My first genome assembly - summer school 23

Introduction to supercomputers

What is a supercomputer?

A supercomputer is just a computer that has very high levels of performance. Super computers are becoming increasingly important as the volume of data created, for example by full genome sequencing, increases.

A typical laptop will have between 4 and 12 cores (maybe 24 if you have a really nice one!). Hamilton 8, Durham's supercomputer, has over 15,000 cores. A core is part of a computer capable of completing a computation/task.

Unfortunately Durham's supercomputer has been closed for maintenance this week! However, we will still be working in the command line in the same way as we do when using the super computer.

Logging in

We are going to be using a Linux computer in for this practical. To start with you will need to take the following steps.

1. Turn the computer on below your desk
2. When prompted use the arrow keys to select the "Boot Linux" option
3. Select the "Debian GNU" option
4. Enter in your temporary university username and password
5. Select "Accept" to the terms and conditions of using a university computer
6. To load the command line click on the small black square in the top left-hand side of the screen.

Using the command line

When using the Hamilton 8 supercomputer we access files and programmes using the command line instead of a graphical user interface (GUI) like we have on our Windows PCs and Macs. This is primarily to save computational resources. Thankfully using the command line is easier than it may initially seem - Google is your friend! Here are some useful commands for this practical:

The most basic command is to ask the computer to print something. In this case we use the `echo` command followed by the text we want to be printed. Type the following out into the command line and press the enter key.

```
echo 'Hello world!'
```

The echo command lets you set markers in your code, which is helpful for checking where the code failed. For instance you could write `echo 'TASK 1 COMPLETE'`. If the code fails and you don't see 'TASK 1 COMPLETE' printed out on the screen the problem must have occurred prior to that line.

Further commands allow us to extract information from the computer.
The list command, `ls` , prints out the names of all of the files within a directory (directory is the Linux word for folder.)

```
ls
```

This command is useful to check all the correct files are all present, as well as a way to make sure you are in the correct directory (folder).

Make directory - the command below will make a new directory called "genome_assembly". Note: its pretty useful when using command line bioinformatics to avoid creating folders or files without spaces in the name.

```
mkdir genome_assembly
```

You can use the `ls` command to check that this new directory has been created.

When working on the command line some important information to find out is what the current working directory is. This is the directory (think folder) that the computer will default to working in if you don't specify any other information. For instance there are many, many, directories on a computer, but the `mkdir` command we specified beforehand will be created within the current working directory.

To check what the working directory is we use the command `pwd` which stands for "print working directory".

We can also change what the working directory is. Use the following code to change the working directory to the `genome_assembly` folder you created earlier and then check that it has worked by using `pwd`.

```
cd genome_assembly
pwd
```

Note if you get lost changing around directories at any point today you can type in `cd` on its own to go back to the default working directory.

Downloading data

We are now going to play around with files. We have a uploaded a range of file types to the the website GitHub. To download the data to the computer you type the following code into the command line.

```
git clone https://github.com/ChristophePatterson/My-first-Genome.git
```

This should have created a folder within your current working directory called "My-first-Genome". You can check that by using the `ls` command. After than change your current working directory to be the folder "My-first-genome" and then check there are a range of file types within this folder.

Viewing files

We have provided you with a file called `TREASURE_ISLAND.txt`. This contains the entire children's book Treasure Island by Robert Louis Stevenson.

Check the file is contained within your working directory by using the `ls` command. If you can see the file printed you can use the follow commands.

`head` prints the first 10 lines of a file

```
head TREASURE_ISLAND.txt
```

`tail` prints the last 10 lines of a file

```
tail TREASURE_ISLAND.txt
```

To avoid spelling mistakes and to save time, start typing TR and then hit the 'Tab' key. This is known as a tab complete and is VERY useful when using the command line.

The number of lines printed out can be altered by changing the code to include `-n`. The following line of code will print out the first 30 lines of the text file.

```
head -n 30 TREASURE_ISLAND.txt
```

Can you change the `tail` command so that it prints out the final 40 lines of the book?

We can also find out information about a text file by using the `wc` command. You can find out how many lines of text are within a text file using `wc -l` command. Note that's a lowercase L for *Laura*, not a number one.

```
wc -l TREASURE_ISLAND.txt
```

You can find out how many individual letters or numbers (characters) are in a file by swapping out the `-l` for `-m`.

There is a another command call `cat`. The `cat` command print out the entire text file in one go. This is very rarely used in bioinformatics because printing out 48 gigabytes of the letters A, C, T, or G, isn't normally useful. You could print out the entire book of treasure island by typing out the line `cat TREASURE_ISLAND.txt`. But! be careful of the size of text file you print using the `cat` command, you do not want text flying past your screen for hours.

Searching for information

We can search for information within a text file using the `grep` command. The following line searches a text file for the string of letters "Jim" and then prints them to the screen.

```
grep 'Jim' TREASURE_ISLAND.txt
```

The next useful bit of coding we are going to look at is "porting". Porting takes the output of one command and puts it straight into the next command. We use the `|` symbol for this. The `|` symbol is normally just to the right of the left-hand shift key.

So if we wanted to know how many lines "Jim" is mentioned on we can port the output of the previous code into the `wc` command.

```
grep 'Jim' TREASURE_ISLAND.txt | wc -l
```

There are a number of other charaters in the book, Treasure Island, "Long John", "Ben", "Trelawney", and "Flint". Can you find out how many times each is mentioned?

Bioinformatics

We are now going to move on to working with DNA data. Because DNA can be simplified down into the letters A, C, T or G, corresponding to each type of nucleotide, DNA is normally stored as a text file.

Types of Sequences files

DNA data comes in a variety of different formats but the main two are *FASTA* and *FASTQ*.

FASTA files

FASTA format files store data over two lines. The first always starts with a "`>`" symbol and provides some information about the DNA sequence that follows it. This may be the name of organism it comes from, how it was sequenced or how the DNA sequences was assembled. DNA sequences are typically composed of the letters `ATCG`, but other symbols are possible such as `N` for unknown.

Here is a typical example.

```
>Clibanarius_erythropus_Sample_Ke8_Assembly_consensus_sequence_for_COX1
GGTCAACAAATCATAAAGATATTGGCACTTTATATTTTATTTTTGGAGCTTGAGCTGGTATGGTAGGATCTTCACTTAGACTAGTTATTCGAGCCGAACTAG
GCCARCCCGGCAGTTTAATTGGAGATGACCAAATCTACAACGTAGTTGTCACAGCCCATGCTTTTGTAATAATTTTTTTCATRGTAATACCTATTATAATTG
```

```
GTGGKTTTGGTAATTGATTAGTTCCTTTAATATTAGGAGCYCCGGATATAGCTTTCCCACGAATAAACAACATAAGATTCTGATTACTRCCCCCGTCTCTCA
CTTTATTATTAATAAGAGGAATGGTTGAAAGAGGTGTCGGAACAGGATGAACTGTTTACCCTCCYYTGTCTGCTGCWATTGCCCACGCGGGTGCTTCGGTYG
ATCTGGGTATTTTTTCTTTACACTTAGCAGGAGTGTCCTCAATCTTAGGCGCCATCAATTTTATAACCACAGTGATCAATATACGGCCCCGTGGKATAAGAA
TAGACCGAATACCTTTATTCGTGTGRTCTATYTTTATTACCACTATTTTACTTTTATTRTCCCTACCTGTTTTAGCAGGYGCCATTACCATATTATTAACAG
ACCGAAACTTAAATACTTCTTTCTTTGACCCRGCTGGTGGAGGRGA-CCCTGTTTTATATCAACATTTATTTTGATTTTTTGGTCACCC
```

We have provided you with a FASTA sequence data file. You can view it using the `head` or `cat` command.

FASTQ files

FASTQ files contain the same information as FASTA files but with the added information of quality around the confidence of each nucleotide in the sequence. DNA Sequencing machines are never 100% accurate and the quality of the base call is preserved within the FASTQ file format. The first two lines of a fastq file are the same as fasta file, but there are two additional lines that contain the quality of each base call. This is displayed by a new line containing the symbol "+" and the next line contains coded information for the base call quality.

Here's an example:

```
@m64157e_211022_191939/7/ccs
ACATCAGCTTTAATCTTGGGTAGGACTAATGAAGCCTTTAGGTTTCCAAACATGGTTTATTCATTACTTAGAGCTGAG
+
cSFRTWRR+L?:INRNbBddZYBOSRXE_ASKaSH_5FLR@T8HV1R1XD@T4CY9HF=HbH*=\E%1UXRYRRVSX8
```

The first line contains the sequence identifier, in this case it corresponds to the name of a sequencing machine and then the specific sequencing run used (dont worry about that too much). The next line contains the DNA sequence. The third line always contains a "+". The forth line displays the encoded quality information. Each symbol corresponds to a numerical quality score from 0 to 93. Zero is complete rubbish we have no idea what that base could be really. 93 is great we are very confident that individual base has been called correctly.

The quality information is not provided as numbers so that each symbol on the forth line corresponds to the same base on the second. For instance the first base has been called as an "`A`" and has a quality score of "`c`". "`c`" is the code for "`66`". The final base in this sequence was called as a "`G`" and has a quality score of "`8`" which is code for "`23`".

Task

We have provided you with some sequence data. Use the `head` command to view the FASTQ directory. Can you spot each of the four components of the fastq file?

Downloading software

Today we are using two pieces of bioinformatics software, the first is a genome assembler called Hifiasm and the other is for assessing genome assemblies called Quast.

First make sure you're in the `genome_assembly` directory, `cd ..` to move you up a level and you can check which directory you're in by using `pwd` print working directory.

It's also good practice to use list to check you're in the right place

```
ls
```

Then use the `git clone` command to download the software from Github

```
git clone https://github.com/chhylp123/hifiasm
```

Change into the Hifiasm directory and unpack the software so it's ready to use. You can do this on a single line by typing in the following code. Afterwards use `cd ..` to move back into the `genome_assembly` directory.

```
cd hifiasm && make
```

Genome construction

We are going to go through some of the stages of genome assembly, assessment, and annotation. Due to the large computational tasks that genome assembly involves we have reduced the scope of the data we are using. The process is the same but the output will differ slightly from that of a whole genome assembly. A typical genome assembly can task a few hours to a few days to complete.

We are now going to construct our first genome. In the interest of time we are going to be assembling a small section of the genome of the rubyspot damselfly, *Hetaerina titia*, specifically this female *H titia*.

To construct our first genome we are going to use the programme *Hifiasm*. The code to use Hifisam is surprisingly simple.

```
hifiasm/hifiasm -o <OUTPUT_FILE_NAME> -t <NUM OF THREADS> </path/to/sequence_data.fastq>
```

Change each of the section in the code to the required input and output files You do not need to include the surrounding "<" and ">" marks. The input file is called `gbk.HiFiMapped.bam.fasta`. You can specify a file that is not in your current working directory by typing out the full path to the file (`My-first-Genome/gbk.HiFiMapped.bam.fasta`). You should select 1 core. You can call the output filename whatever you like, but remember what it is!

This will likely take a few minutes to run.

That's it! You've run your first genome assembly.

Hifiasm does not output a FASTA file. We can convert the `gfa` file created by Hifiasm to a FASTA file by using the following code. You will need to make sure you use the correct file name based of what you decided the output file was going to called.

```
awk '/^S/{print ">"$2;print $3}' My-genome-name.bp.p_ctg.gfa > My-genome-name.p_ctg.fasta
```

You can view the draft genome by using the code `cat /path/to/assembly/`. This likely prints out a LOT of ATCGs, to reduce the output we can use the `head` command again.

We can now find out some information about the assembled genome. Remember that each sequence of DNA in a FASTA format starts with a ">". We can use the `grep` and `wc` command from earlier to count the number of times ">" appears in input and output text file file. Try to compare how many sequences were in the raw sequence data and how many sequences are in the genome assembly.

Bonus tasks

Genome assessment

We are now going to use a tool called Quast. Quast is used to assess the quality of a genome assembly through the calculation of statistics such as N50 and L50.

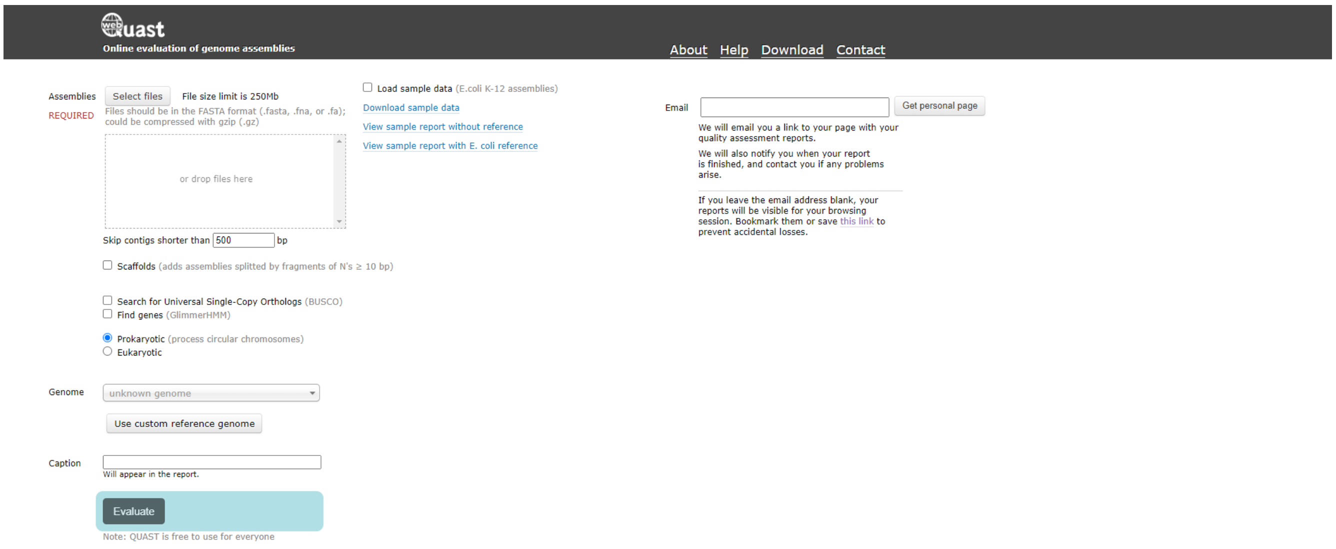
We have uploaded three draft genome from three different organisms. All three can be found and download from NCBI (National Center for Biotechnology Information).

- The red squirrel (*Sciurus vulgaris*) - CACRXJ02.1
- A carnivorous plant (*Genlisea aurea*) - GCA_000441915.1
- The buff-tailed bumblebee (*Bombus terrestris*) - GCF_910591885.1

We are going to run Quast assessments on all of these genome assemblies. The assemblies are stored as `.gz` files which means they are compressed to save memory. For instance, if we had a run of twelve A's (`AAAAAAAAAAAA`) we could store it as `AAAAAAAAAAAA` (12 bytes) but alternatively we could compress this information into `Ax12` (4 bytes) This isn't how .gz files compress data but does show that a typical genome can be easily compressed into a smaller file.

Today we are going to use the web version of Quast. To access this we use `firefox` followed by the link to Web Quast tool.

```
firefox https://www.ccb.uni-saarland.de/quast/
```



Assemblies REQUIRED
File size limit is 250Mb
Files should be in the FASTA format (.fasta, .fna, or .fa); could be compressed with gzip (.gz)
or drop files here
Skip contigs shorter than 500 bp
Scaffolds (adds assemblies splitted by fragments of N's ≥ 10 bp)
Search for Universal Single-Copy Orthologs (BUSCO)
Find genes (GlimmerHMM)
[✓ Prokaryotic] (process circular chromosomes)
Eukaryotic
Genome: unknown genome
Caption: Will appear in the report.
Note: QUAST is free to use for everyone
Load sample data (E.coli K-12 assemblies)
Download sample data
View sample report without reference
View sample report with E. coli reference
Email
We will email you a link to your page with your quality assessment reports.
We will also notify you when your report is finished, and contact you if any problems arise.
If you leave the email address blank, your reports will be visible for your browsing session. Bookmark them or save this link to prevent accidental losses.

The following scripts runs Quast on the three genomes. Because this code is quite long we are going to use `\` to allow us to type it out over multiple lines.

Click on 'select files' and open one (or all three) of the genomes within the `Draft_Genomes` folder. All of these genomes are Eukaryotic so remember to select 'Eukaryotic'. Then, click 'Evaluate'. This will take a few minutes.

Take you time to look over the report. What information can you extract?