Hunt M, Bradley P, Lapierre SG, et al. Antibiotic resistance prediction for Mycobacterium tuberculosis from genome sequence data with Mykrobe. *Wellcome Open Res*. 2019; **4**: 191.

6 Coll F, Mallard K, Preston MD, et al. SpolPred: rapid and accurate prediction of Mycobacterium tuberculosis spoligotypes from short genomic sequences. *Bioinformatics*. 2012; **28**: 2991-2993.

7 Xia E, Teo YY, Ong RT. SpoTyping: fast and accurate in silico Mycobacterium spoligotyping from sequence reads. *Genome Med*. 2016; **8**: 19.

8 Feldgarden M, Brover V, Gonzalez-Escalona N, et al. AMRFinderPlus and the Reference Gene Catalog facilitate examination of the genomic links among antimicrobial resistance, stress response, and virulence. *Sci Rep*. 2021; **11**: 12728.

9 Lam MMC, Wick RR, Watts SC, Cerdeira LT, Wyres KL, Holt KE. A genomic surveillance framework and genotyping tool for Klebsiella pneumoniae and its related species complex. *Nat Commun*. 2021; **12**: 4188.

10 Wyres KL, Wick RR, Gorrie C, et al. Identification of Klebsiella capsule synthesis loci from whole genome data. *Microb Genom*. 2016; **2**: e000102.

11 Sitto F, Battistuzzi FU. Estimating Pangenomes with Roary. *Mol Biol Evol*. 2020; **37**: 933-939.

12 Page AJ, Cummins CA, Hunt M, et al. Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics*. 2015; **31**: 3691-3693.

13 Seemann T. Prokka: rapid prokaryotic genome annotation. *Bioinformatics*. 2014; **30**: 2068-2069.

:: Introduction to Phylogenetics and Public Health::

Introduction

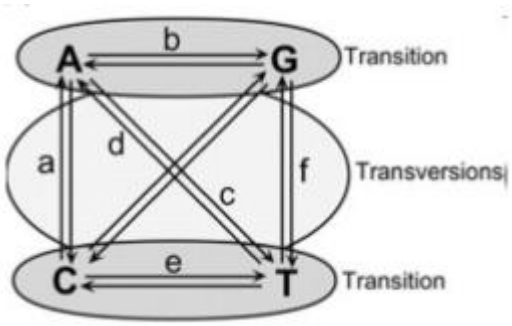
Phylogenetics pertains the **study of the evolutionary relationships** between organisms (species, strains, etc), genes or even genomes and, should represent the evolutionary history of such entities by showing how closely related they are [1, 2]. Usually this is depicted as a phylogenetic tree. Initially, phylogenetic reconstruction relied on morphological and physiological characters but the problem with to using such characters in phylogenetic analysis are: i) these are prone to convergent evolution which leads to homoplasies; and, iii) the number of characters to infer from are often limited. With advances in molecular biology, DNA and protein sequences rapidly became the preferred characters for phylogenetic analysis. From molecular-based phylogenies, **we can infer on how a given sequence has reached its current state** (*e.g.* how has it changed in the face of selective pressures) or **how is it expected to change in the future**. In the scope of this course, phylogenetic analysis can be an invaluable tool to identify the origin of pathogens, how they have spread and even **shed some light on cryptic transmission chains** that were otherwise undetected by conventional epidemiological investigation. In this regard, phylogenies are an increasingly recognized tool to **evaluate and extract biologically relevant information** from large molecular datasets [1-3].

To start a phylogenetic analysis, you generally **start from a multiple sequence alignment** constructed from a given dataset. The fundamental principle in a multiple sequence alignment is, that each column in the alignment should contain homologous residues, that is, with the same evolutionary origin and not the result of insertions or deletions in the sequence. As such, every position in an alignment can be considered an evolutionary marker and given the length of a simple gene it becomes immediately clear the increased robustness of molecular-based phylogenetics. It is therefore important to retain that a proper alignment contains an enormous amount of information.

Constructing a phylogenetic tree from a multiple sequence alignment can be done using methods that generally fall within two categories: **distance-matrix methods** (e.g. Unweighted Pair Group Method with Arithmetic Mean [UPGMA] or Neighbour-Joining) or **discrete data methods** (e.g. Maximum Likelihood or Bayesian methods). Distance based methods look at all pairwise comparison between sequences to calculate a simple distance metric, usually the percent of sequence difference to group isolates into a tree,

while discrete data methods look at each column of the alignment and calculate the best tree that represents the individual information at each of these sites [1, 3].

Another factor affecting the way we can model the distances at each position and affect the genetic distances between isolates pertains to the fact that different sequences can evolve under different ways, under different models. For this, it is necessary to statistically model the substitution of nucleotides using a **nucleotide substitution model**. The simplest of these models is the Jukes and Cantor model introduced in 1969 (**JC69**) which assumes that the equilibrium frequencies for all four nucleotides are 25% and that any nucleotide has an equal probability of being replaced by any other nucleotide. On the other hand, the Kimura 1980 model (**K80**) distinguishes between transversions and transitions; and the General Time Reversible model (**GTR**) assumes a different rate for every possible substitution:



- JC69 : a=b=c=d=e=f
  A=C=G=T=1/4
- K80 : b=e, a=c=d=f
  A=C=G=T=1/4
- HKY85 : b=e, a=c=d=f
  A ≠ C ≠ G ≠ T
- TN93 : b, e, a=c=d=f
  A ≠ C ≠ G ≠ T
- GTR : a, b, c, d, e, f
  A ≠ C ≠ G ≠ T

There are other parameters that increase the complexity of these models of nucleotide substitution: unequal base frequencies (+F); varying proportions of invariant sites (+I); or variations in the substitution rate across sites (+G). A sequence may be evolving under any of these models, so it is important to determine which is the best fit to our data [2, 3].

But, how to evaluate the robustness of a tree and its branches? The most widely used test is called **bootstrapping**, where it randomly samples the entire alignment dataset column by column with replacement until achieving a sequence alignment with the length of the original one. This process is repeated several times (usually 1000 iterations) and the entire tree is recalculated for the pseudo-alignment every single time. By the end, all branches of the original tree are compared to check its presence in the trees obtained from the alignment sampling process. If a branch is present in 70% of these latter trees, it is generally assumed that this is a reliable grouping [1].

This module is composed by a set of 4 simple exercises in which you will be able construct phylogenetic trees from HIV sequence data and from WGS data obtained from *M. tuberculosis* clinical isolates. Also, you will need to assess the distances between these isolates using simple distance metrics as well as more robust models of molecular evolution and translate the latter into a phylogenetic tree that could be used to assess TB transmission dynamics and microevolution. The number of isolates to be used will be limited to cope with the limitations of the course computing system.

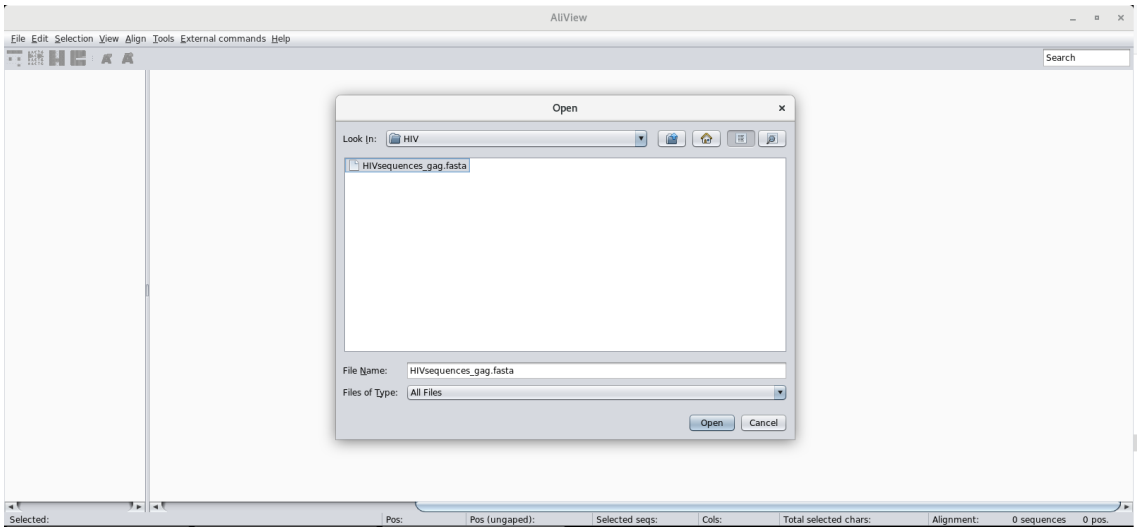
Exercise 1 – A primer on phylogenetic reconstruction: HIV transmission case-study

To get us started in this topic you will look into HIV sequence data for the *gag* gene. The *gag* gene is less diverse than the *env* gene and we will use it to investigate the possible transmission between different individuals. The data you will use has been generated to investigate the possible transmission between three individuals: MP1 and MP2 were married in the past and after their divorce MP1, having found to be infected with HIV-1, prompted an accusation that she had been infected by her ex-husband (MP2). Later, MP3, the current girlfriend of MP2 was also diagnosed with HIV-2. Different clones were sampled at the time of diagnosis by amplification of the env and gag genes. Extensive phylogenetic analysis using this transmission case has been published [4]:

- Romero-Severson EO, Bulla I, Hengartner N, Bartolo I, Abecasis A, Azevedo-Pereira JM, Taveira N, Leitner T. Donor-Recipient Identification in Para- and Poly-Phyletic Trees Under Alternative HIV-1 Transmission Hypotheses Using Approximate Bayesian Computation. Genetics 2017 Nov 207(3):1089-1101 PMID: 28912340 https://doi:10.1534/genetics.117.300284

A multifasta file containing the PCR amplified sequences of the subjects along with several local controls and additional sequences publicly available is available under the "HIV" subdirectory of "Module5" directory. You will need to i) align the sequences; ii) edit and trim the alignment; iii) determine the most adequate model for phylogenetic reconstruction; and, iv) perform the phylogenetic reconstruction and visualize the tree.

To start open Aliview (there is a shortcut in the desktop) and load the multifasta file (named *HIVsequences_gag.fasta*)



This should open all sequences: 117 sequences for a total of 434 positions. Are the sequences aligned? Take a look at the sequences …

To align the sequences go to *Align* > *Realign everything* … You will be asked if you want to realign the entire alignment. It's OK! This will align all sequences using MUSCLE.

Now comes the part where you need to inspect the alignment and edit it. It is important that the alignment maintains its codon structure and Aliview offers the possibility of toggling between nucleotide, codon and protein view. Use this to check the alignment! You will find the corresponding buttons in the tool bar. Another important point is to delete insertions that are absent for mor than 50% of the sequences. Do this by going into the Edito mode (*Edit* > *Edit mode*), selecting the columns you wish to delete and delete those (Ctrl+Delete). Make sure the sequences in the alignment start and end at the same position.

Next save (*File* > *Save as Fasta*) the alignment to the same directory, please name it *HIVsequences_gag_aligned.fasta!*

We will next use IQTree to perform both phylogenetic reconstruction and determine the best fit nucleotide substitution model to our data [5]. IQTree can do both in one single command to check multiple examples just type from the command line:

```
$ cd ~/Module5
$ iqtree
# For more detailed instructions:
$ iqtree -h
```

To estimate the best-fit model we will run only the ModelFinder module (MF) [6]:

```
$ iqtree -s HIVsequences_gag_aligned.fasta -m MF
```

According to ModelFinder what is the model that best fits the data: _____

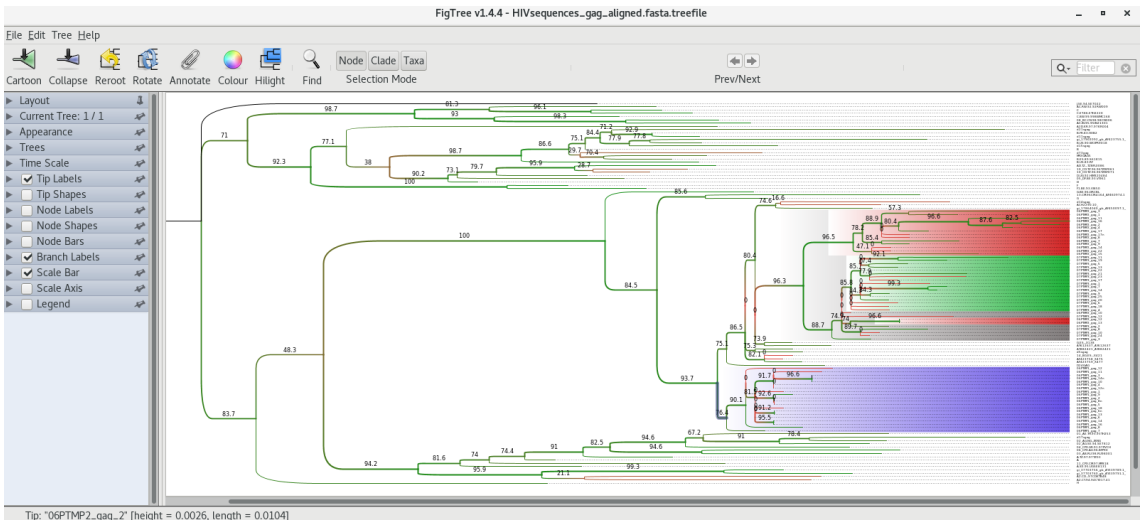
To infer the maximum-likelihood tree with the best fit model and test for branch support using SH-aLRT:

```
$ iqtree -s HIVsequences_gag_aligned.fasta -alrt 1000

# Notice that you are specifying 1000 replicates for SH-aLRT.
```

After this step you should have the tree in the Newick format stored in a file named *HIVsequences_gag_aligned.fasta.treefile*.

You can now visualize the tree in FigTree, or try the online tool iTOL [7], Open the tree with FigTree (there is a shortcut in the desktop). What can you conclude from the tree topology? Does it support the patient accusation?

In the meanwhile check the visualization options of FigTree, namely the branch support calculated by IQTree.



Can you get something like this?

Exercise 2 – Core-genome SNP Alignment

Now that you are more acquainted with phylogenetic inference, let's try to pick up from the previous modules, we are going to perform a simple phylogenetic analysis of five M. tuberculosis clinical isolates:

- PT000033
- PT000049
- PT000050
- PT000271
- PT000279

You can look up at these isolates characteristics such as spoligotyping lineage, shared-type or drug resistance profile in http://cplp-tb.ff.ulisboa.pt.

The idea here is to start by constructing a DNA pseudo-molecule from concatenated SNPs. Of course, this will vary according to the isolates that we are about to use, if we included another isolate with a divergent sub-set of SNPs it would result in a longer sequence. These SNPs will be obtained from comparison against a reference sequence using a mapping approach similar to Module 1. This reference sequence will be the same as in Module 1: *M. tuberculosis* H37Rv (GenBank accession: NC000962.3). Whith this in mind, alignement won't be necessary since we will be constructing this sequence from a SNP table. Such alignment can be called a core SNP alignment since each SNP occurs at a core site, that is, a site that is present at all sites.

In this exercise we will use Snippy, which finds SNPs between a haploid reference genome and your NGS sequence reads (https://github.com/tseemann/snippy). It will find both substitutions (SNPs) and insertions/deletions (indels). This script is designed with speed in mind, and produces a consistent set of output files in a single folder. It can then take a set of Snippy results using the same reference and generate a core SNP alignment (and ultimately a phylogenomic tree).

Let's get started, open a terminal window and type:

```
$ cd Module5

$conda activate snippy

$ snippy --cpus 1 --outdir PT000033 --ref
../course_files/NC000962_3.gbk --R1 ../course_files/PT000033_1.fastq.gz
--R2 ../course_files/PT000033_2.fastq.gz
```

The command above specifies: the number of CPU cores to be used, one in this case due to limitations in the computing system (usually with a quad-core computer you can specify 8 CPUs); an output directory containing the output files (using the --outdir option); the reference (--ref) genbank file of the reference and your reads.

If everything runs ok the output files will be in the PT000033 directory. To go there type:

```
$ cd PT000033
```

Check if the following output files are present:

Extension	Description
.tab	A simple tab-separated summary of all the variants
.csv	A comma-separated version of the .tab file
.html	A HTML version of the .tab file
.vcf	The final annotated variants in VCF format
.vcf.gz	Compressed .vcf file via BGZIP
.vcf.gz.tbi	Index for the .vcf.gz via TABIX
.bed	The variants in BED format
.gff	The variants in GFF3 format
.bam	The alignments in BAM format. Note that multi-mapping and unmapped reads are not present.
.bam.bai	Index for the .bam file
.raw.vcf	The unfiltered variant calls from Freebayes
.filt.vcf	The filtered variant calls from Freebayes
.log	A log file with the commands run and their outputs
.consensus.fa	A version of the reference genome with all variants instantiated
.aligned.fa	A version of the reference but with - at position with depth=0 and N for 0 < depth < --mincov (**does not have variants**)
.depth.gz	Output of samtools depth for the .bam file
.depth.gz.tbi	Index for the .depth.gz (*currently unused*)

For further information on file columns, fields or variant types you can check Snippy's documentation at: https://github.com/tseemann/snippy (Torsten Seemann)

Notice that Snippy already produces a Bam and annotated VCF file!

Let's go back to the Module3 directory:

```
$ cd ..

## Now let's look at what is in this directory:

$ ls
```

After listing its contents you may have realize that this directory already has the four other Snippy sub-directories for the remaining isolates. These have been computed earlier using commands similar to the one above.

For this next step it is necessary to ensure that all Snippy directories have been created using the same reference genome. This is the case. Now we will use the snippy-core command to produce the core SNP listing and sequences:

```
$ snippy-core PT000033 PT000049 PT000050 PT000271 PT000279 --ref
../course_files/NC000962_3.gbk

$ conda deactivate
```

This command will read-in the output files from the previous snippy commands (and within the respective directories) and produce another set of output files containing data from all strains specified above. List the contents of the present directory and you should find the following output files:

Extension	Description
.aln	A core SNP alignment in the --aformat format (default FASTA)
.full.aln	A whole genome SNP alignment (includes invariant sites)
.tab	Tab-separated columnar list of core SNP sites with alleles and annotations
.txt	Tab-separated columnar list of alignment/core-size statistics

The file core.aln will contain the core SNP alignment in FASTA format. Let's look at it:

```
$ more core.aln
```

Does it seem ok?

At a first glimpse it might, but when dealing with genome-wide SNPs you might want to filter out some of these because some of these positions are associated with hard-to-map regions usually due to its repetitive nature. This is the case for PE/PPE genes which are usually not considered for *M. tuberculosis* phylogenetic analysis. As such, we will disregard this FASTA core SNP alignment file and we will build a new one from the core.tab file. This file contains the SNPs along with the respective position on the reference genome along with some data on gene product and gene id associated with the SNPs.

Give this file a look by typing:

```
$ more core.tab
```

Now, to use this file we will use an R-written script to concatenate the columns of this table but not before removing:

- Positions associated with PE/PPE genes;

- Positions with a unique K-mer length below 49/50 – gives an estimate of the mappability of the region by considering only positions that have a unique K-mer ending at that position of 49/50 bp in length (with the k-mer length set to 56bp).

This script requires some pre-computed data containing the k-mer scores at each position and the positions associated with PE/PPE genes (at ~/library_files/). To execute this script just type (assuming you are still within the ~/Module3/PT000033/ directory):

```
$ SNPtable_filter_Mtb.R core.tab
```

This command should have produced a file called *coreSNP_alignment_filtered.fas* from the SNP table. Let's look at it:

```
$ more coreSNP_alignment_filtered.fas
```

Similar to the core.aln file, right? The difference is that it does not contain a number of positions that may effectively influence your phylogenetic analysis.

By the end of this exercise you should have a file containing your aligned SNPs, ready for phylogenetic reconstruction.

Exercise 3 – Estimating simple distances: Hamming distance

For this quick exercise we will compare sequences by simply counting the number of difference between these. This distance metric is called the Hamming distance and if we express this as the proportion of the number of sites that are different, we get the p-distance. Of course, these metrics do not account for multiple substitutions, substitution rate biases or differences in the evolutionary rates across sites.

To get the SNP distance separating our isolates we will use another R-written script from the command-line: HammingFasta.R. This script will output a distance matrix on the screen and will produce a csv file (you can try to open it with LibreOffice Calc) containing the matrix. To do this type:

```
$ HammingFasta.R coreSNP_alignment_filtered.fas
```

Did you get something like this on the screen?

What does this mean? What can be the public health implications from this comparison?

Which strains can be from patients with putative epidemiological links?

What can you learn from these strains in CPLP-TB database (http://cplp-tb.ff.ulisboa.pt)?

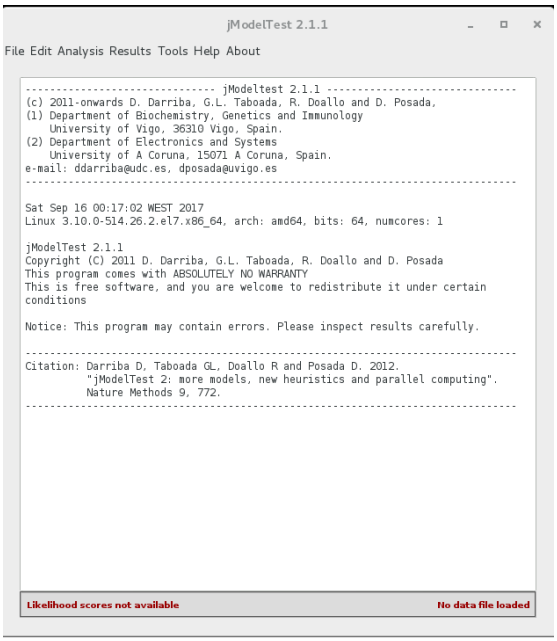
Exercise 4 – Phylogenetic Reconstruction

In the previous exercise you have already evaluate pairwise distance between isolates, but you did not obtain any phylogeny. Although it would be straightforward if we want to know in more detail the transmission dynamics and how these strains are evolving we need more robust methods to reconstruct the phylogeny of these strains.

In this exercise we will use Maximum Likelihood methods to do that. We will start with the alignment in the *coreSNP_alignment_filtered.fas* file. By the way, we are done with the command-line for the day. From now on it's just clicking!

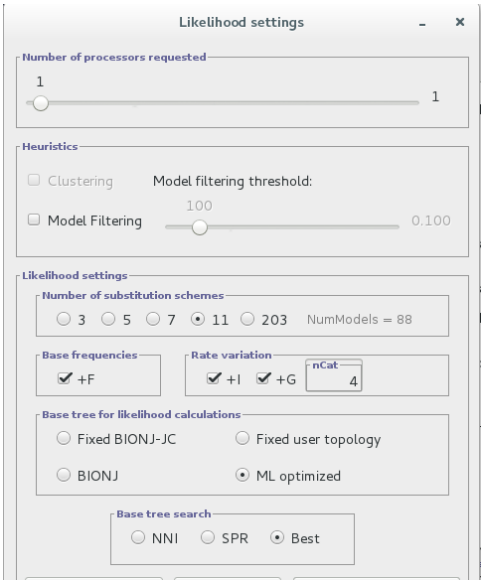
The first step will consist in evaluating what is the model of nucleotide substitution that best fits our data. To do that we will use a program called jModelTest2, which is written in JAVA [8]. To start it just double-click on its shortcut that you can find in the desktop.

Once it starts, you should see something like:



Now, go to *File > Load DNA alignment* and open your alignment file (*coreSNP_alignment_filtered.fas*). Once it opens it will tell you the number of sequences and the number of sites in your alignment.

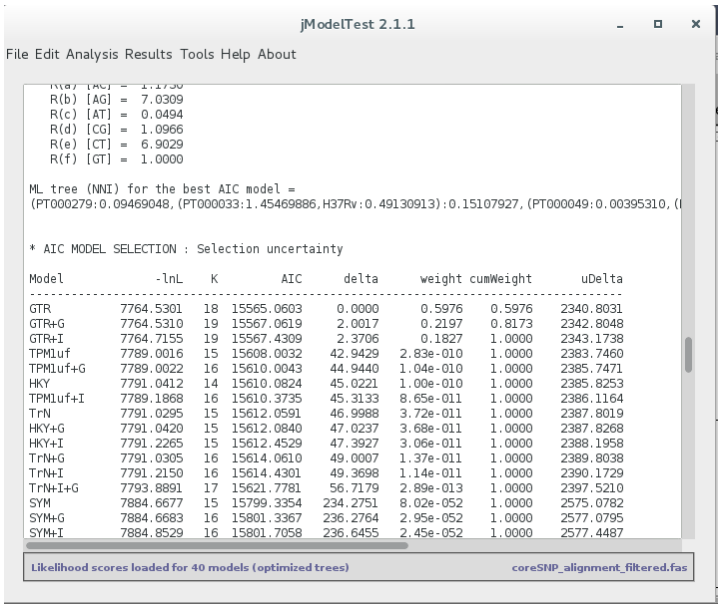
Next, go to Analysis > Compute likelihood scores:



Here you can choose the Likelihood settings. We will keep with the default settings except (to speed-up the analysis) we will only test 5 substitution schemes: Check that unequal base frequencies (+F), rate variation (+G) and invariant sites (+I) are selected. Choose a base tree for likelihood calculation that is ML optimized and for base tree search choose Best of NNI and SPR.

Once this analysis is over the software has calculated the likelihood score for each model. This likelihood score is usually a very low number and it is usually to be presented as the negative of its natural logarithm (-lnL). The lower this -lnL value the more likely the model is. Different model comparison algorithms are implemented in jModelTest. After the module evaluation you can go to Analysis menu and calculate the Akaike information Criterion (AIC), Bayesian information criterion (BIC) and Decision theory method. You can run all three tests with standard settings. The output will be on the screen, you have to scroll up to see the model list from each method, the best model comes first, notice its -lnL value and the weight column.

Here is an example for the AIC method:



According to the output of these programs, which model presents the best fit to our dataset? _____

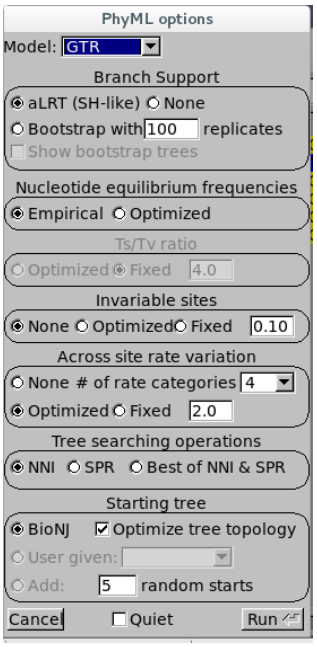
Do all the methods agree on it? _____

And the second best?

AIC is more liberal than BIC and penalizes free parameters less strongly, which may lead to preference for more complex models when simpler models may be a good fit as well.

What can we do if there is no agreement between decision methods?

Since all methods appear to agree on GTR, we will stick with it for the last part. We will reconstruct the phylogenetic tree of these isolates. For this purpose, we will use Seaview. Open Seaview using the shortcut on the desktop and go to *File* > *Open* and open the alignment file. The alignment immediately appears in the alignment window. To construct a Maximum Likelihood phylogenetic tree from this data, go to *Trees* > *PhyML*. A window should appear:



Here, we will set the parameters based on the calculations from jModelTest:

Branch Support: choose aLRT (approximate Likelihood Ratio Test) instead of bootstrapping. This is a more recent method based over likelihood gains from the present branch against a null (collapsing the branch) hypothesis. aLRT is, statistically speaking, less obscure than bootstrap and provides you a *p*-value [2, 9]. For example, the probability of a false positive on branch with aLRT score of 0.95 is 5% (0.05).

Nucleotide Equilibrium Frequencies: Choose empirical (it will calculate from sequence) as the model selected was not +F.

Invariable Sites: Choose None as the selected nodel was not +I (otherwise you would choose Optimized to calculate from data or Fixed and you would have to specify a value).

Across site rate variation: Choose None as the model was not +G.

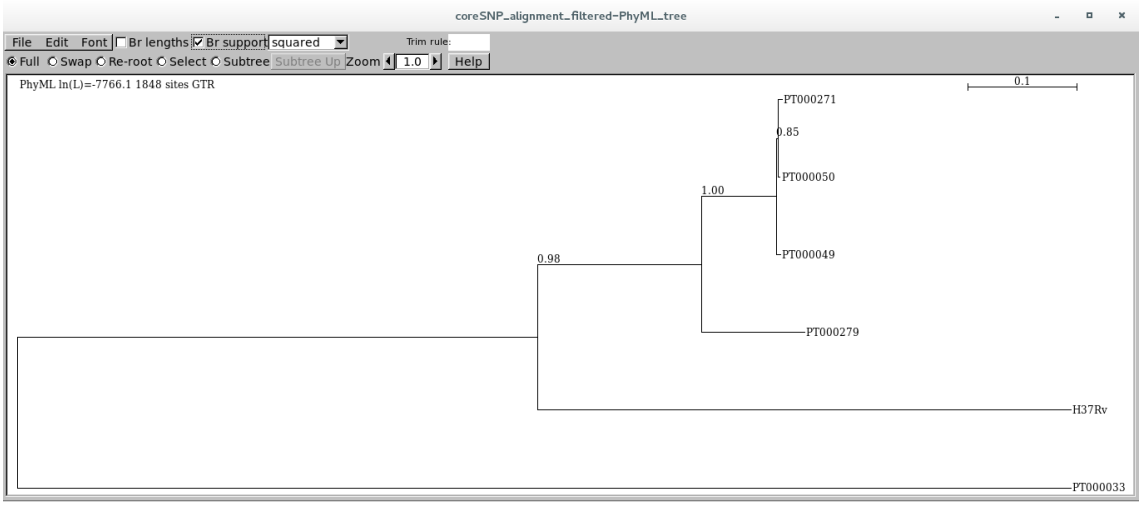
Tree searching operations: Choose Best of NNI & SPR.

Starting Tree: Choose BioNJ to calculate a starting tree and leave *Optimize Tree Selection* selected.

Then click *Run*.

As this alignment does not have many strains it will run rapidly. When the calculations end click OK.

This will show you the tree. You can check *Br support* checkbox to visualize the aLRT scores:



You can try different visualization options other than squared.

What can you tell from this tree? How about the branches, do they seem statistically robust?

Tip: The tree can be saved in the Newick format (parenthetic). Just go to *File* > *Save as rooted/unrooted tree*.

Tip: Try iTOL (Interactive Tree of Life) online tool to annotate your tree after saving in the Newick format

References

1 Baldauf SL. Phylogeny for the faint of heart: a tutorial. *Trends Genet*. 2003; **19**: 345-351.

2 De Bruyn A, Martin DP, Lefeuvre P. Phylogenetic reconstruction methods: an overview. *Methods Mol Biol*. 2014; **1115**: 257-277.

3 Lemey P, Salemi M, Vandamme AM. *The Phylogenetic Handbook: A Practical Approach to Phylogenetic Analysis and Hypothesis Testing*. Cambridge: Cambridge University Press, 2009.

4 Romero-Severson EO, Bulla I, Hengartner N, et al. Donor-Recipient Identification in Para- and Poly-phyletic Trees Under Alternative HIV-1 Transmission Hypotheses Using Approximate Bayesian Computation. *Genetics*. 2017; **207**: 1089-1101.

5 Nguyen LT, Schmidt HA, von Haeseler A, Minh BQ. IQ-TREE: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Mol Biol Evol*. 2015; **32**: 268-274.

6 Kalyaanamoorthy S, Minh BQ, Wong TKF, von Haeseler A, Jermiin LS. ModelFinder: fast model selection for accurate phylogenetic estimates. *Nat Methods*. 2017; **14**: 587-589.

7 Letunic I, Bork P. Interactive Tree Of Life (iTOL) v4: recent updates and new developments. *Nucleic Acids Res*. 2019; **47**: W256-W259.

8 Darriba D, Taboada GL, Doallo R, Posada D. jModelTest 2: more models, new heuristics and parallel computing. *Nat Methods*. 2012; **9**: 772.

9 Anisimova M, Gascuel O. Approximate likelihood-ratio test for branches: A fast, accurate, and powerful alternative. *Syst Biol*. 2006; **55**: 539-552.

:: RNA-Seq and Transcriptomics ::

6.1 Case-Study: *Klebsiella pneumoniae* Transcriptomics - Relationship between the two-component response regulator OmpR and hypermucoviscosity

Introduction

Hypervirulent *Klebsiella pneumoniae* (hvKp) entails a distinct pathotype that contrasts with classical *Klebsiella pneumoniae* (cKp) in its ability and proclivity to cause invasive disease in otherwise healthy individuals [1, 2]. hvKp are often described at multiple sites owing to its metastatic spread and associated with infection acquired in the community. Liver abscess is considered one of its hallmark clinical traits [3].

One of the phenotypic traits associated with hvKp is the hypermucoviscosity phenotype, usually determined by a positive string test upon growth in blood agar [4]. Different genetic determinants have been associated with this specific phenotype, mainly the ability to produce the regulator of mucoid phenotype RmpA [5]. This practical session covers a partial analysis of the transcriptomic data generated by Wang et al (2023) which further elucidates the role of OmpR regulator in the hypermucoviscosity phenotype [6].

Citation:

Wang L, Huang X, Jin Q, Tang J et al. Two-Component Response Regulator OmpR Regulates Mucoviscosity through Energy Metabolism in Klebsiella pneumoniae. *Microbiol Spectr* 2023 Jun 15;11(3):e0054423. PMID: 37097167 http://doi: 10.1128/spectrum.00544-23

The data for this practical is available through the NCBI Gene Expression Omnibus (GEO) database under accession GSE197039. The dataset comprises four samples, subjected to RNA-Seq by paired-end sequencing, representing a *ompR* knockout mutant and the wild-type strain (two biological replicates each, Table 1):

Table 1 – List of runs and biosample accessions and associated genotypes.

Run	BioSample	Genotype
SRR18075271	SAMN26087169	*ompR* knockout
SRR18075272	SAMN26087170	*ompR* knockout
SRR18075273	SAMN26087171	wild-type
SRR18075274	SAMN26087172	wild-type

The practical is divided in three exercises, Exercise 1 pertains the mapping and generation of raw transcript count and has been carried out ahead since the computational steps involved are time consuming. Nonetheless, the commands executed are outlined below along with explanatory context. The second exercise will involve the visualization of mapped reads and the third exercise will cover the identification of differentially expressed genes across the conditions tested.

Exercise 1 – Mapping and Raw read Count

The initial stage of the analytical process of RNA-Seq data for this practical will consist of mapping the reads to a reference genome following a similar approach as the one already covered in the mapping module for this course. As such, this will require an adequate genome reference file. An annotation file covering all genes in this same reference genome will also be needed. We will use the genome of *K. pneumoniae* as reference genome, which has already been downloaded in the FASTA format, GenBank format and annotation (GFF3) from NCBI GenBank (accession CP009208). The following files are therefore already present in the course module directory in the Kp subdirectory. Let's start!

From the home directory type:

```
$ cd Module6/Kp
$ ls
```

Please note for the following files:

- CP009208_1.fasta – reference genome in the fasta format;
- CP009208_1.gb - reference genome in the GenBank format;
- CP009208_1.gff3 – annotation file in the GFF3 format.

You can look inside these files to confirm its format.

In the same directory you will find compressed FastQ files containing the RNA-Seq raw reads. Each is named according to the Run ID in Table 1 and if this is paired-end sequencing, it will have _1.fastq.gz or _2.fastq.gz appended to the Run ID.

Which one do you have? _____

The mapping steps were done using the following commands:

```
$ bwa mem CP009208_1.fasta SRR18075271_1.fastq.gz
SRR18075271_2.fastq.gz | samtools sort --write-index -o
SRR18075271.bam -
$ bwa mem CP009208_1.fasta SRR18075272_1.fastq.gz
SRR18075272_2.fastq.gz | samtools sort --write-index -o
SRR18075272.bam -
$ bwa mem CP009208_1.fasta SRR18075273_1.fastq.gz
SRR18075273_2.fastq.gz | samtools sort --write-index -o
SRR18075273.bam -
$ bwa mem CP009208_1.fasta SRR18075273_1.fastq.gz
SRR18075273_2.fastq.gz | samtools sort --write-index -o
SRR18075273.bam –
```

Each command allows for mapping of the reads using BWA MEM algorithm and pipes it to samtools in order generate a BAM containing all mapped reads and reference. Next you need to index the BAM file by typing:

```
$ samtools index SRR18075271.bam
$ samtools index SRR18075272.bam
$ samtools index SRR18075273.bam
$ samtools index SRR18075274.bam
```


At this point you can already visualize your mapping data but we will leave it to the second part of this practical. For now we need to count the number of reads mapped to each feature/gene. Please note that for this stage you will need to know the position, i.e., start, stop and strand of the genes in your gene. This information is not present in the FASTA reference genome you used to map the reads nor you can extract it directly from the BAM files. This is where the GFF file is needed! You will be using HTSeq to count the reads in the genes, specifically its *htseq-count* script [7]. By typing:

```
$ htseq-count
#or
$ htseq-count --help
```

You will learn a little bit of the different files needed, arguments to this command and output. You can produce the files containing the raw counts by doing the following:

```
$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18075271.bam
CP009208_1.gff3 > SRR18075271_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18075272.bam
CP009208_1.gff3 > SRR18075272_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18075273.bam
CP009208_1.gff3 > SRR18075273_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18075274.bam
CP009208_1.gff3 > SRR18075274_count.txt
```


Can you check why are we using those extra arguments besides the BAM and the GFF file?

Also, please note that the output is being redirected to files named *_count.txt files. Look at their content. Does it resemble something like this:

```
yjdC    985
yjdE    8
yjdH    124
yjeE    24
yjeK    132
yjeR    161
yjfF    16
yjfP    64
yjgD    589
yjgK    18
yjjX    322
ykfE    1821
yliI    508
ymgB    310
ynfA    17
yodA    934
yohD    540
yqaA    27
```

These files contain the raw read counts for each gene and will be used as input for the Differential Gene Expression analysis in Exercise 3. But before that let us visualize the mapped data so that you can understand better what we have done so far and what these counts represent.

Notice that you already have the files with the raw counts! To download all previously created BAM files and index files for this practical session, please go to:

https://ulisboa-my.sharepoint.com/:f:/g/personal/jperdigao_office365_ulisboa_pt/EkBjH_nlesdllqBXPlXy3_oB4NG7dg4RAksXHik2lODmbQ?e=fyu1w7

and download the bam_files_KP.tar.gz file (use command: *tar -xvf bam_files_KP.tar.gz* to decompress the file). Run this last command in the Mtb sub-directory after downloading the file to this location.

Exercise 2 – Visualization of RNA-Seq mapped data

For the next exercise you can visualize the mapped reads in Artemis. Inside the directory containing the data, type:

```
$ art
```

This will start Artemis. You must first load your reference genome, which can be done in multiple ways:

i) you can open the CP009208_1.gb file, containing the genome and features in the GenBank file, or;

ii) you can open the CP009208_1.fasta file and then read in the GFF file so that you can see the different feature tracks.

Next, you need to read in your BAM files. You can start by going to File >> Read BAM/VCF and select the first BAM file from the list. Please remember that to load a BAM file in

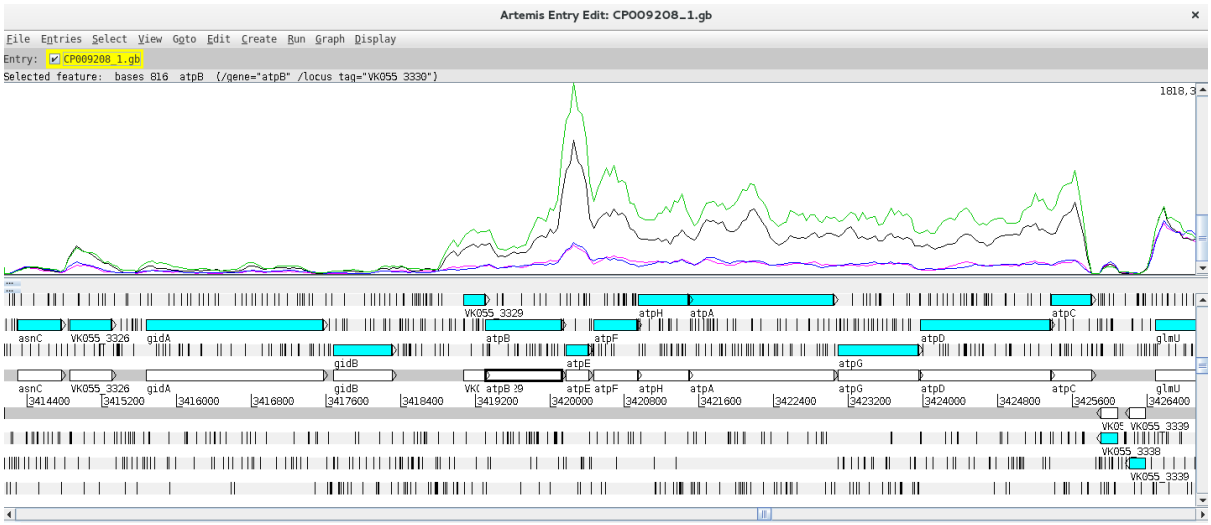
Artemis you need to have the BAM file and the respective .bai index file in the same directory. If everything went ok a BAM window with the mapped reads should appear on top of the genome:

Depending on whether the genome map window is too zoomed in or out you will see the individual reads or a plot. You can change the zoom level in the vertical scroll bar on the right and by right clicking on the BAM window you can also select the type of graph under *View* (try changing between pileup and coverage for example).

Let's look at a particular gene, for example *mhpA*, by using the Navigator (*Goto* >> *Navigator*) and typing the gene name in the appropriate search box.

Notice that in the vicinity of these genes, there are some with no/low coverage while others have coverage and there are even reads mapping to intergenic regions. Why is that?

Next, you can add and compare the coverage from other BAM files. To do this just right click on the BAM window, select "*Add BAM*" and add the different BAM files. Also, changing the Graph type to coverage will facilitate the interpretation of the Data. By right clicking on the BAM window you also have the option to configure these lines (changing the colour, thickness, etc.).



Can you find any differentially expressed genes (DEG) using this approach?

Hint: Try the *atp* operon region!

Exercise 3 – Examining Differential Expression

This last exercise is aimed at identifying differentially expressed genes (DEGs) by taking the raw counts as input to software that normalizes the number of mapped reads and apply statistical tests to identify DEGs. The most widely used packages for this purpose are the R packages DESeq2 and EdgeR. In this practical we will use DESeq2 to identify DEGs, but code DEG identification using EdgeR is also provided and you can even compare the results between packages [8, 9].

It is also important to stress that using different replicates is a fundamental aspect of DEG analysis as expression levels are variable and depend on several external factors. Biological replicates are therefore essential to account for the variability of biological systems and more accurately compare expression levels between groups/conditions.

The exercise will be carried out in R which involves a different programming language. Let's go step by step.

Let's start R by typing:

```
$ R
```

Inside the R command line we need to load the required libraries for this part:

```
> library(DESeq2)
> library(gplots)
```

Next we will list the files with counts into an object called *sampleFiles* and create an object with the condition/genotype. It is important that the order of the genotypes is the same as the order of the listed files:

```
> sampleFiles<-list.files(pattern="count.txt$")
#Use the same order from the sampleFiles object:
> sampleCondition <- c("ompR_KO","ompR_KO","wt","wt")
```

We can then create a data frame, which is a type of R object with a table structure called *sampleTable*:

```
> sampleTable <- data.frame(
  sampleName = gsub("_count.txt","",sampleFiles),
  fileName = sampleFiles,
  condition = sampleCondition
)
```

Notice that you defined the columns in this data frame and for sample names you run the object containing the file list through a substitution command to remove the suffix of the file names.

You can check the structure of the data frame just by typing:

```
> sampleTable
```

Please check if everything as we will next move to the analytical stage. You can import the HTSeq counts into a DESeqDataSet object and look at the contents of this object by running the following:

```
> dds <- DESeqDataSetFromHTSeqCount(
  sampleTable = sampleTable,
  design = ~ condition
)

> dds
```

Pay attention to the design argument of the first function, you need to specify the grouping factor. This argument can accommodate multiple designs, this is the simplest design scheme in which you specified the name of a single column from your original dataframe.

A pre-filtering step can be carried out to reduce the number of genes with a count below 10. This step is not essential but it can help to reduce the size of the DESeqDataSet object. To do this, run the following:

```
> keep <- rowSums(counts(dds)) >= 10
> dds <- dds[keep,]
```

Next, setting the factor levels is important to define the reference level (control group). If this is not specified the levels will be taken in alphabetical order and the reference level will be assumed to be the first one. Below there are two options to set the levels:

```
> dds$condition <- factor(dds$condition, levels = c("wt","ompR_KO"))
# Notice that wt comes first as this is the natural reference group (control)
# or
> dds$condition <- relevel(dds$condition,ref="wt")
```

The differential expression analysis is carried out using the *DESeq* function that takes the DESeqDataSet object as input, does the normalization of the counts, estimates dispersion and additional statistical tests. After, the result table can be generated using the results function *results* :

```
> dds <- DESeq(dds)
> res <- results(dds, contrast = c("condition","ompR_KO","wt"))
```

In the second command we have just used the contrast argument to the results function which might not be necessary but enables you to specifically control the comparison being made across groups. To look at the results table just type:

```
> res
```

What do you see? Something like this:

```
log2 fold change (MLE): condition ompR_KO vs wt
Wald test p-value: condition ompR_KO vs wt
DataFrame with 4069 rows and 6 columns
            baseMean log2FoldChange     lfcSE       stat      pvalue
           <numeric>      <numeric> <numeric>  <numeric>   <numeric>
VK055_0008   14.3907      0.0332812  0.730853  0.0455375   0.9636789
VK055_0010   21.9589     -0.7356173  0.594794 -1.2367594   0.2161764
VK055_0011   17.0704     -0.8991868  0.657200 -1.3682097   0.1712464
VK055_0012   70.2776     -0.9149246  0.329761 -2.7745097   0.0055285
VK055_0015  354.6755     -0.0134678  0.175108 -0.0769112   0.9386942
...              ...            ...       ...        ...         ...
zntR         68.7449      -0.248428  0.329993   -0.75283 4.51552e-01
znuA       1430.5374      -0.672363  0.125105   -5.37440 7.68373e-08
zupT        298.8757      -0.620975  0.183087   -3.39169 6.94641e-04
zur         741.3930      -0.153331  0.131868   -1.16276 2.44928e-01
zwf         627.0477      -0.254292  0.139242   -1.82626 6.78118e-02
                 padj
            <numeric>
VK055_0008  0.9761537
VK055_0010  0.3720904
VK055_0011  0.3154032
VK055_0012  0.0199075
VK055_0015  0.9604458
...               ...
zntR       6.14540e-01
znuA       7.87534e-07
zupT       3.23398e-03
zur        4.05787e-01
zwf        1.55715e-01
```

You can summarize the results by running:

```
> summary(res)
```

```
out of 4605 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 745, 16%
LFC < 0 (down)     : 839, 18%
outliers [1]       : 0, 0%
low counts [2]     : 536, 12%
(mean count < 7)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

You should see the standard adjusted p-value threshold used for the summary and the number of genes with a Log_2 Fold Change (LFC) above and below 0.

Next let's remove results with no adjusted p-values and sort the table by this value:

```
> res <- res[!is.na(res$padj),]
> resOrdered <- res[order(res$pvalue),]
> resOrdered
```


You can construct a heatmap for the top 50 genes using the following commands:

```
> counts_heatmap <- counts(dds, normalized = TRUE)
> idx <- rownames(resOrdered)[1:100]
> counts_heatmap <- counts_heatmap[rownames(counts_heatmap)%in%idx,]


> counts_heatmap


>                    colnames(counts_heatmap)                    <-
c("ompR_KO_2","ompR_KO_2","wt_1","wt_2")
>heatmap.2(as.matrix(counts_heatmap), scale="row", col=greenred(75),
Rowv=NA, dendrogram = "col", trace="none", density.info = "none")
```

Another option is to visualize Gene Plot Counts for multiple genes:

```
> par(mfrow=c(2,3))
> plotCounts(dds,gene="rpsJ", intgroup="condition")
> plotCounts(dds,gene="VK055_3129", intgroup="condition")
> plotCounts(dds,gene="fyuA", intgroup="condition")
> plotCounts(dds,gene="cobO", intgroup="condition")
> plotCounts(dds,gene="VK055_2318", intgroup="condition")
> plotCounts(dds,gene="VK055_2733", intgroup="condition")
> par(mfrow=c(1,1))
```


You can also do a Principal Component Analysis of the results:

```
> vsdata<-vst(dds,blind=FALSE)
> z <- plotPCA(vsdata,intgroup="condition")
> z + coord_fixed(ylim=c(-10,10), xlim=c(-15,15))
```

Do the global transcriptomic signatures from different conditions cluster in this analysis?

For a Volcano Plot, you can construct a basic one:

```
> with(res, plot(log2FoldChange, -log10(pvalue),
pch=20,main="Volcano plot",xlim=c(-8,8)))
```

or highlight in blue if the adjusted p-value is below 0.01 or red if the adjusted p-value is below 0.01 and the Log2 Fold Change is higher than 1:

```
> with(subset(res,padj<.01),points(log2FoldChange, -
log10(pvalue),pch=20,col="blue"))

> with(subset(res,padj<.01 &
abs(log2FoldChange)>2),points(log2FoldChange, -
log10(pvalue),pch=20,col="red"))
```

Moreover, you can write you results table to a csv file, or create individual files of upregulated or downregulated genes based on specific thresholds that you can decide on. For example:


```
> write.table(resOrdered,file="resOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)


##The following produce different files based on criteria adjusted
p-value < 0.05 and Log2 Fold Change > 0.5, sorted by the adjusted p-
value:

> res_sig<-subset(res,padj<0.05)

> res_sig_up<-subset(res_sig, log2FoldChange > 0.5)

> res_sig_upOrdered<-res_sig_up[order(res_sig_up$padj),]

> write.table(res_sig_upOrdered, file="res_sig_upOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)




> res_sig_down<-subset(res_sig, log2FoldChange < 0.5)

> res_sig_downOrdered<-res_sig_down[order(res_sig_down$padj),]

> write.table(res_sig_downOrdered, file="res_sig_downOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)
```

You can then open the CSV files in MS Excel or LibreOffice Calc and examine the results. Please note that you must adjust the column headers.

References

1 Wyres KL, Lam MMC, Holt KE. Population genomics of Klebsiella pneumoniae. *Nat Rev Microbiol*. 2020; **18**: 344-359.

2 Spadar A, Perdigao J, Campino S, Clark TG. Genomic analysis of hypervirulent Klebsiella pneumoniae reveals potential genetic markers for differentiation from classical strains. *Sci Rep*. 2022; **12**: 13671.

3 Siu LK, Yeh KM, Lin JC, Fung CP, Chang FY. Klebsiella pneumoniae liver abscess: a new invasive syndrome. *Lancet Infect Dis*. 2012; **12**: 881-887.

4 Walker KA, Miller VL. The intersection of capsule gene expression, hypermucoviscosity and hypervirulence in Klebsiella pneumoniae. *Curr Opin Microbiol*. 2020; **54**: 95-102.

5 Nadasy KA, Domiati-Saad R, Tribble MA. Invasive Klebsiella pneumoniae syndrome in North America. *Clin Infect Dis*. 2007; **45**: e25-28.

6 Wang L, Huang X, Jin Q, et al. Two-Component Response Regulator OmpR Regulates Mucoviscosity through Energy Metabolism in Klebsiella pneumoniae. *Microbiol Spectr*. 2023; **11**: e0054423.

7 Anders S, Pyl PT, Huber W. HTSeq--a Python framework to work with high-throughput sequencing data. *Bioinformatics*. 2015; **31**: 166-169.

8 Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol*. 2014; **15**: 550.

9 Robinson MD, McCarthy DJ, Smyth GK. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*. 2010; **26**: 139-140.

6.2 Case-Study: *Pseudomonas aeruginosa* Transcriptomics – Pf4 phage variant induced dysregulation

Introduction

Pseudomonas aeruginosa is a Gram-negative opportunistic pathogen that is commonly present in diverse environments such as water, soil or plants [1]. This pathogen is one of the most frequent agents of healthcare-associated infections and frequently associated with multidrug resistance [2, 3]. Moreover, this species has intrinsic resistance to several antibiotics and also displays numerous virulence factors [4]. Previous data obtained by Tortuel *et al* (2022) showed that infection of *P. aeruginosa* with a Pf4 variant bacteriophage displayed altered expression of genes involved in stress response [5]. This practical covers the partial analysis of the global transcriptomic study developed by the authors using *P. aeruginosa* in untreated conditions and those treated with the Pf4 variant bacteriophage.

Citation:

Tortuel D, Tahrioui A, David A, Cambronel M et al. Pf4 Phage Variant Infection Reduces Virulence-Associated Traits in Pseudomonas aeruginosa. *Microbiol Spectr* 2022 Oct 26;10(5):e0154822. PMID: 36036571

The data for this practical is available through the NCBI Gene Expression Omnibus (GEO) database under accession GSE201738. The dataset comprises six samples, subjected to RNA-Seq by paired-end sequencing, representing a Pf4-infected and Pf4-uninfected strains (three biological replicates each, Table 1):

Table 1 – List of runs and biosample accessions and associated genotypes.

Run	BioSample	Treatment
SRR18957713	SAMN27921894	Pf4 phage treatment
SRR18957714	SAMN27921896	Pf4 phage treatment
SRR18957715	SAMN27921897	No treatment
SRR18957716	SAMN27921895	Pf4 phage treatment

SRR18957717	SAMN27921898	No treatment
SRR18957718	SAMN27921899	No treatment

The practical is divided in three exercises, Exercise 1 pertains the mapping and generation of raw transcript count and has been carried out ahead since the computational steps involved are time consuming. Nonetheless, the commands executed are outlined below along with explanatory context. The second exercise will involve the visualization of mapped reads and the third exercise will cover the identification of differentially expressed genes across the conditions tested.

Exercise 1 – Mapping and Raw read Count

The initial stage of the analytical process of RNA-Seq data for this practical will consist of mapping the reads to a reference genome following a similar approach as the one already covered in the mapping module for this course. As such, this will require an adequate genome reference file. An annotation file covering all genes in this same reference genome will also be needed. We will use the genome of *Pseudomonas aeruginosa* PAO1 as reference genome, which has already been downloaded in the FASTA format, GenBank format and annotation (GFF3) from NCBI GenBank (accession NC002516). The following files are therefore already present in the course module directory in the Pa subdirectory. Let's start!

From the home directory type:

```
$ cd Module6/Pa
$ ls
```

Please note for the following files:

- NC002516_2.fasta – reference genome in the fasta format;
- NC002516_2.gb - reference genome in the GenBank format;
- NC002516_2.gff3 – annotation file in the GFF3 format.

You can look inside these files to confirm its format.

In the same directory you will find compressed FastQ files containing the RNA-Seq raw reads. Each is named according to the Run ID in Table 1 and if this is paired-end sequencing, it will have _1.fastq.gz or _2.fastq.gz appended to the Run ID.

Which one do you have? _____

The mapping steps were done using the following commands:

```
$ bwa mem NC002516_2.fasta SRR18957713_1.fastq.gz
SRR18957713_2.fastq.gz | samtools sort --write-index -o
SRR18957713.bam -

$ bwa mem NC002516_2.fasta SRR18957714_1.fastq.gz
SRR18957714_2.fastq.gz | samtools sort --write-index -o
SRR18957714.bam -

$ bwa mem NC002516_2.fasta SRR18957715_1.fastq.gz
SRR18957715_2.fastq.gz | samtools sort --write-index -o
SRR18957715.bam -

$ bwa mem NC002516_2.fasta SRR18957716_1.fastq.gz
SRR18957716_2.fastq.gz | samtools sort --write-index -o
SRR18957716.bam -

$ bwa mem NC002516_2.fasta SRR18957717_1.fastq.gz
SRR18957717_2.fastq.gz | samtools sort --write-index -o
SRR18957717.bam -

$ bwa mem NC002516_2.fasta SRR18957717_1.fastq.gz
SRR18957717_2.fastq.gz | samtools sort --write-index -o
SRR18957717.bam -

$ bwa mem NC002516_2.fasta SRR18957718_1.fastq.gz
SRR18957718_2.fastq.gz | samtools sort --write-index -o
SRR18957718.bam -
```

Each command allows for mapping of the reads using BWA MEM algorithm and pipes it to samtools in order generate a BAM containing all mapped reads and reference. Next you need to index the BAM file by typing:

```
$ samtools index SRR18957713.bam
$ samtools index SRR18957714.bam
$ samtools index SRR18957715.bam
$ samtools index SRR18957716.bam
$ samtools index SRR18957717.bam
$ samtools index SRR18957718.bam
```


At this point you can already visualize your mapping data but we will leave it to the second part of this practical. For now we need to count the number of reads mapped to each feature/gene. Please note that for this stage you will need to know the position, i.e., start, stop and strand of the genes in your gene. This information is not present in the FASTA reference genome you used to map the reads nor you can extract it directly from the BAM files. This is where the GFF file is needed! You will be using HTSeq to count the reads in the genes, specifically its *htseq-count* script [6]. By typing:

```
$ htseq-count
#or
$ htseq-count --help
```

You will learn a little bit of the different files needed, arguments to this command and output. You can produce the files containing the raw counts by doing the following:

```
$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18957713.bam
NC002516_2.gff3 > SRR18957713_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18957714.bam
NC002516_2.gff3 > SRR18957714_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18957715.bam
NC002516_2.gff3 > SRR18957715_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18957716.bam
NC002516_2.gff3 > SRR18957716_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18957717.bam
NC002516_2.gff3 > SRR18957717_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18957718.bam
NC002516_2.gff3 > SRR18957718_count.txt
```


Can you check why are we using those extra arguments besides the BAM and the GFF file?

Also, please note that the output is being redirected to files named *_count.txt files. Look at their content. Does it resemble something like this:

```
yjdC    985
yjdE    8
yjdH    124
yjeE    24
yjeK    132
yjeR    161
yjfF    16
yjfP    64
yjgD    589
yjgK    18
yjjX    322
ykfE    1821
yliI    508
ymgB    310
ynfA    17
yodA    934
yohD    540
yqaA    27
```

These files contain the raw read counts for each gene and will be used as input for the Differential Gene Expression analysis in Exercise 3. But before that let us visualize the

mapped data so that you can understand better what we have done so far and what these counts represent.

Notice that you already have the files with the raw counts! To download all previously created BAM files and index files for this practical session, please go to:

https://ulisboa-my.sharepoint.com/:f:/g/personal/jperdigao_office365_ulisboa_pt/EkBjH_nlesdllqBXPlXy3_oB4NG7dg4RAksXHik2lODmbQ?e=fyu1w7

and download the bam_files_PA.tar.gz file (use command: *tar -xvf bam_files_PA.tar.gz* to decompress the file). Run this last command in the Pa sub-directory after downloading the file to this location.

Exercise 2 – Visualization of RNA-Seq mapped data

For the next exercise you can visualize the mapped reads in Artemis. Inside the directory containing the data, type:

```
$ art
```

This will start Artemis. You must first load your reference genome, which can be done in multiple ways:

i) you can open the NC002516_2.gb file, containing the genome and features in the GenBank file, or;

ii) you can open the NC002516_2.fasta file and then read in the GFF file so that you can see the different feature tracks.

Next, you need to read in your BAM files. You can start by going to File >> Read BAM/VCF and select the first BAM file from the list. Please remember that to load a BAM file in Artemis you need to have the BAM file and the respective .bai index file in the same

directory. If everything went ok a BAM window with the mapped reads should appear on top of the genome:

Depending on whether the genome map window is too zoomed in or out you will see the individual reads or a plot. You can change the zoom level in the vertical scroll bar on the right and by right clicking on the BAM window you can also select the type of graph under *View* (try changing between pileup and coverage for example).

Let's look at a particular gene, for example *lepB*, by using the Navigator (*Goto* >> *Navigator*) and typing the gene name in the appropriate search box.

Notice that in the vicinity of these genes, there are some with no/low coverage while others have coverage and there are even reads mapping to intergenic regions. Why is that?

Next, you can add and compare the coverage from other BAM files. To do this just right click on the BAM window, select "*Add BAM*" and add the different BAM files. Also, changing the Graph type to coverage will facilitate the interpretation of the Data. By right clicking on the BAM window you also have the option to configure these lines (changing the colour, thickness, etc.).

Can you find any differentially expressed genes (DEG) using this approach?

Hint: Try the *aprA* gene!

Exercise 3 – Examining Differential Expression

This last exercise is aimed at identifying differentially expressed genes (DEGs) by taking the raw counts as input to software that normalizes the number of mapped reads and apply statistical tests to identify DEGs. The most widely used packages for this purpose are the R packages DESeq2 and EdgeR. In this practical we will use DESeq2 to identify DEGs, but code DEG identification using EdgeR is also provided and you can even compare the results between packages [7, 8].

It is also important to stress that using different replicates is a fundamental aspect of DEG analysis as expression levels are variable and depend on several external factors. Biological replicates are therefore essential to account for the variability of biological systems and more accurately compare expression levels between groups/conditions.

The exercise will be carried out in R which involves a different programming language. Let's go step by step.

Let's start R by typing:

```
$ R
```

Inside the R command line we need to load the required libraries for this part:

```
> library(DESeq2)
> library(gplots)
```

Next, we will list the files with counts into an object called *sampleFiles* and create an object with the condition/genotype. It is important that the order of the genotypes is the same as the order of the listed files:

```
> sampleFiles<-list.files(pattern="count.txt$")
#Use the same order from the sampleFiles object:
> sampleCondition <- c("Pf4", "Pf4", "uninfected", "Pf4",
"uninfected", "uninfected")
```

We can then create a data frame, which is a type of R object with a table structure called *sampleTable*:

```
> sampleTable <- data.frame(
  sampleName = gsub("_count.txt","",sampleFiles),
  fileName = sampleFiles,
  condition = sampleCondition
)
```

Notice that you defined the columns in this data frame and for sample names you run the object containing the file list through a substitution command to remove the suffix of the file names.

You can check the structure of the data frame just by typing:

```
> sampleTable
```

Please check if everything as we will next move to the analytical stage. You can import the HTSeq counts into a DESeqDataSet object and look at the contents of this object by running the following:

```
> dds <- DESeqDataSetFromHTSeqCount(
  sampleTable = sampleTable,
  design = ~ condition
)

> dds
```

Pay attention to the design argument of the first function, you need to specify the grouping factor. This argument can accommodate multiple designs, this is the simplest

design scheme in which you specified the name of a single column from your original dataframe.

A pre-filtering step can be carried out to reduce the number of genes with a count below 10. This step is not essential but it can help to reduce the size of the DESeqDataSet object. To do this, run the following:

```
> keep <- rowSums(counts(dds)) >= 10
> dds <- dds[keep,]
```

Next, setting the factor levels is important to define the reference level (control group). If this is not specified the levels will be taken in alphabetical order and the reference level will be assumed to be the first one. Below there are two options to set the levels:

```
> dds$condition <- factor(dds$condition, levels = c("uninfected", "Pf4"))
# Notice that wt comes first as this is the natural reference group (control)
# or
> dds$condition <- relevel(dds$condition,ref="uninfected")
```

The differential expression analysis is carried out using the *DESeq* function that takes the DESeqDataSet object as input, does the normalization of the counts, estimates dispersion and additional statistical tests. After, the result table can be generated using the results function *results* :

```
> dds <- DESeq(dds)
> res <- results(dds, contrast = c("condition","Pf4","uninfected"))
```

In the second command we have just used the contrast argument to the results function which might not be necessary but enables you to specifically control the comparison being made across groups. To look at the results table just type:

```
> res
```


What do you see? Something like this:

```
log2 fold change (MLE): condition ompR_KO vs wt
Wald test p-value: condition ompR_KO vs wt
DataFrame with 4069 rows and 6 columns
           baseMean log2FoldChange     lfcSE       stat      pvalue
          <numeric>      <numeric> <numeric>  <numeric>   <numeric>
VK055_0008   14.3907      0.0332812  0.730853  0.0455375   0.9636789
VK055_0010   21.9589     -0.7356173  0.594794 -1.2367594   0.2161764
VK055_0011   17.0704     -0.8991868  0.657200 -1.3682097   0.1712464
VK055_0012   70.2776     -0.9149246  0.329761 -2.7745097   0.0055285
VK055_0015  354.6755     -0.0134678  0.175108 -0.0769112   0.9386942
...              ...            ...       ...        ...         ...
zntR         68.7449      -0.248428  0.329993   -0.75283 4.51552e-01
znuA       1430.5374      -0.672363  0.125105   -5.37440 7.68373e-08
zupT        298.8757      -0.620975  0.183087   -3.39169 6.94641e-04
zur         741.3930      -0.153331  0.131868   -1.16276 2.44928e-01
zwf         627.0477      -0.254292  0.139242   -1.82626 6.78118e-02
                 padj
            <numeric>
VK055_0008  0.9761537
VK055_0010  0.3720904
VK055_0011  0.3154032
VK055_0012  0.0199075
VK055_0015  0.9604458
...               ...
zntR       6.14540e-01
znuA       7.87534e-07
zupT       3.23398e-03
zur        4.05787e-01
zwf        1.55715e-01
```

You can summarize the results by running:

```
> summary(res)
```

```
out of 4605 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 745, 16%
LFC < 0 (down)     : 839, 18%
outliers [1]       : 0, 0%
low counts [2]     : 536, 12%
(mean count < 7)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

You should see the standard adjusted p-value threshold used for the summary and the number of genes with a Log_2 Fold Change (LFC) above and below 0.

Next let's remove results with no adjusted p-values and sort the table by this value:

```
> res <- res[!is.na(res$padj),]
> resOrdered <- res[order(res$pvalue),]
> resOrdered
```

You can construct a heatmap for the top 50 genes using the following commands:

```
> counts_heatmap <- counts(dds, normalized = TRUE)

> idx <- rownames(resOrdered)[1:100]

>  counts_heatmap  <-  counts_heatmap[rownames(counts_heatmap)  %in% idx,]


> counts_heatmap


> colnames(counts_heatmap) <- c("Pf4_1", "Pf4_2", "uninfected_1", "Pf4_3", "uninfected_2", "uninfected_3")

>heatmap.2(as.matrix(counts_heatmap), scale="row", col=greenred(75), Rowv=NA, dendrogram = "col", trace="none", density.info = "none")
```


Another option is to visualize Gene Plot Counts for multiple genes:

```
> par(mfrow=c(2,3))

> plotCounts(dds,gene="PA0048", intgroup="condition")

> plotCounts(dds,gene="PA0049", intgroup="condition")

> plotCounts(dds,gene="PA0122", intgroup="condition")

> plotCounts(dds,gene="PA5160.1", intgroup="condition")

> plotCounts(dds,gene="nosY", intgroup="condition")

> plotCounts(dds,gene="PA0100", intgroup="condition")

> par(mfrow=c(1,1))
```

You can also do a Principal Component Analysis of the results:

```
> vsdata<-vst(dds,blind=FALSE)

> z <- plotPCA(vsdata,intgroup="condition")

> z + coord_fixed(ylim=c(-40,40), xlim=c(-40,40))
```

Do the global transcriptomic signatures from different conditions cluster in this analysis?

For a Volcano Plot, you can construct a basic one:

```
> with(res, plot(log2FoldChange, -log10(pvalue),
pch=20,main="Volcano plot",xlim=c(-8,8)))
```

or highlight in blue if the adjusted p-value is below 0.01 or red if the adjusted p-value is below 0.01 and the Log2 Fold Change is higher than 1:

```
> with(subset(res,padj<.01),points(log2FoldChange, -
log10(pvalue),pch=20,col="blue"))

> with(subset(res,padj<.01 &
abs(log2FoldChange)>2),points(log2FoldChange, -
log10(pvalue),pch=20,col="red"))
```

Moreover, you can write you results table to a csv file, or create individual files of upregulated or downregulated genes based on specific thresholds that you can decide on. For example:

```
> write.table(resOrdered,file="resOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)


##The following produce different files based on criteria adjusted
p-value < 0.05 and Log2 Fold Change > 0.5, sorted by the adjusted p-
value:

> res_sig<-subset(res,padj<0.05)

> res_sig_up<-subset(res_sig, log2FoldChange > 0.5)

> res_sig_upOrdered<-res_sig_up[order(res_sig_up$padj),]

> write.table(res_sig_upOrdered, file="res_sig_upOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)



> res_sig_down<-subset(res_sig, log2FoldChange < 0.5)

> res_sig_downOrdered<-res_sig_down[order(res_sig_down$padj),]

> write.table(res_sig_downOrdered, file="res_sig_downOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)
```

You can then open the CSV files in MS Excel or LibreOffice Calc and examine the results. Please note that you must adjust the column headers.

References

1 Moradali MF, Ghods S, Rehm BH. Pseudomonas aeruginosa Lifestyle: A Paradigm for Adaptation, Survival, and Persistence. *Front Cell Infect Microbiol*. 2017; **7**: 39.

2 Tenover FC, Nicolau DP, Gill CM. Carbapenemase-producing Pseudomonas aeruginosa -an emerging challenge. *Emerg Microbes Infect*. 2022; **11**: 811-814.

3 Tomczyk S, Zanichelli V, Grayson ML, et al. Control of Carbapenem-resistant Enterobacteriaceae, Acinetobacter baumannii, and Pseudomonas aeruginosa in Healthcare Facilities: A Systematic Review and Reanalysis of Quasi-experimental Studies. *Clin Infect Dis*. 2019; **68**: 873-884.

4 Azam MW, Khan AU. Updates on the pathogenicity status of Pseudomonas aeruginosa. *Drug Discov Today*. 2019; **24**: 350-359.

5 Tortuel D, Tahrioui A, David A, et al. Pf4 Phage Variant Infection Reduces Virulence-Associated Traits in Pseudomonas aeruginosa. *Microbiol Spectr*. 2022; **10**: e0154822.

6 Anders S, Pyl PT, Huber W. HTSeq--a Python framework to work with high-throughput sequencing data. *Bioinformatics*. 2015; **31**: 166-169.

7 Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol*. 2014; **15**: 550.

8 Robinson MD, McCarthy DJ, Smyth GK. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*. 2010; **26**: 139-140.

6.3 Case-Study: *Mycobacterium tuberculosis* Transcriptomics – examining the regulome of sigma factor A and B

Introduction

Mycobacterium tuberculosis is the etiological agent of tuberculosis, and probably the most successful pathogen in human history [1]. The pathogen's natural cycle of infection entails a complex interaction with the human host and distinct metabolic stages [1, 2]. The capacity of metabolic adaptation is intertwined with the pathogen's capability to also modulate the expression of several genes, a process that can be driven by the regulation and expression of different sigma factors [3, 4]. This practical session covers a partial analysis of the work reported by Singha *et al* (2023) which used different *M. tuberculosis* H37Rv mutant strains to study the regulatory effect exerted by sigma factor A (SigA) and B (SigB) 36690275. For this purpose the authors constructed knock-out mutants of both genes, including a conditional mutant of *sigA*, by cloning this gene under a anhydro-tetracycline (ATc)-inducible promoter in a *sigA* knock-out genetic background.

Citation:

Singha B, Behera D, Khan MZ, Singh NK et al. The unique N-terminal region of Mycobacterium tuberculosis sigma factor A plays a dominant role in the essential function of this protein. *J Biol Chem* 2023 Mar;299(3):102933. PMID: 36690275 https://doi.org/10.1016/j.jbc.2023.102933

The data for this practical is available through the NCBI Gene Expression Omnibus (GEO) database under accession GSE197742. The dataset comprises 10 samples, subjected to RNA-Seq by paired-end sequencing, representing a wild-type control strain (H37Rv), a sigB knockout mutant, and conditional sigA knockout mutant under induced (+ATc) and non-induced conditions (-ATc) Pf4-uninfected strains (2-3 biological replicates each, Table 1):

Table 1 – List of runs and biosample accessions and associated genotypes.

Run	BioSample	Genotype/Treatment
SRR18190582	SAMN26363810	Wild-type, control
SRR18190583	SAMN26363811	Wild-type, control
SRR18190584	SAMN26363812	*sigB*KO
SRR18190585	SAMN26363813	*sigB*KO
SRR18190586	SAMN26363814	*sigA*KO+ATc
SRR18190587	SAMN26363815	*sigA*KO+ATc
SRR18190588	SAMN26363816	*sigA*KO+ATc
SRR18190589	SAMN26363817	*sigA*KO-ATc
SRR18190590	SAMN26363818	*sigA*KO-ATc
SRR18190591	SAMN26363819	*sigA*KO-ATc

The practical is divided in three exercises, Exercise 1 pertains the mapping and generation of raw transcript count and has been carried out ahead since the computational steps involved are time consuming. Nonetheless, the commands executed are outlined below along with explanatory context. The second exercise will involve the visualization of mapped reads and the third exercise will cover the identification of differentially expressed genes across the conditions tested.

Exercise 1 – Mapping and Raw read Count

The initial stage of the analytical process of RNA-Seq data for this practical will consist of mapping the reads to a reference genome following a similar approach as the one already covered in the mapping module for this course. As such, this will require an adequate genome reference file. An annotation file covering all genes in this same reference genome will also be needed. We will use the genome of *Pseudomonas aeruginosa* PAO1 as reference genome, which has already been downloaded in the FASTA format, GenBank format and annotation (GFF3) from NCBI GenBank (accession

NC002516). The following files are therefore already present in the course module directory in the Mtb subdirectory. Let's start!

From the home directory type:

```
$ cd Module6/Mtb
$ ls
```

Please note for the following files:

- NC000962_3.fasta – reference genome in the fasta format;
- NC000962_3.gb - reference genome in the GenBank format;
- NC000962_3.gff3 – annotation file in the GFF3 format.

You can look inside these files to confirm its format.

In the same directory you will find compressed FastQ files containing the RNA-Seq raw reads. Each is named according to the Run ID in Table 1 and if this is paired-end sequencing, it will have _1.fastq.gz or _2.fastq.gz appended to the Run ID.

Which one do you have? _____

The mapping steps were done using the following commands:

```
$ bwa mem NC000962_3.fasta SRR18190582_1.fastq.gz
SRR18190582_2.fastq.gz | samtools sort --write-index -o
SRR18190582.bam -

$ bwa mem NC000962_3.fasta SRR18190583_1.fastq.gz
SRR18190583_2.fastq.gz | samtools sort --write-index -o
SRR18190583.bam -

$ bwa mem NC000962_3.fasta SRR18190584_1.fastq.gz
SRR18190584_2.fastq.gz | samtools sort --write-index -o
SRR18190584.bam -

$ bwa mem NC000962_3.fasta SRR18190585_1.fastq.gz
SRR18190585_2.fastq.gz | samtools sort --write-index -o
SRR18190585.bam -

$ bwa mem NC000962_3.fasta SRR18190586_1.fastq.gz
SRR18190586_2.fastq.gz | samtools sort --write-index -o
SRR18190586.bam -

$ bwa mem NC000962_3.fasta SRR18190587_1.fastq.gz
SRR18190587_2.fastq.gz | samtools sort --write-index -o
SRR18190587.bam -

$ bwa mem NC000962_3.fasta SRR18190588_1.fastq.gz
SRR18190588_2.fastq.gz | samtools sort --write-index -o
SRR18190588.bam -

$ bwa mem NC000962_3.fasta SRR18190589_1.fastq.gz
SRR18190589_2.fastq.gz | samtools sort --write-index -o
SRR18190589.bam -

$ bwa mem NC000962_3.fasta SRR18190590_1.fastq.gz
SRR18190590_2.fastq.gz | samtools sort --write-index -o
SRR18190590.bam -

$ bwa mem NC000962_3.fasta SRR18190591_1.fastq.gz
SRR18190591_2.fastq.gz | samtools sort --write-index -o
SRR18190591.bam -
```

Each command allows for mapping of the reads using BWA MEM algorithm and pipes it to samtools in order generate a BAM containing all mapped reads and reference. Next you need to index the BAM file by typing:

```
$ samtools index SRR18190582.bam
$ samtools index SRR18190583.bam
$ samtools index SRR18190584.bam
$ samtools index SRR18190585.bam
$ samtools index SRR18190586.bam
$ samtools index SRR18190587.bam
```


At this point you can already visualize your mapping data but we will leave it to the second part of this practical. For now we need to count the number of reads mapped to each feature/gene. Please note that for this stage you will need to know the position, i.e., start, stop and strand of the genes in your gene. This information is not present in the FASTA reference genome you used to map the reads nor you can extract it directly from the BAM files. This is where the GFF file is needed! You will be using HTSeq to count the reads in the genes, specifically its *htseq-count* script [5]. By typing:

```
$ htseq-count
#or
$ htseq-count --help
```

You will learn a little bit of the different files needed, arguments to this command and output. You can produce the files containing the raw counts by doing the following:

```
$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190582.bam
NC000962_3.gff3 > SRR18190582_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190583.bam
NC000962_3.gff3 > SRR18190583_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190584.bam
NC000962_3.gff3 > SRR18190584_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190585.bam
NC000962_3.gff3 > SRR18190585_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190586.bam
NC000962_3.gff3 > SRR18190586_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190587.bam
NC000962_3.gff3 > SRR18190587_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190588.bam
NC000962_3.gff3 > SRR18190588_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190589.bam
NC000962_3.gff3 > SRR18190589_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190590.bam
NC000962_3.gff3 > SRR18190590_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR18190591.bam
NC000962_3.gff3 > SRR18190591_count.txt
```

Can you check why are we using those extra arguments besides the BAM and the GFF file?

Also, please note that the output is being redirected to files named *_count.txt files. Look at their content. Does it resemble something like this:

```
yjdC    985
yjdE    8
yjdH    124
yjeE    24
yjeK    132
yjeR    161
yjfF    16
yjfP    64
yjgD    589
yjgK    18
yjjX    322
ykfE    1821
yliI    508
ymgB    310
ynfA    17
yodA    934
yohD    540
yqaA    27
```


These files contain the raw read counts for each gene and will be used as input for the Differential Gene Expression analysis in Exercise 3. But before that let us visualize the mapped data so that you can understand better what we have done so far and what these counts represent.

Notice that you already have the files with the raw counts! To download all previously created BAM files and index files for this practical session, please go to:

https://ulisboa-my.sharepoint.com/:f:/g/personal/jperdigao_office365_ulisboa_pt/EkBjH_nIesdIlqBXPIXy3_oB4NG7dg4RAksXHik2lODmbQ?e=fyu1w7

and download the bam_files_MTB.tar.gz file (use command: *tar -xvf bam_files_MTB.tar.gz* to decompress the file). Run this last command in the Mtb sub-directory after downloading the file to this location.

Exercise 2 – Visualization of RNA-Seq mapped data

For the next exercise you can visualize the mapped reads in Artemis. Inside the directory containing the data, type:

```
$ art
```

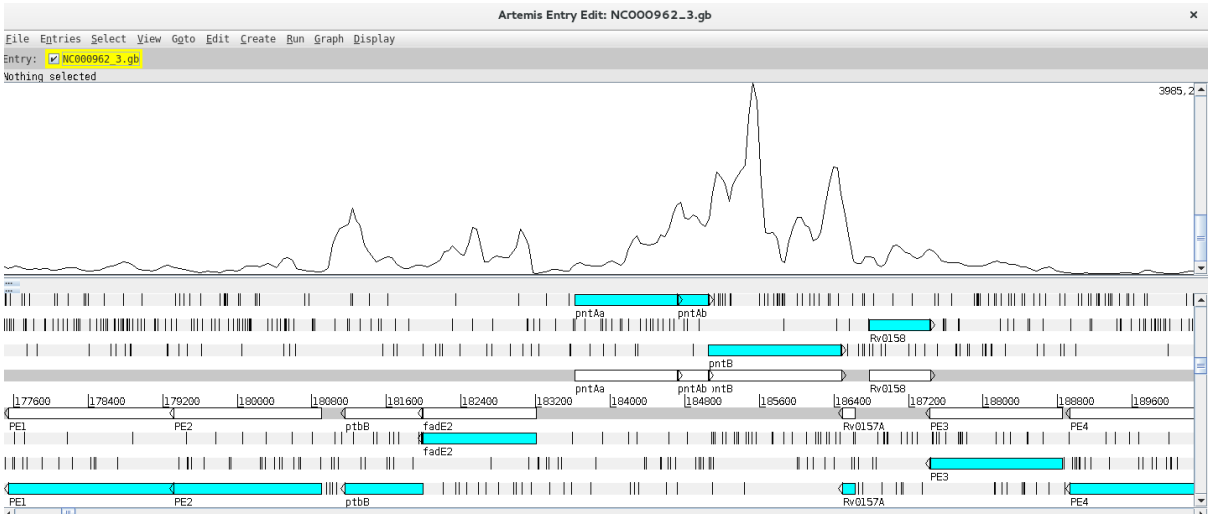
This will start Artemis. You must first load your reference genome, which can be done in multiple ways:

i) you can open the NC000962_2.gb file, containing the genome and features in the GenBank file, or;

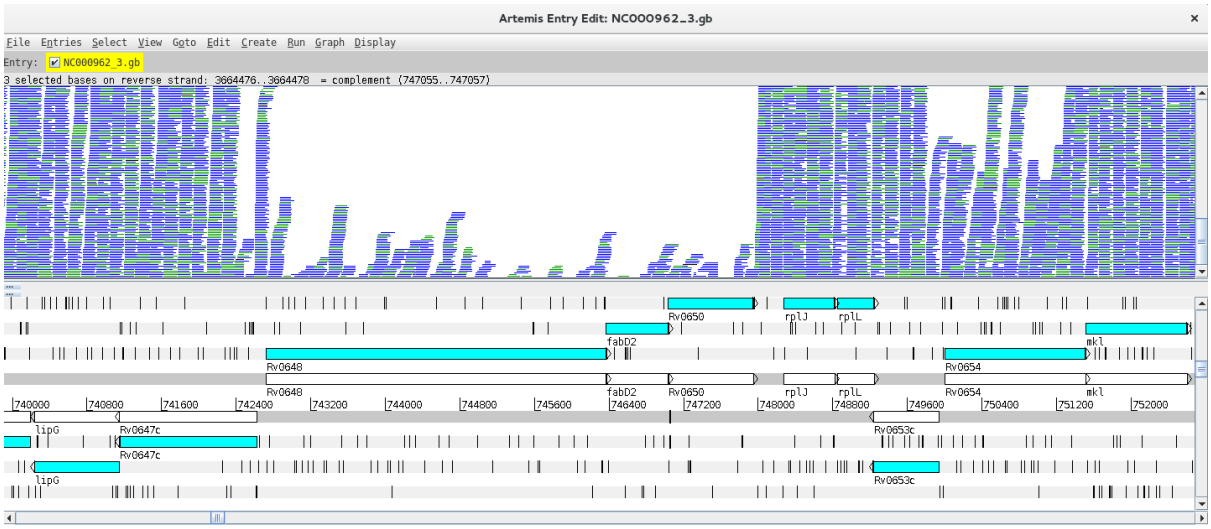
ii) you can open the NC000962_3.fasta file and then read in the GFF file so that you can see the different feature tracks.

Next, you need to read in your BAM files. You can start by going to File >> Read BAM/VCF and select the first BAM file from the list. Please remember that to load a BAM file in Artemis you need to have the BAM file and the respective .bai index file in the same directory. If everything went ok a BAM window with the mapped reads should appear on top of the genome:

Depending on whether the genome map window is too zoomed in or out you will see the individual reads or a plot. You can change the zoom level in the vertical scroll bar on the right and by right clicking on the BAM window you can also select the type of graph under *View* (try changing between pileup and coverage for example).

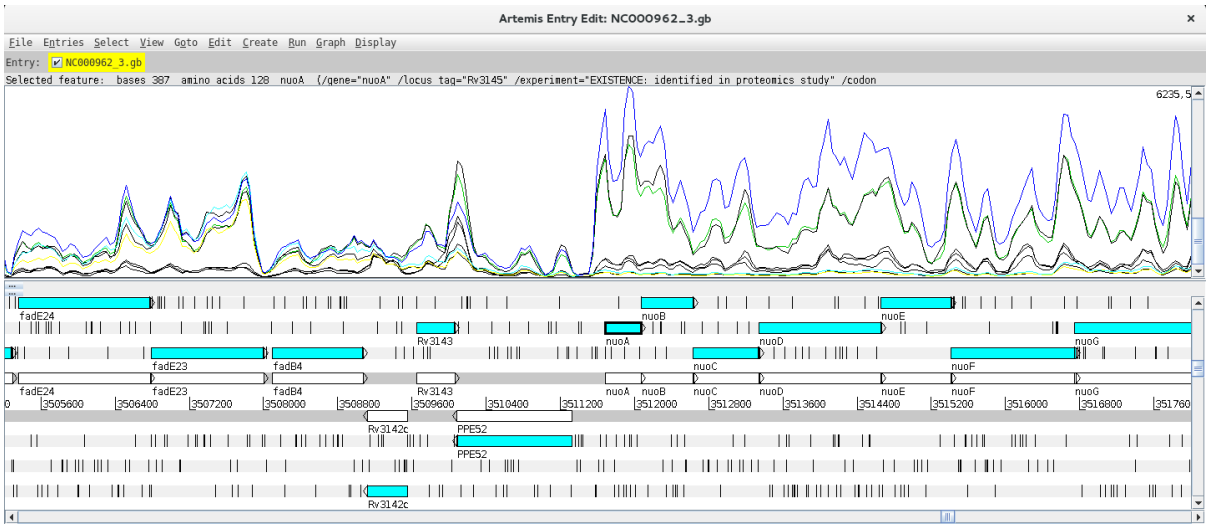


Let's look at a particular gene, for example *fabD12*, by using the Navigator (*Goto >> Navigator*) and typing the gene name in the appropriate search box.



Notice that in the vicinity of these genes, there are some with no/low coverage while others have coverage and there are even reads mapping to intergenic regions. Why is that?

Next, you can add and compare the coverage from other BAM files. To do this just right click on the BAM window, select "*Add BAM*" and add the different BAM files. Also, changing the Graph type to coverage will facilitate the interpretation of the Data. By right clicking on the BAM window you also have the option to configure these lines (changing the colour, thickness, etc.).



Can you find any differentially expressed genes (DEG) using this approach?

Hint: Try the *nuo* gene!

Exercise 3 – Examining Differential Expression

This last exercise is aimed at identifying differentially expressed genes (DEGs) by taking the raw counts as input to software that normalizes the number of mapped reads and apply statistical tests to identify DEGs. The most widely used packages for this purpose are the R packages DESeq2 and EdgeR. In this practical we will use DESeq2 to identify DEGs, but code DEG identification using EdgeR is also provided and you can even compare the results between packages [6, 7].

It is also important to stress that using different replicates is a fundamental aspect of DEG analysis as expression levels are variable and depend on several external factors. Biological replicates are therefore essential to account for the variability of biological systems and more accurately compare expression levels between groups/conditions.

The exercise will be carried out in R which involves a different programming language. Let's go step by step.

Let's start R by typing:

```
$ R
```

Inside the R command line we need to load the required libraries for this part:

```
> library(DESeq2)
> library(gplots)
```


Next, we will list the files with counts into an object called *sampleFiles* and create an object with the condition/genotype. It is important that the order of the genotypes is the same as the order of the listed files:

```
> sampleFiles<-list.files(pattern="count.txt$")
#Use the same order from the sampleFiles object:
> sampleCondition <- c("Control","Control","SigBKO","SigBKO",
                       "SigAKO_ATc","SigAKO_ATc","SigAKO_ATc",
                       "SigAKO_NoATc","SigAKO_NoATc","SigAKO_NoATc")
```

We can then create a data frame, which is a type of R object with a table structure called *sampleTable*:

```
> sampleTable <- data.frame(
  sampleName = gsub("_count.txt","",sampleFiles),
  fileName = sampleFiles,
  condition = sampleCondition
)
```

Notice that you defined the columns in this data frame and for sample names you run the object containing the file list through a substitution command to remove the suffix of the file names.

You can check the structure of the data frame just by typing:

```
> sampleTable
```

Please check if everything as we will next move to the analytical stage. You can import the HTSeq counts into a DESeqDataSet object and look at the contents of this object by running the following:

```
> dds <- DESeqDataSetFromHTSeqCount(
  sampleTable = sampleTable,
  design = ~ condition
)

> dds
```

Pay attention to the design argument of the first function, you need to specify the grouping factor. This argument can accommodate multiple designs, this is the simplest

design scheme in which you specified the name of a single column from your original dataframe.

A pre-filtering step can be carried out to reduce the number of genes with a count below 10. This step is not essential, but it can help to reduce the size of the DESeqDataSet object. To do this, run the following:

```
> keep <- rowSums(counts(dds)) >= 10
> dds <- dds[keep,]
```

Next, setting the factor levels is important to define the reference level (control group). If this is not specified the levels will be taken in alphabetical order and the reference level will be assumed to be the first one. Below there are two options to set the levels:

```
> dds$condition <- factor(dds$condition, levels = c("Control",
"SigBKO", "SigAKO_ATc", "SigAKO_NoATc"))
# Notice that wt comes first as this is the natural reference group
(control)
# or
> dds$condition <- relevel(dds$condition,ref="Control")
```

The differential expression analysis is carried out using the *DESeq* function that takes the DESeqDataSet object as input, does the normalization of the counts, estimates dispersion and additional statistical tests. After, the result table can be generated using the results function *results* :

```
> dds <- DESeq(dds)
> res <- results(dds, contrast = c("condition", "SigAKO_ATc",
"SigAKO_NoATc"))
```

In the second command we have just used the contrast argument to the results function which might not be necessary but enables you to specifically control the comparison being made across groups. To look at the results table just type:

```
> res
```

What do you see? Something like this:

```
log2 fold change (MLE): condition ompR_KO vs wt
Wald test p-value: condition ompR_KO vs wt
DataFrame with 4069 rows and 6 columns
           baseMean log2FoldChange     lfcSE       stat      pvalue
          <numeric>      <numeric> <numeric>  <numeric>   <numeric>
VK055_0008   14.3907      0.0332812  0.730853  0.0455375   0.9636789
VK055_0010   21.9589     -0.7356173  0.594794 -1.2367594   0.2161764
VK055_0011   17.0704     -0.8991868  0.657200 -1.3682097   0.1712464
VK055_0012   70.2776     -0.9149246  0.329761 -2.7745097   0.0055285
VK055_0015  354.6755     -0.0134678  0.175108 -0.0769112   0.9386942
...              ...            ...       ...        ...         ...
zntR         68.7449      -0.248428  0.329993   -0.75283 4.51552e-01
znuA       1430.5374      -0.672363  0.125105   -5.37440 7.68373e-08
zupT        298.8757      -0.620975  0.183087   -3.39169 6.94641e-04
zur         741.3930      -0.153331  0.131868   -1.16276 2.44928e-01
zwf         627.0477      -0.254292  0.139242   -1.82626 6.78118e-02
                 padj
            <numeric>
VK055_0008  0.9761537
VK055_0010  0.3720904
VK055_0011  0.3154032
VK055_0012  0.0199075
VK055_0015  0.9604458
...               ...
zntR       6.14540e-01
znuA       7.87534e-07
zupT       3.23398e-03
zur        4.05787e-01
zwf        1.55715e-01
```

You can summarize the results by running:

```
> summary(res)
```

```
out of 4605 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 745, 16%
LFC < 0 (down)     : 839, 18%
outliers [1]       : 0, 0%
low counts [2]     : 536, 12%
(mean count < 7)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

You should see the standard adjusted p-value threshold used for the summary and the number of genes with a Log_2 Fold Change (LFC) above and below 0.

Next let's remove results with no adjusted p-values and sort the table by this value:

```
> res <- res[!is.na(res$padj),]
> resOrdered <- res[order(res$pvalue),]
> resOrdered
```

You can construct a heatmap for the top 50 genes using the following commands:

```
> counts_heatmap <- counts(dds, normalized = TRUE)

> idx <- rownames(resOrdered)[1:100]

> counts_heatmap <- counts_heatmap[rownames(counts_heatmap) %in% idx,]


> counts_heatmap


> colnames(counts_heatmap) <- c("Control1", "Control2", "SigBKO1", "SigBKO2", "SigAKO_ATc1", "SigAKO_ATc2", "SigAKO_ATc3", "SigAKO_NoATc1", "SigAKO_NoATc2", "SigAKO_NoATc3")

>heatmap.2(as.matrix(counts_heatmap), scale="row", col=greenred(75), Rowv=NA, dendrogram = "col", trace="none", density.info = "none")
```

Another option is to visualize Gene Plot Counts for multiple genes:

```
> par(mfrow=c(2,3))

> plotCounts(dds,gene="PE31", intgroup="condition")

> plotCounts(dds,gene="PPE60", intgroup="condition")

> plotCounts(dds,gene="lpqQ", intgroup="condition")

> plotCounts(dds,gene="mutB", intgroup="condition")

> plotCounts(dds,gene="Rv3912", intgroup="condition")

> plotCounts(dds,gene="Rv0585c", intgroup="condition")

> par(mfrow=c(1,1))
```

You can also do a Principal Component Analysis of the results:

```
> vsdata<-vst(dds,blind=FALSE)

> z <- plotPCA(vsdata,intgroup="condition")

> z + coord_fixed(ylim=c(-30,30), xlim=c(-30,30))
```

Do the global transcriptomic signatures from different conditions cluster in this analysis?

For a Volcano Plot, you can construct a basic one:

```
> with(res, plot(log2FoldChange, -log10(pvalue),
pch=20,main="Volcano plot",xlim=c(-8,8)))
```

or highlight in blue if the adjusted p-value is below 0.01 or red if the adjusted p-value is below 0.01 and the Log2 Fold Change is higher than 1:

```
> with(subset(res,padj<.01),points(log2FoldChange, -
log10(pvalue),pch=20,col="blue"))

> with(subset(res,padj<.01 &
abs(log2FoldChange)>2),points(log2FoldChange, -
log10(pvalue),pch=20,col="red"))
```

Moreover, you can write you results table to a csv file, or create individual files of upregulated or downregulated genes based on specific thresholds that you can decide on. For example:

```
> write.table(resOrdered,file="resOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)


##The following produce different files based on criteria adjusted
p-value < 0.05 and Log2 Fold Change > 0.5, sorted by the adjusted p-
value:

> res_sig<-subset(res,padj<0.05)

> res_sig_up<-subset(res_sig, log2FoldChange > 0.5)

> res_sig_upOrdered<-res_sig_up[order(res_sig_up$padj),]

> write.table(res_sig_upOrdered, file="res_sig_upOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)



> res_sig_down<-subset(res_sig, log2FoldChange < 0.5)

> res_sig_downOrdered<-res_sig_down[order(res_sig_down$padj),]

> write.table(res_sig_downOrdered, file="res_sig_downOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)
```

You can then open the CSV files in MS Excel or LibreOffice Calc and examine the results. Please note that you must adjust the column headers.

References

1 Cambier CJ, Falkow S, Ramakrishnan L. Host evasion and exploitation schemes of Mycobacterium tuberculosis. *Cell*. 2014; **159**: 1497-1509.

2 Ernst JD. The immunological life cycle of tuberculosis. *Nat Rev Immunol*. 2012; **12**: 581-591.

3 Sachdeva P, Misra R, Tyagi AK, Singh Y. The sigma factors of Mycobacterium tuberculosis: regulation of the regulators. *FEBS J*. 2010; **277**: 605-626.

4 Manganelli R. Sigma Factors: Key Molecules in Mycobacterium tuberculosis Physiology and Virulence. *Microbiol Spectr*. 2014; **2**: MGM2-0007-2013.

5 Anders S, Pyl PT, Huber W. HTSeq--a Python framework to work with high-throughput sequencing data. *Bioinformatics*. 2015; **31**: 166-169.

6 Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol*. 2014; **15**: 550.

7 Robinson MD, McCarthy DJ, Smyth GK. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*. 2010; **26**: 139-140.

6.4 Case-Study: *Aspergillus fumigatus* Transcriptomics – Role of the transcriptional regulator AtrR as a determinant of azole resistance

Introduction

Opportunistic fungal infections, such as Invasive Aspergillosis, are increasingly recognized as a major health problem owing to the development of antineoplastic and immunosuppressive therapies [1]. Aspergillus fumigatus is the most frequent fungal species associated with invasive aspergillosis and frequently associated with poor outcomes [1, 2]. Moreover, evidence for the emergence of resistance to azoles is mounting with considerable clinical implications [3]. Paul et al (2019) sought to understand the role of AtrR transcription factor in the development of azole resistance using a multidisciplinary approach which included the transcriptomic characterization of AtrR mutant strains [4]. This practical covers a partial analysis of the data obtained to further elucidate the role of the AtrR regulator in azole resistance.

Citation:

Paul S, Stamnes M, Thomas GH, Liu H et al. AtrR Is an Essential Determinant of Azole Resistance in *Aspergillus fumigatus*. *mBio* 2019 Mar 12;10(2). PMID: 30862750

The data for this practical is available through the NCBI Gene Expression Omnibus (GEO) database under accession GSE123445. The dataset comprises eight samples, subjected to RNA-Seq by paired-end sequencing, representing a *atrR* wild-type strain, a null mutant, a hyperactive tagged *atrR* form and and an overexpressed wild-type *atrR* strain (2 biological replicates each, Table 1):

Table 1 – List of runs and biosample accessions and associated genotypes.

Run	BioSample	Genotype/Treatment
SRR8286614	SAMN10527538	Wild-type
SRR8286615	SAMN10527537	Null mutant
SRR8286616	SAMN10527544	Hyperactive epitope tagged form
SRR8286617	SAMN10527543	Overexpressed wild-type factor
SRR8286618	SAMN10527540	Wild-type
SRR8286619	SAMN10527539	Null mutante
SRR8286620	SAMN10527542	Hyperactive epitope tagged form
SRR8286621	SAMN10527541	Overexpressed wild-type factor

The practical is divided in three exercises, Exercise 1 pertains the mapping and generation of raw transcript count and has been carried out ahead since the computational steps involved are time consuming. Nonetheless, the commands executed are outlined below along with explanatory context. The second exercise will involve the visualization of mapped reads and the third exercise will cover the identification of differentially expressed genes across the conditions tested.

Exercise 1 – Mapping and Raw read Count

The initial stage of the analytical process of RNA-Seq data for this practical will consist of mapping the reads to a reference genome following a similar approach as the one already covered in the mapping module for this course. As such, this will require an adequate genome reference file. An annotation file covering all genes in this same reference genome will also be needed. We will use the genome of *Aspergillus fumigatus* Af293 as reference genome, which has already been downloaded in the FASTA format, GenBank format and annotation (GFF3) from NCBI GenBank (accession GCF_000002655.1). The following files are therefore already present in the course module directory in the Af subdirectory. Let's start!

From the home directory type:

```
$ cd Module6/Af
$ ls
```

Please note for the following files:

- ASM265_1.fasta – reference genome in the fasta format;
- CM000169_1.gb - Chromosome 1 of the A. fumigatus Af293 in the GenBank format;
- ASM2665_1.gff3 – annotation file in the GFF3 format.

You can look inside these files to confirm its format.

In the same directory you will find compressed FastQ files containing the RNA-Seq raw reads. Each is named according to the Run ID in Table 1 and if this is paired-end sequencing, it will have _1.fastq.gz or _2.fastq.gz appended to the Run ID.

Which one do you have? _____

The mapping steps were done using the following commands:

```
$ bwa mem ASM265_1.fasta SRR8286614_1.fastq.gz
SRR8286614_2.fastq.gz | samtools sort --write-index -o
SRR8286614.bam -

$ bwa mem ASM265_1.fasta SRR8286615_1.fastq.gz
SRR8286615_2.fastq.gz | samtools sort --write-index -o
SRR8286615.bam -

$ bwa mem ASM265_1.fasta SRR8286616_1.fastq.gz
SRR8286616_2.fastq.gz | samtools sort --write-index -o
SRR8286616.bam -

$ bwa mem ASM265_1.fasta SRR8286617_1.fastq.gz
SRR8286617_2.fastq.gz | samtools sort --write-index -o
SRR8286617.bam -

$ bwa mem ASM265_1.fasta SRR8286618_1.fastq.gz
SRR8286618_2.fastq.gz | samtools sort --write-index -o
SRR8286618.bam -

$ bwa mem ASM265_1.fasta SRR8286619_1.fastq.gz
SRR8286619_2.fastq.gz | samtools sort --write-index -o
SRR8286619.bam -

$ bwa mem ASM265_1.fasta SRR8286620_1.fastq.gz
SRR8286620_2.fastq.gz | samtools sort --write-index -o
SRR8286620.bam -

$ bwa mem ASM265_1.fasta SRR8286621_1.fastq.gz
SRR8286621_2.fastq.gz | samtools sort --write-index -o
SRR8286621.bam -
```

Each command allows for mapping of the reads using BWA MEM algorithm and pipes it to samtools in order generate a BAM containing all mapped reads and reference. Next you need to index the BAM file by typing:

```
$ samtools index SRR8286614.bam
$ samtools index SRR8286615.bam
$ samtools index SRR8286616.bam
$ samtools index SRR8286617.bam
$ samtools index SRR8286618.bam
$ samtools index SRR8286619.bam
$ samtools index SRR8286620.bam
$ samtools index SRR8286621.bam
```


At this point you can already visualize your mapping data but we will leave it to the second part of this practical. For now we need to count the number of reads mapped to each feature/gene. Please note that for this stage you will need to know the position, i.e., start, stop and strand of the genes in your gene. This information is not present in the FASTA reference genome you used to map the reads nor you can extract it directly from the BAM files. This is where the GFF file is needed! You will be using HTSeq to count the reads in the genes, specifically its *htseq-count* script [5]. By typing:

```
$ htseq-count
#or
$ htseq-count --help
```

You will learn a little bit of the different files needed, arguments to this command and output. You can produce the files containing the raw counts by doing the following:

```
$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR8286614.bam
ASM265_1.gff > SRR8286614_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR8286615.bam
ASM265_1.gff > SRR8286615_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR8286616.bam
ASM265_1.gff > SRR8286616_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR8286617.bam
ASM265_1.gff > SRR8286617_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR8286618.bam
ASM265_1.gff > SRR8286618_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR8286619.bam
ASM265_1.gff > SRR8286619_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR8286620.bam
ASM265_1.gff > SRR8286620_count.txt

$ htseq-count -f bam -r pos -s reverse -t gene -i Name SRR8286621.bam
ASM265_1.gff > SRR8286621_count.txt
```

Can you check why are we using those extra arguments besides the BAM and the GFF file?

Also, please note that the output is being redirected to files named *_count.txt files. Look at their content. Does it resemble something like this:

```
yjdC    985
yjdE    8
yjdH    124
yjeE    24
yjeK    132
yjeR    161
yjfF    16
yjfP    64
yjgD    589
yjgK    18
yjjX    322
ykfE    1821
yliI    508
ymgB    310
ynfA    17
yodA    934
yohD    540
yqaA    27
```


These files contain the raw read counts for each gene and will be used as input for the Differential Gene Expression analysis in Exercise 3. But before that let us visualize the mapped data so that you can understand better what we have done so far and what these counts represent.

Notice that you already have the files with the raw counts! To download all previously created BAM files and index files for this practical session, please go to:

https://ulisboa-my.sharepoint.com/:f:/g/personal/jperdigao_office365_ulisboa_pt/EkBjH_nIesdIlqBXPIXy3_oB4NG7dg4RAksXHik2lODmbQ?e=fyu1w7

and download the bam_files_AF.tar.gz file (use command: *tar -xvf bam_files_AF.tar.gz* to decompress the file). Run this last command in the Af sub-directory after downloading the file to this location.

Exercise 2 – Visualization of RNA-Seq mapped data

For the next exercise you can visualize the mapped reads in Artemis. Inside the directory containing the data, type:

```
$ art
```

This will start Artemis. You must first load your reference genome, which can be done in multiple ways:

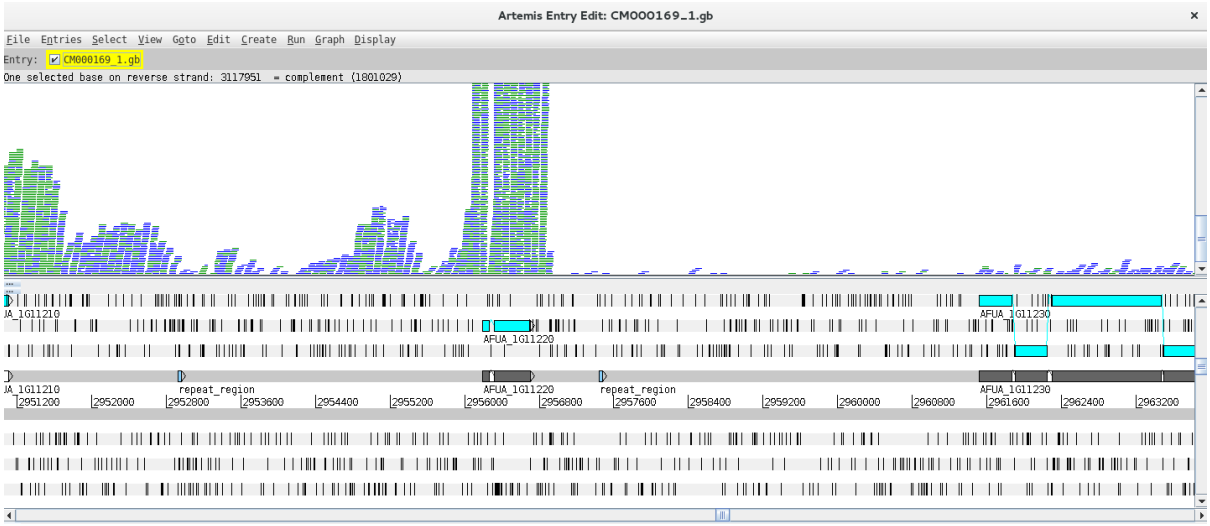
i) you can open the NC000962_2.gb file, containing the genome and features in the GenBank file, or;

ii) you can open the NC000962_3.fasta file and then read in the GFF file so that you can see the different feature tracks.

Next, you need to read in your BAM files. You can start by going to File >> Read BAM/VCF and select the first BAM file from the list. Please remember that to load a BAM file in Artemis you need to have the BAM file and the respective .bai index file in the same directory. If everything went ok a BAM window with the mapped reads should appear on top of the genome:

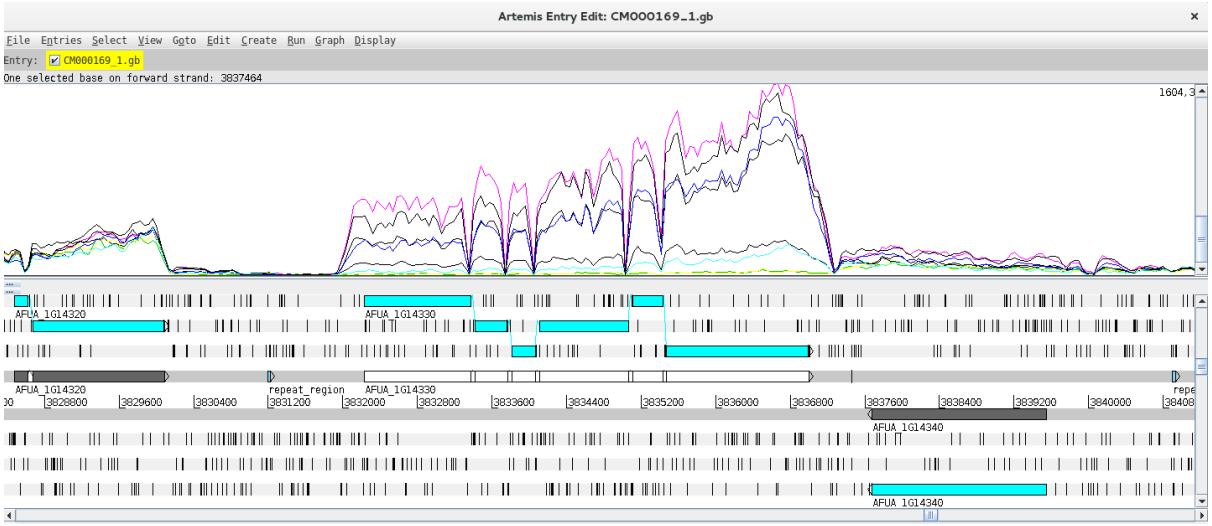
Depending on whether the genome map window is too zoomed in or out you will see the individual reads or a plot. You can change the zoom level in the vertical scroll bar on the right and by right clicking on the BAM window you can also select the type of graph under *View* (try changing between pileup and coverage for example).

Let's look at a particular gene, for example the *AFUA_1G11220 gene*, by using the Navigator (*Goto >> Navigator*) and typing the gene name in the appropriate search box.



Notice that in the vicinity of these genes, there are some with no/low coverage while others have coverage and there are even reads mapping to intergenic regions. Why is that?

Next, you can add and compare the coverage from other BAM files. To do this just right click on the BAM window, select "*Add BAM*" and add the different BAM files. Also, changing the Graph type to coverage will facilitate the interpretation of the Data. By right clicking on the BAM window you also have the option to configure these lines (changing the colour, thickness, etc.).



Can you find any differentially expressed genes (DEG) using this approach?

Hint: Try the *AFUA_1G14330* gene!

Exercise 3 – Examining Differential Expression

This last exercise is aimed at identifying differentially expressed genes (DEGs) by taking the raw counts as input to software that normalizes the number of mapped reads and apply statistical tests to identify DEGs. The most widely used packages for this purpose are the R packages DESeq2 and EdgeR. In this practical we will use DESeq2 to identify DEGs, but code DEG identification using EdgeR is also provided and you can even compare the results between packages [6, 7].

It is also important to stress that using different replicates is a fundamental aspect of DEG analysis as expression levels are variable and depend on several external factors. Biological replicates are therefore essential to account for the variability of biological systems and more accurately compare expression levels between groups/conditions.

The exercise will be carried out in R which involves a different programming language. Let's go step by step.

Let's start R by typing:

```
$ R
```

Inside the R command line we need to load the required libraries for this part:

```
> library(DESeq2)
> library(gplots)
```


Next, we will list the files with counts into an object called *sampleFiles* and create an object with the condition/genotype. It is important that the order of the genotypes is the same as the order of the listed files:

```
> sampleFiles<-list.files(pattern="count.txt$")
#Use the same order from the sampleFiles object:
> sampleCondition <- c ("atrR_wt", "atrR_Del", "atrR_HA",
"PhspA_atrR", "atrR_wt", "atrR_Del",  "atrR_HA", "PhspA_atrR")
```

We can then create a data frame, which is a type of R object with a table structure called *sampleTable*:

```
> sampleTable <- data.frame(
  sampleName = gsub("_count.txt","",sampleFiles),
  fileName = sampleFiles,
  condition = sampleCondition
)
```

Notice that you defined the columns in this data frame and for sample names you run the object containing the file list through a substitution command to remove the suffix of the file names.

You can check the structure of the data frame just by typing:

```
> sampleTable
```

Please check if everything as we will next move to the analytical stage. You can import the HTSeq counts into a DESeqDataSet object and look at the contents of this object by running the following:

```
> dds <- DESeqDataSetFromHTSeqCount(
  sampleTable = sampleTable,
  design = ~ condition
)

> dds
```

Pay attention to the design argument of the first function, you need to specify the grouping factor. This argument can accommodate multiple designs, this is the simplest design scheme in which you specified the name of a single column from your original dataframe.

A pre-filtering step can be carried out to reduce the number of genes with a count below 10. This step is not essential, but it can help to reduce the size of the DESeqDataSet object. To do this, run the following:

```
> keep <- rowSums(counts(dds)) >= 10
> dds <- dds[keep,]
```

Next, setting the factor levels is important to define the reference level (control group). If this is not specified the levels will be taken in alphabetical order and the reference level will be assumed to be the first one. Below there are two options to set the levels:

```
> dds$condition <- factor(dds$condition, levels = c("atrR_wt",
"atrR_HA", "atrR_Del", "PhspA_atrR"))

# Notice that wt comes first as this is the natural reference group
(control)

# or

> dds$condition <- relevel(dds$condition,ref="atrR_wt")
```


The differential expression analysis is carried out using the *DESeq* function that takes the DESeqDataSet object as input, does the normalization of the counts, estimates dispersion and additional statistical tests. After, the result table can be generated using the results function *results* :

```
> dds <- DESeq(dds)

> res <- results(dds, contrast = c("condition","atrR_HA","atrR_wt"))
```

In the second command we have just used the contrast argument to the results function which might not be necessary but enables you to specifically control the comparison being made across groups. To look at the results table just type:

```
> res
```

What do you see? Something like this:

```
log2 fold change (MLE): condition ompR_KO vs wt
Wald test p-value: condition ompR_KO vs wt
DataFrame with 4069 rows and 6 columns
            baseMean log2FoldChange     lfcSE       stat      pvalue
           <numeric>      <numeric> <numeric>  <numeric>   <numeric>
VK055_0008   14.3907      0.0332812  0.730853  0.0455375   0.9636789
VK055_0010   21.9589     -0.7356173  0.594794 -1.2367594   0.2161764
VK055_0011   17.0704     -0.8991868  0.657200 -1.3682097   0.1712464
VK055_0012   70.2776     -0.9149246  0.329761 -2.7745097   0.0055285
VK055_0015  354.6755     -0.0134678  0.175108 -0.0769112   0.9386942
...               ...            ...       ...        ...         ...
zntR         68.7449      -0.248428  0.329993   -0.75283 4.51552e-01
znuA       1430.5374      -0.672363  0.125105   -5.37440 7.68373e-08
zupT        298.8757      -0.620975  0.183087   -3.39169 6.94641e-04
zur         741.3930      -0.153331  0.131868   -1.16276 2.44928e-01
zwf         627.0477      -0.254292  0.139242   -1.82626 6.78118e-02
                  padj
             <numeric>
VK055_0008   0.9761537
VK055_0010   0.3720904
VK055_0011   0.3154032
VK055_0012   0.0199075
VK055_0015   0.9604458
...                ...
zntR       6.14540e-01
znuA       7.87534e-07
zupT       3.23398e-03
zur        4.05787e-01
zwf        1.55715e-01
```

You can summarize the results by running:

```
> summary(res)
```

```
out of 4605 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 745, 16%
LFC < 0 (down)     : 839, 18%
outliers [1]       : 0, 0%
low counts [2]     : 536, 12%
(mean count < 7)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

You should see the standard adjusted p-value threshold used for the summary and the number of genes with a Log_2 Fold Change (LFC) above and below 0.

Next let's remove results with no adjusted p-values and sort the table by this value:

```
> res <- res[!is.na(res$padj),]
> resOrdered <- res[order(res$pvalue),]
> resOrdered
```


You can construct a heatmap for the top 50 genes using the following commands:

```
> counts_heatmap <- counts(dds, normalized = TRUE)
> idx <- rownames(resOrdered)[1:100]
> counts_heatmap <- counts_heatmap[rownames(counts_heatmap) %in% idx,]

> counts_heatmap

> colnames(counts_heatmap) <- c("atrR_wt1", "atrR_Del1", "atrR_HA1", "PhspA_atrR1", "atrR_wt2", "atrR_Del2", "atrR_HA2", "PhspA_atrR2")
>heatmap.2(as.matrix(counts_heatmap), scale="row", col=greenred(75), Rowv=NA, dendrogram = "col", trace="none", density.info = "none")
```

Another option is to visualize Gene Plot Counts for multiple genes:

```
> par(mfrow=c(2,3))
> plotCounts(dds,gene="AFUA_5G10250", intgroup="condition")
> plotCounts(dds,gene="AFUA_2G17240", intgroup="condition")
> plotCounts(dds,gene="AFUA_7G06330", intgroup="condition")
> plotCounts(dds,gene="AFUA_1G16970", intgroup="condition")
> plotCounts(dds,gene="AFUA_2G02590", intgroup="condition")
> plotCounts(dds,gene="AFUA_2G14090", intgroup="condition")
> par(mfrow=c(1,1))
```


You can also do a Principal Component Analysis of the results:

```
> vsdata<-vst(dds,blind=FALSE)
> z <- plotPCA(vsdata,intgroup="condition")
> z + coord_fixed(ylim=c(-30,30), xlim=c(-35,35))
```

Do the global transcriptomic signatures from different conditions cluster in this analysis?

For a Volcano Plot, you can construct a basic one:

```
> with(res, plot(log2FoldChange, -log10(pvalue),
pch=20,main="Volcano plot",xlim=c(-8,8)))
```

or highlight in blue if the adjusted p-value is below 0.01 or red if the adjusted p-value is below 0.01 and the Log2 Fold Change is higher than 1:

```
> with(subset(res,padj<.01),points(log2FoldChange, -
log10(pvalue),pch=20,col="blue"))
> with(subset(res,padj<.01 &
abs(log2FoldChange)>2),points(log2FoldChange, -
log10(pvalue),pch=20,col="red"))
```

Moreover, you can write you results table to a csv file, or create individual files of upregulated or downregulated genes based on specific thresholds that you can decide on. For example:

```
> write.table(resOrdered,file="resOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)


##The following produce different files based on criteria adjusted
p-value < 0.05 and Log2 Fold Change > 0.5, sorted by the adjusted p-
value:

> res_sig<-subset(res,padj<0.05)

> res_sig_up<-subset(res_sig, log2FoldChange > 0.5)

> res_sig_upOrdered<-res_sig_up[order(res_sig_up$padj),]

> write.table(res_sig_upOrdered, file="res_sig_upOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)



> res_sig_down<-subset(res_sig, log2FoldChange < 0.5)

> res_sig_downOrdered<-res_sig_down[order(res_sig_down$padj),]

> write.table(res_sig_downOrdered, file="res_sig_downOrdered.csv",
sep=";",row.names=TRUE, quote = FALSE, col.names=TRUE)
```

You can then open the CSV files in MS Excel or LibreOffice Calc and examine the results. Please note that you must adjust the column headers.

References

1 Rivelli Zea SM, Toyotome T. Azole-resistant Aspergillus fumigatus as an emerging worldwide pathogen. *Microbiol Immunol*. 2022; **66**: 135-144.

2 Rhodes J, Abdolrasouli A, Dunne K, et al. Population genomics confirms acquisition of drug-resistant Aspergillus fumigatus infection by humans from the environment. *Nat Microbiol*. 2022; **7**: 663-674.

3 Lestrade PPA, Meis JF, Melchers WJG, Verweij PE. Triazole resistance in Aspergillus fumigatus: recent insights and challenges for patient management. *Clin Microbiol Infect*. 2019; **25**: 799-806.

4 Paul S, Stamnes M, Thomas GH, et al. AtrR Is an Essential Determinant of Azole Resistance in Aspergillus fumigatus. *mBio*. 2019; **10**.

5 Anders S, Pyl PT, Huber W. HTSeq--a Python framework to work with high-throughput sequencing data. *Bioinformatics*. 2015; **31**: 166-169.

6 Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol*. 2014; **15**: 550.

7 Robinson MD, McCarthy DJ, Smyth GK. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*. 2010; **26**: 139-140.