Command line bioinformatic pipelines
Part 1 + 2

Mpox training workshop

August - September 2023

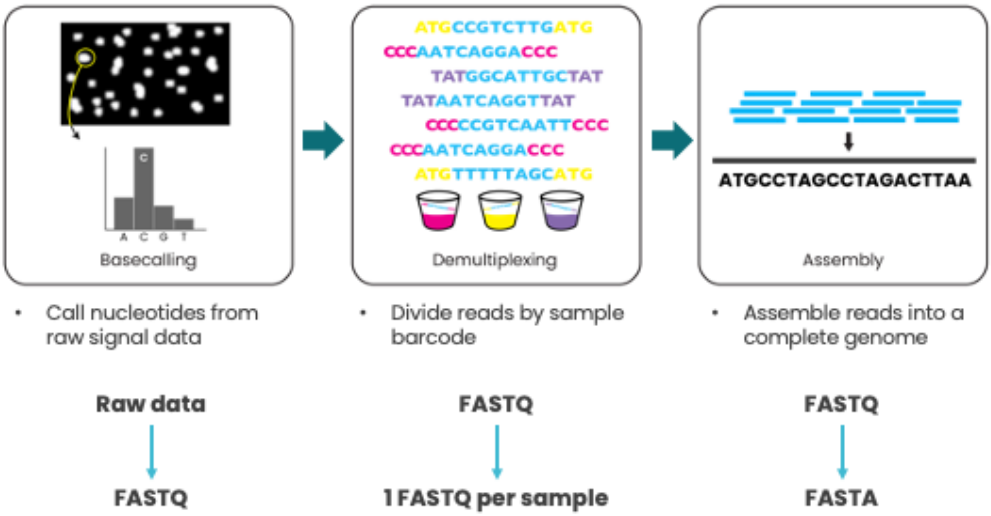
eparker@scripps.edu

As a slight preamble (South Africans love a preamble), these notes are still pretty rough – they're meant to guide us along, help us stay on the same page (quite literally) and serve as a reference when we return to our labs!

So, be kind about any spelling or formatting mistakes, and if you pick up any big, conceptual mistakes, email me at eparker@scripps.edu and I'll owe you a beer or cake.

1. Bioinformatic workflow overview

As a reminder (from the first week), this is a high-level breakdown of the key steps in our bioinformatic workflow:



Briefly: the raw read files off the sequencing machine (one big fastq file encompassing all reads, based on the base call image files) is demultiplexed based on the barcodes you added during library prep. Demultiplexing sorts all reads (in the one big fastq) based on the barcodes into fastq files per sample. Your sample sheet tells the sequencing machine (or your computer) what barcodes to look for and the sample name corresponding to each set of barcodes. Demultiplexing normally happens on the Illumina sequencing machine. We can then transfer the per-sample fastq files to our computers/servers for assembly. If we did paired end sequencing (which we did) we will have two fastq files – one with the forward reads and one with the reverse.

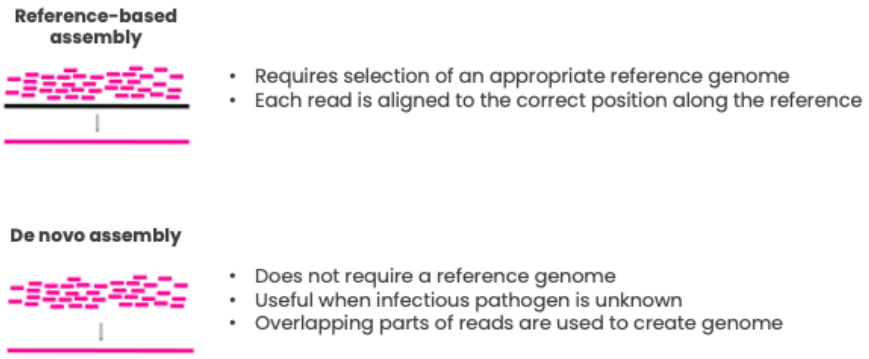
The demultiplexed raw reads (now two fastq files per sample – I'll call them R1 and R2 every now and again in this text) will have to be processed in order by a number of tools to generate our desired output, which here is a assembly or consensus sequence in the fasta format – and maybe a variant call file, especially if we're interested in within-host diversity.

The tools all chained together in order and run automatically is fundamentally our pipeline!

The tools we use and what steps we include in our pipeline varies across organisms and technologies and research questions, but a pretty generic pipeline would look a little like this:

Step	Read quality	Read filtering	Read mapping	Coverage	Consensus calling	Variant calling
Tool	Fastqc	Trimmomatic	Bwa + Samtools	Samtools	Samtools + iVar	Samtools + iVar
Input	R1.fastq R2.fastq	R1.fastq R2.fastq	R1_trim.fastq R2_trim.fastq	Bam	Bam	Bam + reference
Output	R1.html R2.html	R1_trim.fastq R2_trim.fastq	Bam	CSV	Fasta	VCF/ CSV

Now, this is a pipeline that does reference-based assembly, not *de novo* assembly. As a reminder:



Reference-based assembly is not appropriate when 1) we don't know the target organism, 2) if there is no decent reference sequence available or 3) if we suspect our pathogen might be very divergent from the reference. For *de novo* assembly, the steps excluding assembly will be similar – in the gray blockish figure, just swap out "Read mapping" with "De novo assembly" and the tools with a program like *Spades*.

It is also important to remember, we performed metagenomic sequencing, not targeted sequencing of Mpox, right? So, not every read in our fastqs is from the Mpox virus. – there will be host reads and reads from potential co-infections, the microbiome etc.

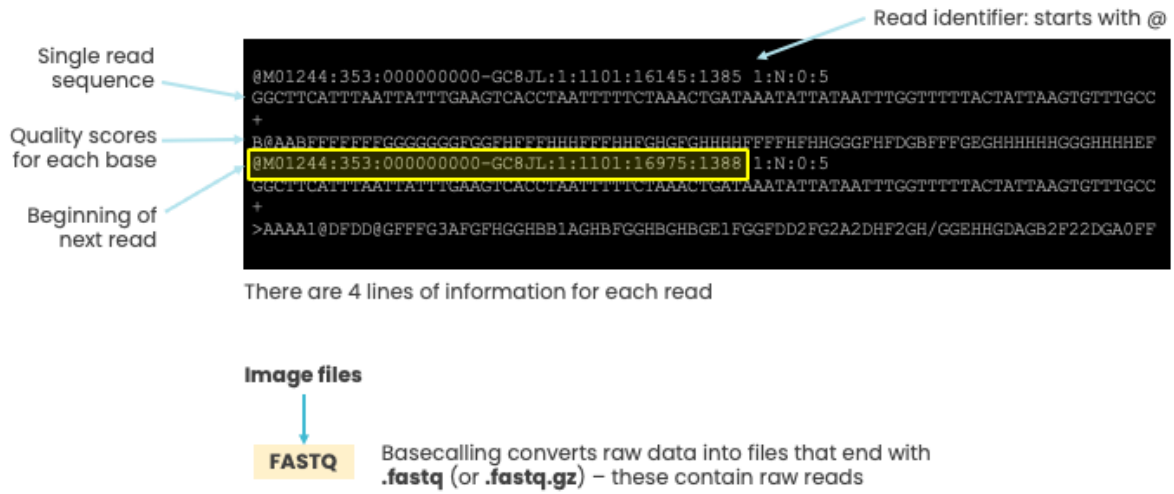
We will attempt to do both reference based and *de novo* assembly, don't worry! But we'll start with reference-based assembly, as it's slightly more clear conceptually. As we have metagenomic data, we can perform a step to deplete host (here human) reads as part of our pipeline too – as you did in Terra. As we're going to start with reference-based assembly, mapping to a Mpox reference, depletion of human reads is not necessary unless we're trying to save memory and space. When we are performing *de novo* assembly, it may be worth depleting human (our host) reads, before we spend all our computational time assembling human contigs! We'll get to this in later parts of this tut…

2. Our pipeline input: fastq files

As we can see from the gray figure above, the input into the first step (Read quality assessment) is R1.fastq and R2.fastq. This is the demultiplexed forward and reverse reads from one sample. The file names normally look a little like this: SampleID_R1_001.fastq and SampleID_R2_001.fastq, where SampleID is, well, you sample identifier.

Our read fastqs are a random sample from the data you generated (which I called S1), which is stored here: https://www.dropbox.com/sh/qmwz86ii8hld1ta/AABsx53lOlAzTGHpH4SRuWLNa?dl=0.

Let's briefly review what a fastq file looks like:



The quality score for each base is a prediction of the probability of an error in base calling. For base calls with a score of Q40, 1/10000 base calls is predicted to be incorrect. For base calls with a score of Q30, 1/1000 base calls is predicted to be incorrect. So, a higher score indicates that a base call is less likely to be incorrect!

You can read more on base quality scores here:
https://www.illumina.com/content/dam/illumina-marketing/documents/products/technotes/technote_understanding_quality_scores.pdf

And it's always helpful to remind ourselves how the sequencing technology (here Illumina) actually works: https://www.illumina.com/science/technology/next-generation-sequencing/sequencing-technology.html
Usually we won't open fastq files directly. If for some reason we need to assess if the file format is typical, we can look at them with command in our terminal. So, let's see if we can remember how to operate that…

For the purposes of this tutorial, anything in a gray box like the one below is a command you can type into the terminal directly. You can also copy and paste the text into your terminal. And if something is in italics, it's probably a command or a tool!

Importantly: your terminal should be open in the directory (folder) where the fastq files from this tutorial (that you downloaded from the Dropbox link) are saved! They are most likely in your Downloads directory. In Windows subsystems, this should be somewhere like

/mnt/c/Users/<username>/Downloads

Where <username> is the name of your account on your computer. If you're unsure what your username is, try typing the following into the terminal:

```
cd  /mnt/c/Users/
```

Remember how we navigated around the terminal in our last session? This will change directory (the command *cd*) to the folder /mnt/c/Users/**,** which is the home directory of Windows Linux subsystem. Now type:

```
ls
```

Which is the list command – we'll use *ls* a lot, so remember it – it lists all the files and subdirectories in a directory. Anything followed by a slash (/) is a directory!

Do you see your username in the folders and files listed? Note that down that!

So, to get to the downloads folder we type:

```
cd  /mnt/c/Users/YOURNAME/Downloads
```

Where YOURNAME is your username you just noted down. For example, if I were ona Windows subsystem I'd type:

```
cd  /mnt/c/Users/edythp/Downloads
```

Now type the list command again and check if our read files are there!

```
ls
```

If you're on a Mac or a Linux machine, you're in luck! You can just type:

```
cd ~/Downloads
```

This will change directory (the command cd) to the Downloads directory (or folder), which is saved in your home directory – on linux/unix systems, the home directory can be accessed by the shortcut "~". Again, check if our read files are there by the list command!

We probably don't want to just work in our Downloads folder. You may want to create a new folder (directory) to work in. We're going to use the *mkdir* command to make a new directory.

```
mkdir Mpox_workshop
```

Very creatively, Mk= Make, dir= directory. So the *mkdir* command will make a new directory named Mpox_workshop. This is basically a new folder, like we're used to creating in Windows or in Mac's Finder.

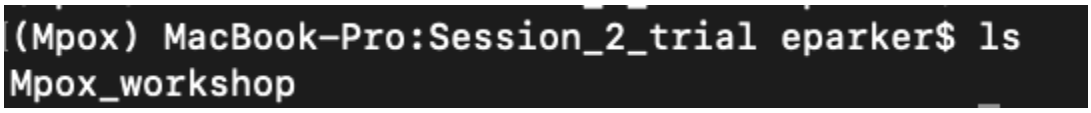
Now, we need to move our fastq files to the new directory named Mpox_workshop. We're going to use the similarly creatively named *mv* command. *Mv* = Move.

```
mv S1_R1_001.fastq.gz Mpox_workshop
```

```
mv S1_R2_001.fastq.gz Mpox_workshop
```

In this command, you can see we use *mv* to move the file S1_R1_001.fastq.gz to the folder Mpox_workshop, and then S1_R2_001.fastq.gz to Mpox_workshop too.

If we type *ls*, we now see our files have moved, and we only have the Mpox_workshop folder*:*



In these screenshots, don't worry about anything that comes in front of the $, it's just details about my folder structure, so it'll be different from yours. The dollar sign $ is the prompt – anything after the $ is the command, and everything below it is the output of e.g. the *ls* **command. So e.g. we ask the terminal to list everything in the directory, and we see there's now just the** Mpox_workshop **directory we just created!**

Now, if we want to work with our fastqs, we need to be in the Mpox_workshop directory, right? So we're going to change directory to the Mpox_workshop directory:

```
cd Mpox_workshop
```

And then *ls* again:

```
ls
```

Our output should look like:

Now we're in the right folder with our fastqs! Let's get back to checking our fastq file format! But notably, our fastq files end in ".gz", right? This means they are compressed! We first need to uncompress them. We compress them as they are rather large files – and many bioinformatic tools can actually work with compressed fastqs! But let's uncompress (unzip, decompress, pick your poison etc) them for now.

We'll use a command called *gunzip* – this should work for Linux and MacOs. For Windows subsystem (WSL), it seems like gzip would work (https://learnubuntu.com/unzip-gz-files/) but if not, try unzipping it in the graphical user interface – or flag down an instructor (shout at me too)!

We want to uncompress both fastqs, right? So we can either run (don't do this for now!):

```
gunzip S1_R1_001.fastq.gz
```

```
gunzip S1_R2_001.fastq.gz
```

And then:

```
ls
```

To get:

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ gunzip S1_R1_001.fastq.gz
(Mpox) MacBook-Pro:Mpox_workshop eparker$ gunzip S1_R2_001.fastq.gz
(Mpox) MacBook-Pro:Mpox_workshop eparker$ ls
S1_R1_001.fastq S1_R2_001.fastq
```

Or we can be professionally and efficiently lazy and start using wildcards!

Here is a good introduction to wildcards: https://support.microsoft.com/en-gb/office/examples-of-wildcard-characters-939e153f-bd30-47e4-a763-61897c87b3f4

Briefly, a wild card is a character that can act as a placeholder for a bunch of characters, matching a specified pattern. The "*" is a wild card that will match any character (letter, number) and any number of characters.

So, let's run:

```
gunzip *.gz
```

So "*.gz" will basically find any file that ends in .gz – which is both of our compressed fastqs, right? So it'll match S1_R1_001.fastq.gz and S1_R2_001.fastq.gz, but could even match THISFILE.gz and YAYAYAYA123.gz – because the wildcard "*" captures all characters before the ".gz" in the file name.

So it'll find both compressed fastqs and unzip them respectively!

Our output should look like:

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ gunzip *.gz
(Mpox) MacBook-Pro:Mpox_workshop eparker$ ls
S1_R1_001.fastq S1_R2_001.fastq
```

Now we're ready to view these uncompressed fastqs – again, not a routine step in any pipeline, but a helpful exercise in getting used to terminal operations!

We will use the *head* command, which prints out the top lines of the file we specify:

```
head -n 4 S1_R1_001.fastq
```

The *head* command takes a few arguments – arguments are the specifics or parameters we give to a command. They normally follow the command and may start with dashes like "-" or "--", often referred to as flags.

Here we have:
"-n 4" – this argument tells the *head* command to print the first 4 lines of the file to the terminal/screen.

This is followed by the file name we want the *head* command to operate on: "S1_R1_001.fastq" – the forward reads of our Sample 1.

This output should look like:

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ head -n 4 S1_R1_001.fastq
@HG5N3DMXY:1:1101:10104:25347/1
GTATTTGTGGGAGTATTATGGAGTAGAAACAAAAACAGACGCTGGTGCGGCAACATATGTGCTTCCTCA
ATCCATGGTATTCGAATATAGAGCGAGTACAA
+
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

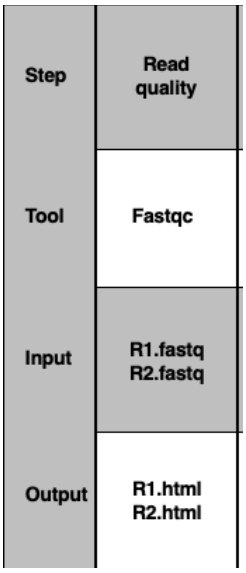
Again, no need to look at fastqs directly as part of your routine (unless a tool tells you there's something wrong with the file) – just want us to get comfortable in the terminal!

3.Read quality assessment with *FastQC*

In the first real step in the pipeline, we are going to use a program called *FastQC* to assess the quality of our reads! *FastQC* is going to read in the fastqs and take a look at the length of the reads, their associated base quality scores etc and help us understand the quality of our data.

As a reminder: our input here is our raw, demultiplexed fastqs – R1, forward reads, and R2, reverse reads, as we did paired-end sequencing (As a reminder: https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/paired-end-vs-single-read.html#:~:text=Unlike%20single%2Dread%20seqeuncing%2C%20paired,gene%20fusions%20and%20novel%20transcripts)

Here is our first step in the pipeline, in terms of input, output and tools:



We're going to run *FastQC* from the terminal, so we should have installed it previously. We can easily install *FastQC* with Anaconda or Miniconda or Mamba (all package managers).

We'll call Anaconda/Miniconda using the *conda* command.

Let's see if our Anaconda package manager is installed properly. Let's type:

```
conda –version
```

This should print out the version number of Anaconda if it's installed correctly, e.g:

```
conda 23.3.1
```

We can also try

```
which conda
```

That command should tell us where the *conda* command is installed in our computer – what does your output say?

It'll be different from mine, of course – you have a different system, username and I'm using the Mamba package manager (hence /mambaforge/).

```
(base) MacBook-Pro:Mpox_workshop eparker$ which conda
/Users/eparker/mambaforge/bin/conda
```

We call that the "path" (I'll use PATH), remember?

"/Users/eparker/mambaforge/bin/" is the path where my tool *conda* is installed. It's basically the address of the program – where the terminal needs to go look if it wants to run the tools. Anaconda or package managers (like Mamba) are great because they install things in a PATH where the terminal can always find them – so we can just run *conda* without telling it the path.

What does it mean if *which conda* returns no output?

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ which conda
(Mpox) MacBook-Pro:Mpox_workshop eparker$
```

It means my terminal cannot find the command – it's not in the PATH! It can still be installed, the terminal just doesn't know where to look for it… We can always specify the PATH before we call the tool, e.g.:

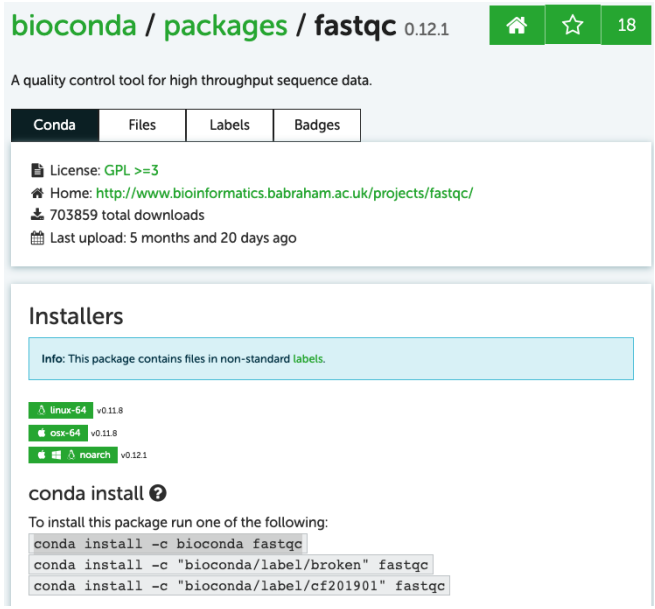
```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ /Users/eparker/mambaforge/bin/conda --version
```

But for now, our installs with Anaconda should automatically be placed in the PATH – shout if you run into problems! We'll look at exporting to the PATH a bit later on.

Let's install *FastQC* with Anaconda.

```
conda install -c bioconda fastqc
```

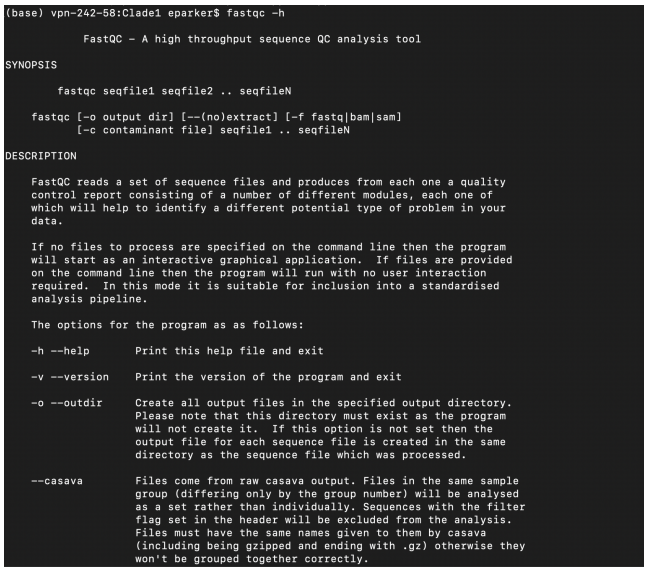
As a reminder, we call Anaconda with the *conda* command, and tell it we want to "install" "fastqc" from the channel bioconda (the "-c bioconda" basically tell Anaconda what aisle of the grocery store fastqc is in – aka where to go look to download it). If you ever want to download a bioinformatic tool, just google "Anaconda install " and the name of the tool and Anaconda will tell you the command e.g: https://anaconda.org/bioconda/fastqc



Let's see if *FastQC* installed correctly:

```
fastqc -h
```

The "-h" (-help) is a flag to the command *fastqc* that will print out the manual on how to use *fastqc*. You should see something like the figure below:



This help page should help us understand what the tool does, but also what format the command needs to be in, what input it requires and what the output looks like. The “-help” argument should work for most tools run in the command line. So if you’re unsure what a tool needs as input, or how you can change the parameters of the tool, look at the help function by typing the tool name and the argument “-h” or “--h”.

Let’s use *FastQC* to visually assess the quality of our reads, and run some diagnostic metrics! We should have already encountered the metrics in our “Evaluating assemblies” session in week 1.

Let’s run *FastQC* on our first fastq:

```
fastqc S1_R1_001.fastq
```

Our output might look a little like:

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ fastqc S1_R1_001.fastq
Started analysis of S1_R1_001.fastq
Approx 5% complete for S1_R1_001.fastq
Approx 10% complete for S1_R1_001.fastq
Approx 15% complete for S1_R1_001.fastq
Approx 20% complete for S1_R1_001.fastq
Approx 25% complete for S1_R1_001.fastq
Approx 30% complete for S1_R1_001.fastq
Approx 35% complete for S1_R1_001.fastq
Approx 40% complete for S1_R1_001.fastq
Approx 45% complete for S1_R1_001.fastq
Approx 50% complete for S1_R1_001.fastq
Approx 55% complete for S1_R1_001.fastq
Approx 60% complete for S1_R1_001.fastq
Approx 65% complete for S1_R1_001.fastq
Approx 70% complete for S1_R1_001.fastq
Approx 75% complete for S1_R1_001.fastq
Approx 80% complete for S1_R1_001.fastq
Approx 85% complete for S1_R1_001.fastq
Approx 90% complete for S1_R1_001.fastq
Approx 95% complete for S1_R1_001.fastq
Analysis complete for S1_R1_001.fastq
```

Let's use the *ls* command

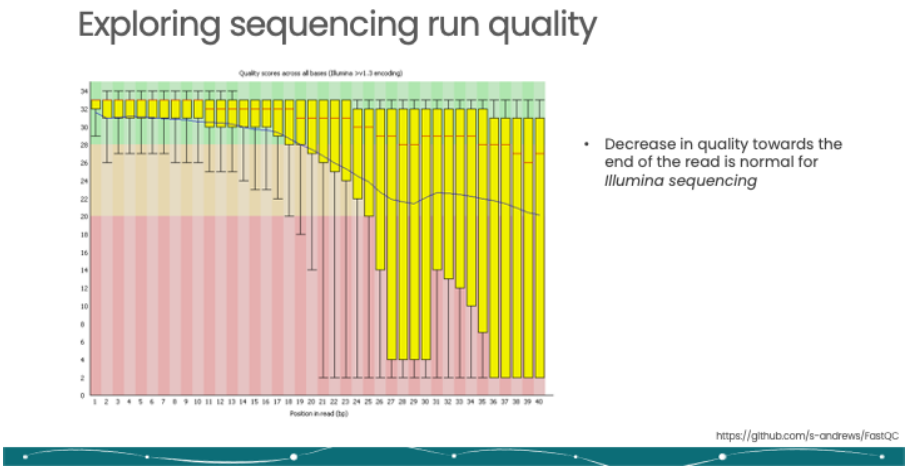
```
ls
```

We can see that *FastQC* has produced a html file and a zipped directory (.zip).

```
(base) vpn-242-58:Session_2 eparker$ ls
S1_R1_001.fastq  S1_R1_001_fastqc.html  S1_R1_001_fastqc.zip  S1_R2_001.fastq
```

We can open the html file in any web browser. The zipped directory basically just has a compressed set of the output files.

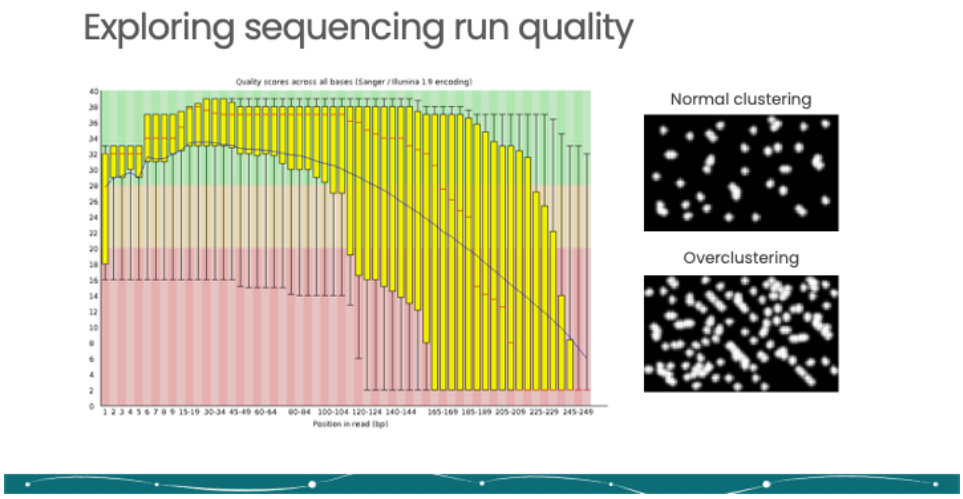
You'll already be familiar with some of these diagnostic metrics from the first week.



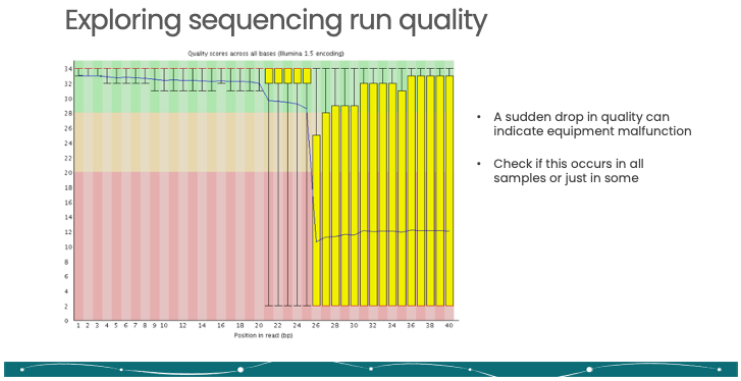
We've already seen these plots of the median and interquartile range of base quality scores (on the y-axis) across all positions in the read (on the x-axis). This score reflects

the average across all reads in the fastq. It can help us see if something went wrong in the sequencing run. The higher the score on the y-axis, the better the base call. The green zone indicates good quality calls, orange zone indicates reasonable quality calls and the red indicates poor quality base calls. For most sequencing technologies, the quality will drop off towards the end of the read as observed in the figure – this is normal, especially for Illumina data!

However, if the drop off in quality is not just in the last few bases, this might be indicative of overclustering. Overclustering makes it difficult to distinguish the spectra of the different base calls. This would require resequencing, balancing sequencing libraries etc (consult a wet lab biologist!). E.g:



If there's a complete drop off in base quality, something very likely went wrong with the sequencer, e.g:



There are also several other statistics for us to look at. For the sake of time, we won't look into all of them, but here is a good manual to understand what they all reflect: https://dnacore.missouri.edu/PDF/FastQC_Manual.pdf

Basically the statistics reflect:

- **Per tile sequence quality:** the machines that perform sequencing are divided into tiles. This plot displays patterns in base quality along these tiles. Consistently low scores are often found around the edges, but hot spots can also occur in the middle if an air bubble was introduced at some point during the run.
- **Per sequence quality scores:** a density plot of quality for all reads at all positions. This plot shows what quality scores are most common.
- **Per base sequence content:** plots the proportion of each base position over all of the reads. Typically, we expect to see each base roughly 25% of the time at each position, but this often fails at the beginning or end of the read due to quality or adapter content.
- **Per sequence GC content:** a density plot of average GC content in each of the reads.
- **Per base N content:** the percent of times that 'N' occurs at a position in all reads. If there is an increase at a particular position, this might indicate that something went wrong during sequencing.
- **Sequence Length Distribution:** the distribution of sequence lengths of all reads in the file. If the data is raw, there is often on sharp peak, however if the reads have been trimmed, there may be a distribution of shorter lengths.
- **Sequence Duplication Levels:** A distribution of duplicated sequences. In sequencing, we expect most reads to only occur once. If some sequences are occurring more than once, it might indicate enrichment bias (e.g. from PCR). If the samples are high coverage (or RNA-seq or amplicon), this might not be true.
- **Overrepresented sequences:** A list of sequences that occur more frequently than would be expected by chance.
- **Adapter Content:** a graph indicating where adapter sequences occur in the reads.
- **K-mer Content:** a graph showing any sequences which may show a positional bias within the reads.

Now, let's run *FastQC* on our R2 or reverse reads.

```
fastqc S1_R2_001.fastq
```

We should now have a second html and zipped directory corresponding to the output for the second fastq.

```
ls
```

```
(base) vpn-242-58:Session_2 eparker$ ls
S1_R1_001.fastq  S1_R1_001_fastqc.html  S1_R1_001_fastqc.zip  S1_R2_001.fastq  S1_R2_001_fastqc.html  S1_R2_001_fastqc.zip
```

We might want to structure our output into new folders inside our folder (subdirectories inside our directory), so that our output is more organized.

We're going to use the *mkdir* command to make a new directory.

```
mkdir fastqc_report
```

If we use *ls*, we can see that we created the subdirectory (subfolder) fastqc_report in our directory (folder).

```
ls
```

```
(base) vpn-242-58:Session_2 eparker$ mkdir fastqc_report
(base) vpn-242-58:Session_2 eparker$ ls
S1_R1_001.fastq  S1_R1_001_fastqc.html  S1_R1_001_fastqc.zip  S1_R2_001.fastq  S1_R2_001_fastqc.html  S1_R2_001_fastqc.zip  fastqc_report
```

Now, we need to move our html and zip files to the new directory.

```
mv S1_R1_001_fastqc.html fastqc_report
```

In this command, you can see we use *mv* to move the file S1_R1_001_fastqc.html to the folder fastqc_report. We can do the same for the zipped directory.

```
mv S1_R1_001_fastqc.zip fastqc_report
```

Now we can go to the folder fastqc_report and check if the files were moved there. Let's change to the directory fastqc_report.

```
cd fastqc_report
```

Now let's *ls.*

```
ls
```

```
(base) vpn-242-58:Session_2 eparker$ cd fastqc_report
(base) vpn-242-58:fastqc_report eparker$ ls
S1_R1_001_fastqc.html  S1_R1_001_fastqc.zip
```

Good! Our files are in the right place! Now, we need to go back to the original folder. Fastqc_report is a subdirectory – we need to be in the Mpox_workshop directory with our fastqs (or wherever you saved them!).

We can go up one directory by:

```
cd ../
```

Cd = Change directory. The "../" means the directory above this one – Remember "/" indicates a directory (folder).

So we should now be in the right place. We can check with *ls* or *pwd.*

We can either move the quality reports of the reverse reads with a similar command:

```
mv S1_R2_001_fastqc.html fastqc_report
```

```
mv S1_R2_001_fastqc.zip fastqc_report
```

Or: Remember wildcards? Let's try typing:

```
mv *.html fastqc_report
```

The “*” is a wild card that will match any character (letter, number) and any number of characters. So “*.html” will basically find any file that ends in .html. So it’ll match S1_R2_001_fastqc.html and S1_R1_001_fastqc.html, but could even match HIERSO.html and BAM7BAM.html – because “*” captures all characters before the “.html”.

So instead of running:

```
mv S1_R1_001_fastqc.html fastqc_report
```

And

```
mv S1_R2_001_fastqc.html fastqc_report
```

We can run

```
mv *.html fastqc_report
```

And it would move both htmls (but one of them is already moved, right?).

Same for the zipped files.

```
mv *.zip fastqc_report
```

Get familiar with wildcards – you will use them a lot in command line bioinformatics and coding in general!

Now we have a directory with quality assessment files for our raw reads. If something goes wrong with our genome assembly, we can return to these reports and troubleshoot! It’ll also help us understand the type of read filtering we need to do to get rid of the poor quality reads, and if we might need to resequence.

But remember: we performed metagenomic sequencing! So even if there’s 50 million reads in these raw fastqs, it does not mean there’s enough reads to assemble a high quality Mpox genome…

4.Read filtering with *Trimmomatic*

We need to filter out reads that are too short or of poor quality to reduce the number of errors introduced into our assembly/consensus sequence and/or variant calls in downstream processes. There are numerous tools to filter poor quality reads out of fastq files, but we'll focus on using a tool called *Trimmomatic*.

As a reminder: our input into this second step in our pipeline is our raw fastqs, R1 and R2!

Step	Read quality	Read filtering
Tool	Fastqc	Trimmomatic
Input	R1.fastq R2.fastq	R1.fastq R2.fastq
Output	R1.html R2.html	R1_trim.fastq R2_trim.fastq

We should have installed *Trimmomatic* with Anaconda/Miniconda at this point:

```
conda install -c bioconda trimmomatic
```

Let's check our install with:

```
trimmomatic --help
```

Our output should look a little like:

```
(base) vpn-242-58:fastqc_report eparker$ trimmomatic --help
Usage:
       PE [-version] [-threads <threads>] [-phred33|-phred64] [-trimlog <trimLogFile>]
 [-summary <statsSummaryFile>] [-quiet] [-validatePairs] [-basein <inputBase> | <input
File1> <inputFile2>] [-baseout <outputBase> | <outputFile1P> <outputFile1U> <outputFil
e2P> <outputFile2U>] <trimmer1>...
   or:
       SE [-version] [-threads <threads>] [-phred33|-phred64] [-trimlog <trimLogFile>]
 [-summary <statsSummaryFile>] [-quiet] <inputFile> <outputFile> <trimmer1>...
   or:
       -version
```

Here's the *Trimmomatic* manual: http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/TrimmomaticManual_V0.32.pdf

As a rule, you should always read the manual of the tools you use – do not just black box the arguments or parameters of others/the default as they may not be appropriate for your pathogen or research question!

We will run *Trimmomatic* in the PE mode – PE here stands for paired ends (we did paired-end sequencing, right?). Based on the figure above (the –help function), we can see that the *Trimmomatic* command requires that we specify quite a few arguments.

Let's take a look at them:

We need to give *Trimmomatic* our input – our fastqs! We need to be very sure of the order in which we list the files. We can get this information from the help function above!

Trimmomatic wants us to list the input first:

```
trimmomatic PE S1_R1_001.fastq S1_R2_001.fastq
```

After the input fastq files, we list the names we want the new output files saved as. Our output here is new, trimmed fastq files, which have all the poor reads filtered out! Notably, there's two sets of output reads and for each we have a forward and reverse fastq (so four output fastq files in general).

- We have new output read files (R1 and R2) that contain only the paired reads aka all the read pairs that remain paired after filtering out the poor quality reads.
- We also have new output read files (R1 and R2) that have reads that are unpaired after read filtering, i.e either the forward or reverse read was filtered out.

So we need to give *Trimmomatic* four names to call our output eg:

```
trimmomatic PE S1_R1_001.fastq S1_R2_001.fastq S1_R1_001_P.fastq
S1_R2_001_UP.fastq  S1_R2_001_P.fastq S1_R2_001_UP.fastq
```

Here S1_R1_001_P.fastq and S1_R2_001_P.fastq are the new output files with paired forward and reverse reads (hence the "P"). S1_R1_001_U.fastq and S1_R2_001_U.fastq are the new output files with unpaired reads ("U").

We normally like to work with paired reads to prevent artifacts from sequencing errors being introduced. If we see a variant on a forward read that is not present on its paired reverse read, then it's likely not a true variant i.e. its a sequencing error. If we work with unpaired or single-end reads, we don't have that information to reduce sequencing errors.

So, we will use the paired reads fastq going forward. However, if we lose a lot of reads at this filtering step and we are struggling to assemble a quality full length genome, we might come back to the unpaired reads (but we try to avoid this to reduce errors!).

Trimmomatic requires us to input a few other parameters outside of the input/output files:

- **Threads** (optional argument): this is the number of processors on our computer that we allow *Trimmomatic* to use. A few tools will take a "threads" argument, so remember that! The number of threads you specify is dependent on your machine – how many processors you have. Using more processors typically speeds things up (with some caveats), but it's not necessary that you specify this argument always.

And then there's the actual filtering arguments and their parameters. Here's a few ways we can trim and filter reads:

- **ILLUMINACLIP**: Perform adapter removal.
- **SLIDINGWINDOW**: Perform 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 an entire read if it is below a specified length.
- **TOPHRED33**: Convert quality scores to Phred-33.
- **TOPHRED64**: Convert quality scores to Phred-64.

There is no one universal rule as to what filtering parameters fits all datasets and pathogens, so it's a good idea to understand how these parameters can be adapted (see *Trimmomatic* manual!).

For today's purpose, we're going to use the **threads**, **ILLUMINACLIP** and **SLIDINGWINDOW** options**.**

Let's type (all on one line – or copy and paste – but it's all one line!):

```
trimmomatic PE -threads 3 S1_R1_001.fastq S1_R2_001.fastq S1_R1_001_P.fastq
S1_R1_001_UP.fastq S1_R2_001_P.fastq S1_R2_001_UP.fastq
ILLUMINACLIP:Nextera_transpose.fasta:2:30:10 SLIDINGWINDOW:4:20 MINLEN:50
```

Apart from the input and output files we discussed above we have:

- "--thread 3"
    - Tells *Trimmomatic* to use three processors
- ILLUMINACLIP:Nextera_transpose.fa:2:30:10
    - This step clips off the Illumina adapters, if we haven't already automatically
    - We need to give *trimmomatic* the input fasta file Nextera_transpose.fa, which has the sequences of the Nextera adapters in it.
    - Nextera_transpose.fa should be in the dropbox folder too with the reads (in the directory Download_me_too)
    - We can also check the *FastQC* output to see if we still have adapters in our sequencing data, right?
    - The numbers at the end of the argument (e..g the 3:30:10) tells *Trimmomatic* how to handle sequence matches to the adapters
- SLIDINGWINDOW:4:20
    - This step uses a sliding window of size 4 to remove bases if their quality score is below 20
- MINLEN:50
    - This step drops all reads below 50nt

Our output should look a little like:

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ trimmomatic PE -threads 3 S1_R1_001.fastq
S1_R2_001.fastq S1_R1_001_P.fastq S1_R1_001_UP.fastq S1_R2_001_P.fastq S1_R2_001_UP.
fastq ILLUMINACLIP:Nextera_transpose.fasta:2:30:10 SLIDINGWINDOW:4:20 MINLEN:50
TrimmomaticPE: Started with arguments:
 -threads 3 S1_R1_001.fastq S1_R2_001.fastq S1_R1_001_P.fastq S1_R1_001_UP.fastq S1_
R2_001_P.fastq S1_R2_001_UP.fastq ILLUMINACLIP:Nextera_transpose.fasta:2:30:10 SLIDI
NGWINDOW:4:20 MINLEN:50
Using PrefixPair: 'AGATGTGTATAAGAGACAG' and 'AGATGTGTATAAGAGACAG'
Using PrefixPair: 'TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG' and 'GTCTCGTGGGCTCGGAGATGTGTAT
AAGAGACAG'
Using Long Clipping Sequence: 'GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTCCGAGCCCACGAGAC'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTGACGCTGCCGACGA'
ILLUMINACLIP: Using 2 prefix pairs, 4 forward/reverse sequences, 0 forward only sequ
ences, 0 reverse only sequences
Quality encoding detected as phred33
Input Read Pairs: 445904 Both Surviving: 416683 (93.45%) Forward Only Surviving: 200
48 (4.50%) Reverse Only Surviving: 4874 (1.09%) Dropped: 4299 (0.96%)
TrimmomaticPE: Completed successfully
```

We can see we started with 445 904 read pairs, and we didn't lose too many read pairs along the way!

What if we wanted to save this text from the screen, so we can go back to see how many read pairs we lost? We can redirect the output printed to the screen to a file. So, instead of printing it to the terminal screen, it'll save it in a text file.

To redirect the screen output, we'll run (this is all one line):

```
trimmomatic PE -threads 3 S1_R1_001.fastq S1_R2_001.fastq S1_R1_001_P.fastq
S1_R1_001_UP.fastq S1_R2_001_P.fastq S1_R2_001_UP.fastq
ILLUMINACLIP:Nextera_transpose.fasta:2:30:10 SLIDINGWINDOW:4:20 MINLEN:50
&> S1_trimsummary.txt
```

We've added "&> S1_trimsummary.txt" to the command. "&>" is the argument to redirect the screen output and "S1_trimsummary.txt" is the new file we're creating with the screen output printed to it.

If we use *ls* we should find the file S1_trimsummary.txt.

```
ls
```

We can open S1_trimsummary.txt in a text editor – on Windows, this will probably be Notebook or Notebook+. For Mac, I recommend you download Sublime Text!

But we can also open the file in the command line by using the *cat* command. *Cat* = concatenate – It's a bit less intuitive than *mv* or *ls*, but *cat* basically prints a file to your screen/terminal.

So let's try

```
cat S1_trimsummary.txt
```

Our output should look like:

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ cat S1_trimsummary.txt
TrimmomaticPE: Started with arguments:
 -threads 3 S1_R1_001.fastq S1_R2_001.fastq S1_R1_001_P.fastq S1_R1_001_UP.fastq S1_
R2_001_P.fastq S1_R2_001_UP.fastq ILLUMINACLIP:Nextera_transpose.fasta:2:30:10 SLIDI
NGWINDOW:4:20 MINLEN:50
Using PrefixPair: 'AGATGTGTATAAGAGACAG' and 'AGATGTGTATAAGAGACAG'
Using PrefixPair: 'TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG' and 'GTCTCGTGGGCTCGGAGATGTGTAT
AAGAGACAG'
Using Long Clipping Sequence: 'GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTCCGAGCCCACGAGAC'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTGACGCTGCCGACGA'
ILLUMINACLIP: Using 2 prefix pairs, 4 forward/reverse sequences, 0 forward only sequ
ences, 0 reverse only sequences
Quality encoding detected as phred33
Input Read Pairs: 445904 Both Surviving: 416683 (93.45%) Forward Only Surviving: 200
48 (4.50%) Reverse Only Surviving: 4874 (1.09%) Dropped: 4299 (0.96%)
TrimmomaticPE: Completed successfully
```

You can see this is the same as what was printed to your terminal, but now you have a saved file for future reference.

We should probably clean our folder up again, right?

Let's make a few new directories to move our output to. We're also moving forward with the new trimmed, paired fastqs, so maybe we can compress the original read files and the unpaired read files to save storage space (these files can be very big!) and then we can move the compressed fastqs to a new folder?

Let's try:

```
gzip *_UP.fastq
```

The *gzip* command compressed the unpaired reads fastq files to save storage space. Remember, the wildcard "*" should capture all files that end with "_UP.fastq" – so it'll compress S1_R1_001_UP.fastq and S1_R2_001_UP.fastq. You can double check with *ls.* Note: this command may take a second if your files are very large!

We can also compress our original untrimmed fastqs:

```
gzip *_001.fastq
```

If we tryL

```
ls
```

We see

```
(base) vpn-242-58:Session_2 eparker$ ls
Nextera_transpose.fasta  S1_R1_001_UP.fastq    S1_R2_001_UP.fastq
S1_R1_001.fastq.gz       S1_R2_001.fastq.gz    S1_trimsummary.txt
S1_R1_001_P.fastq.gz     S1_R2_001_P.fastq.gz  fastqc_report
```

We can see we now have two sets of compressed fastqs (with the file extension .gz indicating they're compressed!). If you ever need to uncompress these files, you can do so with:

```
gunzip *.fastq.gz
```

Let's make a new directory to move them to for more permanent storage. Note, it's not necessary to keep the unpaired reads around *per se*, as we're going to use the paired reads for assembly. You can always rerun the commands in the tutorial above to generate them from the raw fastqs, if you need to delete them now to save space.

Let's make a directory:

```
mkdir fastqs
```

And move our compressed fastqs there:

```
mv *.gz fastqs
```

Remember, this will move all files that end in ".gz" to the directory named fastqs/ – so let's be sure we're only moving the files we want to.

```
cd fastqs
```

```
ls
```

```
cd ../
```

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ cd fastqs
(Mpox) MacBook-Pro:fastqs eparker$ ls
S1_R1_001.fastq.gz      S1_R2_001.fastq.gz
S1_R1_001_UP.fastq.gz   S1_R2_001_UP.fastq.gz
(Mpox) MacBook-Pro:fastqs eparker$ cd ../
(Mpox) MacBook-Pro:Mpox_workshop eparker$
```

What does "cd ../" do? It moves us up to our original directory Mpox_workshop, right?

What files do we have left in our Mpox_workshop directory?

```
ls
```

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ ls
Nextera_transpose.fasta S1_R2_001_P.fastq       fastqc_report
S1_R1_001_P.fastq       S1_trimsummary.txt      fastqs
```

Maybe we want to move the *Trimmomatic* summary and the adapter fasta file to a different directory, as we don't need them any more. This is not necessary, but it's good practice to have a good directory structure!

Let's try:

```
mkdir intermediates
```

```
mv Nextera_transpose.fasta intermediates
```

```
mv S1_trimsummary.txt intermediates
```

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ ls
S1_R1_001_P.fastq       fastqc_report           intermediates
S1_R2_001_P.fastq       fastqs
```

Now in our directory, we have three subdirectories and our two fastq files. The two fastqs now represent the filtered, paired forward and reverse reads respectively – these are ready to be mapped to our reference!

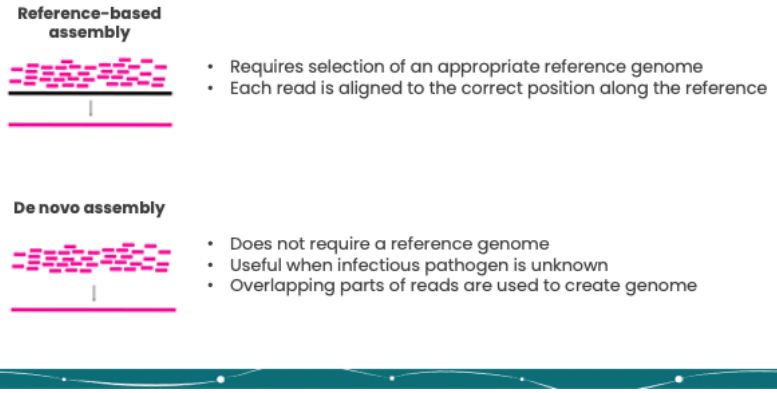
If you want to, you can run *FastQC* on these fastq files again and see how the quality metrics have changed after read filtering. (We'll skip that for now, but not a bad exercise when you're playing around with things on your own).

As a reminder: our input was our raw fastq files, and our output is our trimmed, paired fastqs!

Step	Read quality	Read filtering
Tool	Fastqc	Trimmomatic
Input	R1.fastq R2.fastq	R1.fastq R2.fastq
Output	R1.html R2.html	R1_trim.fastq R2_trim.fastq

5.Read mapping with *bwa*

Our next step is to map our reads (paired and trimmed, forward and reverse fastqs) to our reference genome to see which position of the genome they align to. Remember, we have two types of assemblies:



We'll attempt a reference based assembly using a reference genome isolated from a Mpox case in Nigeria in 2018 (Genbank: NC_063383). Let's download the reference genome from NCBI (It should also be in the Dropbox). Note: we need to save the reference in the same folder where our fastqs are i.e. Mpox_workshop.

Monkeypox virus, complete genome

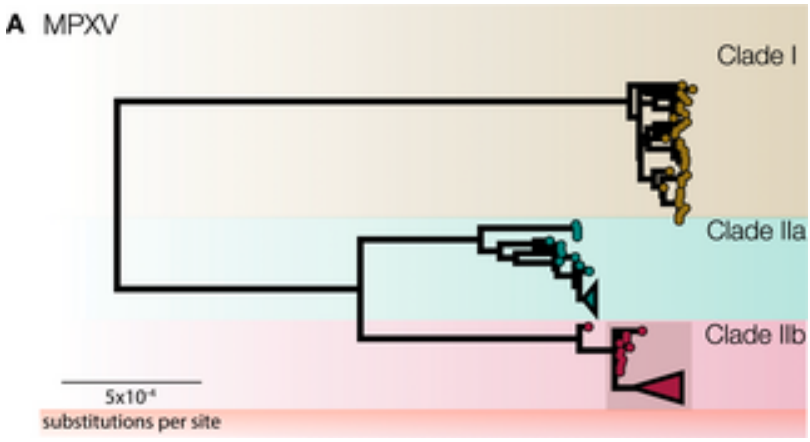
NCBI Reference Sequence: NC_063383.1

FASTA Graphics

```
LOCUS       NC_063383             197209 bp    DNA     linear   VRL 18-NOV-2022
DEFINITION  Monkeypox virus, complete genome.
ACCESSION   NC_063383
VERSION     NC_063383.1
DBLINK      BioProject: PRJNA485481
KEYWORDS    RefSeq.
SOURCE      Monkeypox virus
  ORGANISM  Monkeypox virus
            Viruses; Varidnaviria; Bamfordvirae; Nucleocytoviricota;
            Pokkesviricetes; Chitovirales; Poxviridae; Chordopoxvirinae;
            Orthopoxvirus.
REFERENCE   1  (bases 1 to 197209)
  AUTHORS   Mauldin,M.R., McCollum,A.M., Nakazawa,Y.J., Mandra,A.,
            Whitehouse,E.R., Davidson,W., Zhao,H., Gao,J., Li,Y., Doty,J.,
            Yinka-Ogunleye,A., Akinpelu,A., Aruna,O., Naidoo,D.,
            Lewandowski,K., Afrough,B., Graham,V., Aarons,E., Hewson,R.,
            Vipond,R., Dunning,J., Chand,M., Brown,C., Cohen-Gihon,I., Erez,N.,
            Shifman,O., Israeli,O., Sharon,M., Schwartz,E., Beth-Din,A.,
            Zvi,A., Mak,T.M., Ng,Y.K., Cui,L., Lin,R.T.P., Olson,V.A.,
            Brooks,T., Paran,N., Ihekweazu,C. and Reynolds,M.G.
  TITLE     Exportation of Monkeypox Virus From the African Continent
  JOURNAL   J Infect Dis 225 (8), 1367-1376 (2022)
   PUBMED   32880628
REFERENCE   2  (bases 1 to 197209)
  AUTHORS   Senkevich,T.G., Yutin,N., Wolf,Y.I., Koonin,E.V. and Moss,B.
  TITLE     Ancient Gene Capture and Recent Gene Loss Shape the Evolution of
            Orthopoxvirus-Host Interaction Genes
  JOURNAL   mBio 12 (4), e0149521 (2021)
   PUBMED   34253028
  REMARK    Gene nomenclature follows the proposal set forth in this
            publication, based on ortholog clustering of ORFs in
            orthopoxviruses.
REFERENCE   3  (bases 1 to 197209)
  AUTHORS   Zmasek,C.M., Knipe,D.M., Pellett,P.E. and Scheuermann,R.H.
  TITLE     Classification of human Herpesviridae proteins using
            Domain-architecture Aware Inference of Orthologs (DAIO)
  JOURNAL   Virology 529, 29-42 (2019)
```

Choosing a reference genome is not always non-trivial… How would you do it?

Importantly, we need to map our reads to a reference from the right Mpox Clade, as we know the Clades are very divergent from one another. Here is a midpoint rooted global phylogeny of Mpox – you can see Clade 1 and Clade 2 are separated by very long internal branches that represent a lot of divergence! Ifeanyi will touch on this a bit more later on!



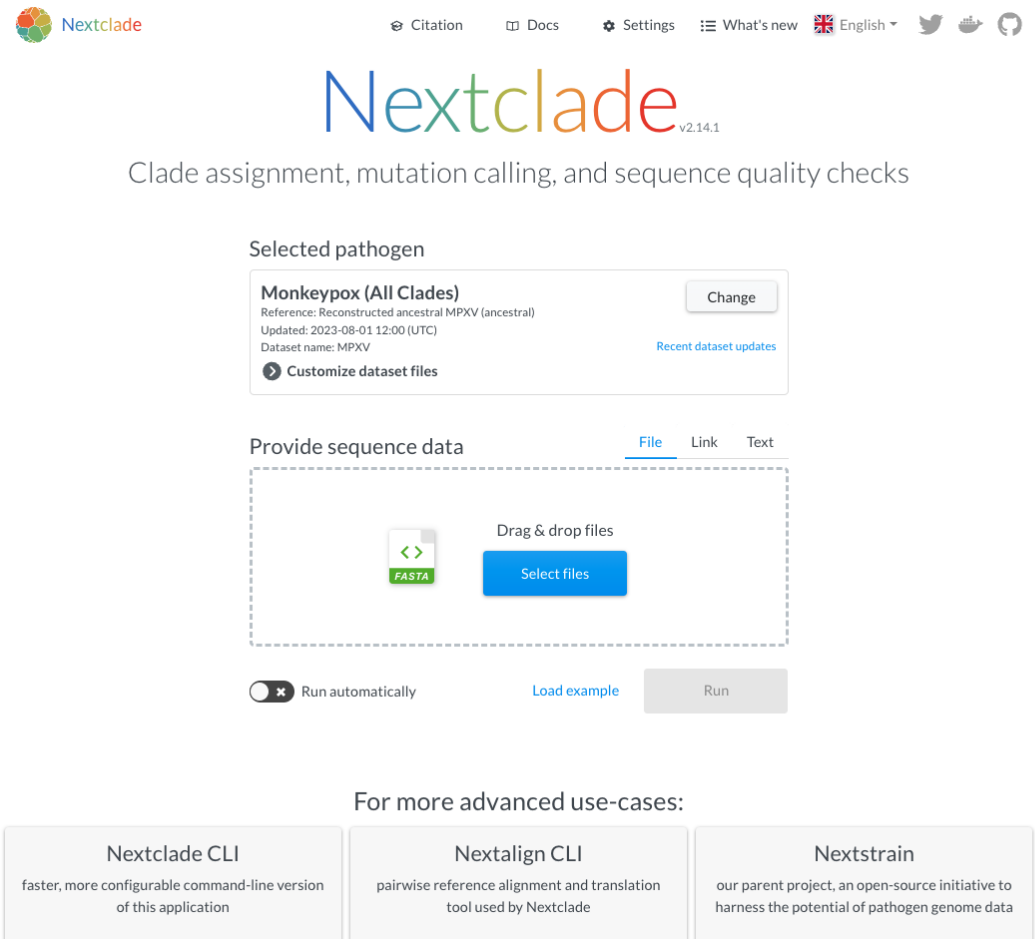
That figure is from:
https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.3001769

So, we don't really know what Clade our sample is from *a priori*, right? This is why *de novo* assembly is a more appropriate assembly method for metagenomics if we are unsure as a first round. However, it is a little more computationally/conceptually intensive, so we'll go with reference-based assembly for now. We'll get back to *de novo* assembly! Additionally, we can always just map to a reference from Clade 1 and Clade 2 (a and b) separately and see which one generates a more complete sequence!

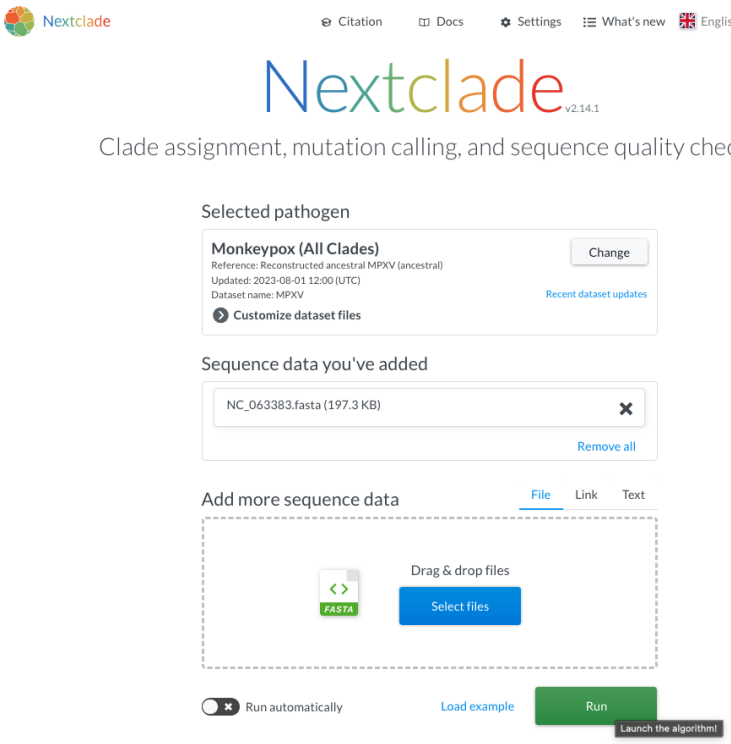
But for now we'll use NC_063383 – but what Clade is NC_063383? We know it's from Nigeria, where CLade 2b predominates, but let's double check…

We'll use a web-server tool called Nextclade to find out! Nextclade is run by the Nextstrain team and can help us designate Clades for consensus sequences as well as perform alignments and quality checks – a very useful tool (and available for a few pathogens including SARS-CoV-2!).

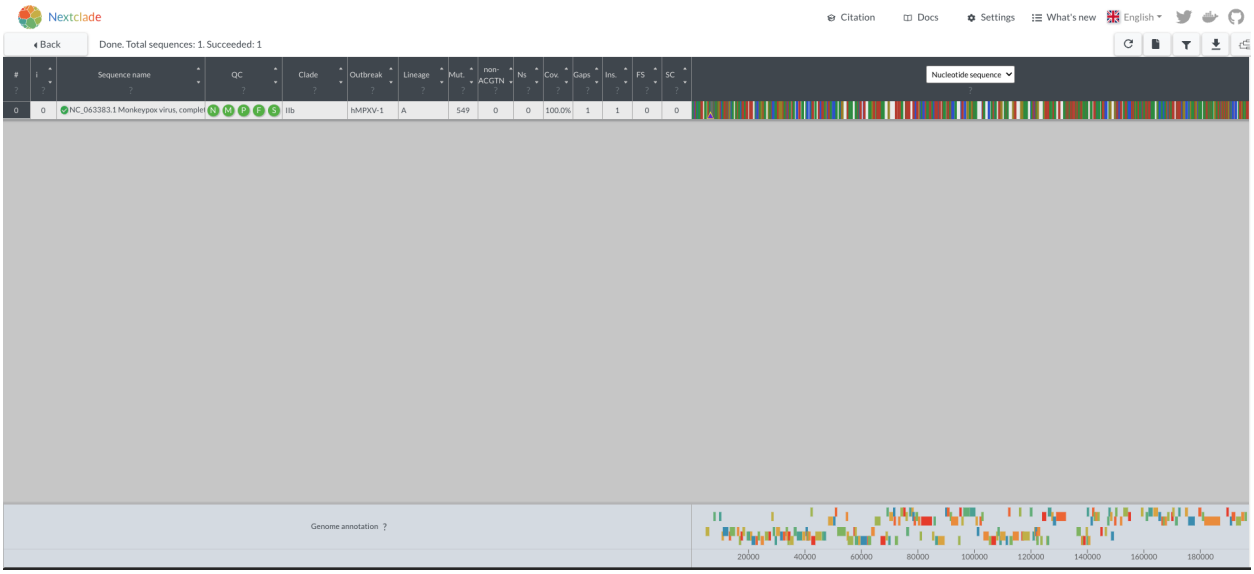
We'll go to the webpage: https://clades.nextstrain.org/, and make sure we select Mpox (All Clades) as the pathogen.



And we'll upload the fasta of NC_063383 we just downloaded – you can drag and drop, or click upload and select the file.

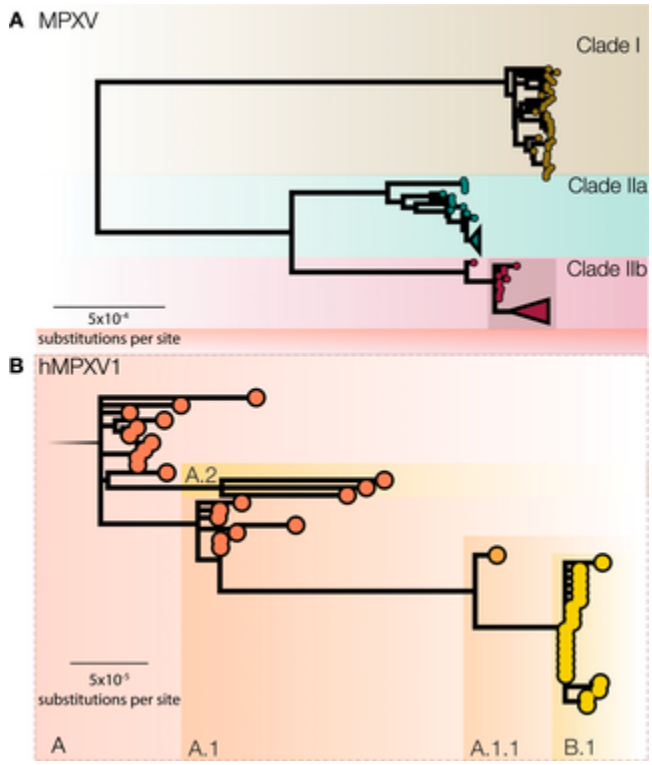


We'll give it a minute to run, but then our output should look like:



So, in the "Clade" column, we can see it's a Clade 2b sequence, of the A lineage.

What does that mean? Here's the paper designated the new nomenclature: https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.3001769

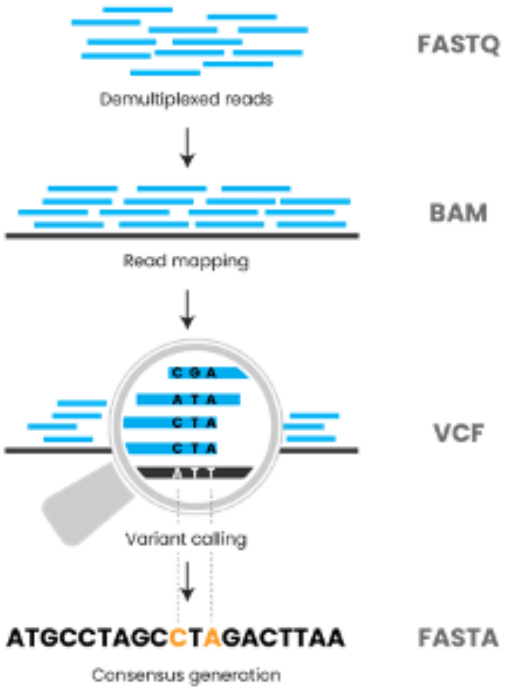


We can see in Figure A above that Clade 2b is the Clade circulating predominantly in West Africa, with export to other countries globally. We can also see, based on Figure B, that Clade 2B has lineages designated within the Clade, which reflects the evolutionary history of the 2022 European outbreak (designated lineage B.1). Lineage A is the more basal lineage, ancestral to the currently circulating diversity, including the European outbreak (lineage B.1). Our invited speaker Áine O'Toole will touch on this a little bit more later in the week!

So back to read mapping! Now we know we're mapping against a Clade2b reference, that we expect to be ancestral to currently circulating Clade2b diversity.

As a brief reminder, here is how read mapping works conceptually:

Once all of our reads (from our paired, trimmed fastqs) are mapped to the right reference genome (in a fasta format), we can use this file (the output BAM file) to generate our consensus genome from, and call variants e.g:



There are many, many read mapper tools we could use for this step. Today, we'll use *bwa* (which stands for the Burrows Wheeler Aligner), but there are other great options like *minimap2* or *bowtie2*. *Bwa* is great for low divergent sequences and large reference genomes, but *minimap2* is probably a bit faster for short reads. It is always good to

consider your specific pathogen/sequencing technology/computational power needs when setting up your own pipelines! It might not be the same tools as we're using.

We'll combine *bwa* with a tool called *samtools*, which is a great tool in general for working with genomic data.

Both *bwa* and *samtools* can be downloaded from conda!

```
conda install -c bioconda bwa
```

```
conda install -c bioconda samtools
```

We can see where *bwa* is installed with:

```
which bwa
```

Here is *bwa* in my PATH:

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ which bwa
/Users/eparker/mambaforge/envs/Mpox/bin/bwa
```

Just as a reminder, our input is the paired trimmed fastq files (R1 and R2) and our output is called a BAM (or bam) file – which is all our reads aligned to the reference genome:

Step	Read quality	Read filtering	Read mapping
Tool	Fastqc	Trimmomatic	Bwa + Samtools
Input	R1.fastq R2.fastq	R1.fastq R2.fastq	R1_trim.fastq R2_trim.fastq
Output	R1.html R2.html	R1_trim.fastq R2_trim.fastq	Bam

Let's see if we have all the right files in our directory:

```
ls
```

```
(base) MacBook-Pro:Session_2 eparker$ ls
NC_063383.fasta    S1_R2_001_P.fastq  fastqs
S1_R1_001_P.fastq  fastqc_report      intermediates
```

The first step of read mapping with *bwa* is indexing the reference genome. Indexing creates a set of files that helps *bwa* find the right potential alignment site for the reads relative to the reference genome a lot faster. We need to index every new reference genome we work with for *bwa* and keep the files generated in the same folder we're going to run the read mapping in.

Let's try:

```
bwa index NC_063383.fasta
```

Our terminal should look like:

```
(base) MacBook-Pro:Session_2 eparker$ bwa index NC_063383.fasta
[bwa_index] Pack FASTA... 0.00 sec
[bwa_index] Construct BWT for the packed sequence...
[bwa_index] 0.03 seconds elapse.
[bwa_index] Update BWT... 0.00 sec
[bwa_index] Pack forward-only FASTA... 0.00 sec
[bwa_index] Construct SA from BWT and Occ... 0.01 sec
[main] Version: 0.7.17-r1198-dirty
[main] CMD: bwa index NC_063383.fasta
[main] Real time: 0.040 sec; CPU: 0.040 sec
```

Let's try and see if the index files were generated correctly:

```
ls
```

```
(base) MacBook-Pro:Session_2 eparker$ ls
NC_063383.fasta      NC_063383.fasta.bwt  S1_R1_001_P.fastq  fastqs
NC_063383.fasta.amb  NC_063383.fasta.pac  S1_R2_001_P.fastq  intermediates
NC_063383.fasta.ann  NC_063383.fasta.sa   fastqc_report
```

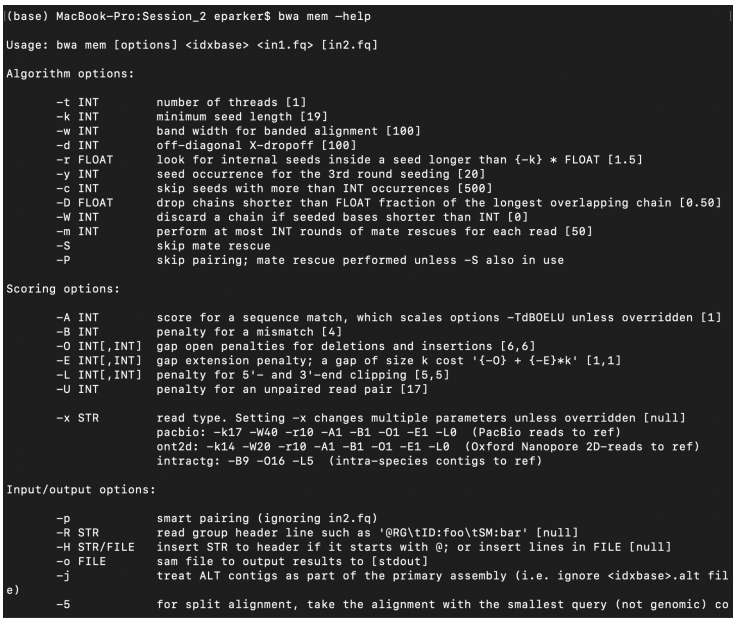
Here we can see that *bwa index* created a few files: NC_063383.fasta.* (NC_063383.fasta.amb, NC_063383.fasta.ann, NC_063383.fasta.bwt,

NC_063383.fasta.pac etc etc). These are our indexing files – we have to keep them in the same directory as our read files (where we're running the read mapping commands).

Now, we can align our reads to our reference genome! We will use *bwa mem*, which is the most accurate and fast version of the *bwa* algorithm. But as always, consult the manual to see if this is the best tool for your needs too!

Let's see what input *bwa mem* requires:

```
bwa mem –help
```



```
(base) MacBook-Pro:Session_2 eparker$ bwa mem -help

Usage: bwa mem [options] <idxbase> <in1.fq> [in2.fq]

Algorithm options:

       -t INT        number of threads [1]
       -k INT        minimum seed length [19]
       -w INT        band width for banded alignment [100]
       -d INT        off-diagonal X-dropoff [100]
       -r FLOAT      look for internal seeds inside a seed longer than {-k} * FLOAT [1.5]
       -y INT        seed occurrence for the 3rd round seeding [20]
       -c INT        skip seeds with more than INT occurrences [500]
       -D FLOAT      drop chains shorter than FLOAT fraction of the longest overlapping chain [0.50]
       -W INT        discard a chain if seeded bases shorter than INT [0]
       -m INT        perform at most INT rounds of mate rescues for each read [50]
       -S            skip mate rescue
       -P            skip pairing; mate rescue performed unless -S also in use

Scoring options:

       -A INT        score for a sequence match, which scales options -TdBOELU unless overridden [1]
       -B INT        penalty for a mismatch [4]
       -O INT[,INT]  gap open penalties for deletions and insertions [6,6]
       -E INT[,INT]  gap extension penalty; a gap of size k cost '{-O} + {-E}*k' [1,1]
       -L INT[,INT]  penalty for 5'- and 3'-end clipping [5,5]
       -U INT        penalty for an unpaired read pair [17]

       -x STR        read type. Setting -x changes multiple parameters unless overridden [null]
                     pacbio: -k17 -W40 -r10 -A1 -B1 -O1 -E1 -L0  (PacBio reads to ref)
                     ont2d: -k14 -W20 -r10 -A1 -B1 -O1 -E1 -L0  (Oxford Nanopore 2D-reads to ref)
                     intractg: -B9 -O16 -L5  (intra-species contigs to ref)

Input/output options:

       -p            smart pairing (ignoring in2.fq)
       -R STR        read group header line such as '@RG\tID:foo\tSM:bar' [null]
       -H STR/FILE   insert STR to header if it starts with @; or insert lines in FILE [null]
       -o FILE       sam file to output results to [stdout]
       -j            treat ALT contigs as part of the primary assembly (i.e. ignore <idxbase>.alt fil
e)
       -5            for split alignment, take the alignment with the smallest query (not genomic) co
```

You can also see the full *bwa* help page here: https://bio-bwa.sourceforge.net/bwa.shtml

We will run *bwa mem* with the default parameters for now – but always make sure you read the manual and understand if these parameters are appropriate for your dataset!

Now, let's align our reads against our indexed reference genome. This command is a bit long, so we'll break it down bit by bit – don't run it yet!

```
bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq | samtools view -u -@ 3  - | samtools sort -@ 3 -o S1.bam
```

The first part:

```
bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq
```

- This first part tells *bwa mem* to align our read files (S1_R1_001_P.fastq and S1_R2_001_P.fastq) to the reference genome (NC_063383.fasta).

- "*-t 3"* indicates "threads", or the number of processors *bwa mem* is going to run on. We specify three, but again, adapt to your server/computer as needed!

If we just run:

```
bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq
```

We can see *bwa mem* running as below:

```
(base) MacBook-Pro:Session_2 eparker$ bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq
[M::bwa_idx_load_from_disk] read 0 ALT contigs
@SQ     SN:NC_063383.1  LN:197209
@HD     VN:1.5  SO:unsorted     GO:query
@PG     ID:bwa  PN:bwa  VN:0.7.17-r1198-dirty   CL:bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq
[M::process] read 298358 sequences (30000045 bp)...
[M::process] read 298362 sequences (30000014 bp)...
[M::mem_pestat] # candidate unique pairs for (FF, FR, RF, RR): (14, 131194, 22, 20)
[M::mem_pestat] analyzing insert size distribution for orientation FF...
[M::mem_pestat] (25, 50, 75) percentile: (107, 231, 239)
[M::mem_pestat] low and high boundaries for computing mean and std.dev: (1, 503)
[M::mem_pestat] mean and std.dev: (210.21, 103.06)
[M::mem_pestat] low and high boundaries for proper pairs: (1, 635)
[M::mem_pestat] analyzing insert size distribution for orientation FR...
[M::mem_pestat] (25, 50, 75) percentile: (185, 224, 282)
[M::mem_pestat] low and high boundaries for computing mean and std.dev: (1, 476)
[M::mem_pestat] mean and std.dev: (236.12, 73.61)
[M::mem_pestat] low and high boundaries for proper pairs: (1, 573)
[M::mem_pestat] analyzing insert size distribution for orientation RF...
[M::mem_pestat] (25, 50, 75) percentile: (49, 133, 214)
[M::mem_pestat] low and high boundaries for computing mean and std.dev: (1, 544)
[M::mem_pestat] mean and std.dev: (126.90, 104.23)
[M::mem_pestat] low and high boundaries for proper pairs: (1, 709)
[M::mem_pestat] analyzing insert size distribution for orientation RR...
[M::mem_pestat] (25, 50, 75) percentile: (156, 184, 313)
[M::mem_pestat] low and high boundaries for computing mean and std.dev: (1, 627)
[M::mem_pestat] mean and std.dev: (217.40, 78.29)
[M::mem_pestat] low and high boundaries for proper pairs: (1, 784)
[M::mem_pestat] skip orientation FF
[M::mem_pestat] skip orientation RF
[M::mem_pestat] skip orientation RR
```

However, you'll also soon see this madness pop up for lines and lines and lines:

```
[M::mem_process_seqs] Processed 298358 reads in 4.230 CPU sec, 1.388 real sec
HG5N3DMXY:1:1101:10104:25347    83      NC_063383.1     52938   60      101M    =       52722   -317TTGTACTCGCTCTATATTCGAATACCATGGATTGAGGAAGCACATATGTTGCCG
CACCAGCGTCTGTTTTTGTTTCTACTCCATAATACTCCCACAAATACFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10104:25347    163     NC_063383.1     52722   60      101M    =       52938   317 CAACGACACCGACTAATGTTTCCGGAGATAGATTTCCAAAGATACACACATTAG
GATACAGACTGTTATAATCAAAGATTAATACATTATTACTAAACATTFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10176:5619     99      NC_063383.1     173461  60      101M    =       173555  195 CCACAACGGACACCGACGTCTTTTGGATAAGTAATGGTATATGTATTACGAAGA
AGAAGACGAGGACGGAGACGGTAGAATAAGTGTAGCAAATAAAATCTFFFFFFFFFFFFFFFFFFFF::FFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FF,FF,FFFFF:FF:FF:FFFFFFF:FFFF:FFFFFFFFFFFF,FNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10176:5619     147     NC_063383.1     173555  60      101M    =       173461  -195AAAATCTATATGACCGATAAGAGACGTGTTATTACATCCTGGTTAAACATTAAT
CCTGTCAAGGAAGAAGATGCTACAACGTTTACGTGTATGGCGTTTACFFFFFF,F:FFFFFFFF,F:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFF:NM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10321:26068    83      NC_063383.1     58748   60      101M    =       58541   -308CTGTCAAATCATCTTCCTGAGAACCCATAAATACACCAAATATAGCAGCGTACA
ACTTATCCATTTATACATTGAATATTGGCTTTTCTTTATCGCTATCTFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10321:26068    163     NC_063383.1     58541   60      101M    =       58748   308 CACTGATACTATTGAAAACGAGTTGATCTTCAAATTCCGCCATTTAAGTTCACC
AAACAACTTTTAAATACAAATATATCAATAGTAGTAGAATAAGAACTFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFF,FFFFFFFFFFF:FF,FFFFFFFFFFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10366:28902    83      NC_063383.1     50762   60      101M    =       50578   -285GATACGCAAACCTAGATGCTAGTGACTATTATCCCATTTCGTTGGGATTACTTA
AAAAGTTTGAATTTCTCATGTCTCTATATAAAGGTCCTATTCCCGTAF:FFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10366:28902    163     NC_063383.1     50578   60      101M    =       50762   285 GATATAAATATTATGGCCGCAACCGTTCCGCGTTTTGACGATGTGTACAAAAAT
GCACAAAGAAGAATTCTAGATCAAGAAACATTTTTAGTAGAGGTCTFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFF:FF:FFFFFFFFF:FFFFFFFFFFF,FF,FFFFFFFFFF:FFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10429:8750     83      NC_063383.1     94660   60      96M     =       94372   -384ATGGACATTTTTATAGTTAAGGATAATAAGTATCCCAAAGTAGATAACGACGAT
AACGAAGTATTTATACTTTTAGGAAATCACAATGACTTTATC FFFFF,:FFFFFFFFFFFFFF,:FFF,F:FFF,FFFFF,:FF:FF:FFFFFFF,FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFF      NM:i:0
       MD:Z:96 MC:Z:101M       MQ:i:60 AS:i:96 XS:i:0
HG5N3DMXY:1:1101:10429:8750     163     NC_063383.1     94372   60      101M    =       94660   384 GTATTATTAATAGATATATTAATACTTTTGTAGGGTAACAGAGGTTCTACGTCT
TCTAAAAATAAAAGTTTTATAACATCTGGCCTGTTCATAAATAAAAA:FFFFFFFF:FFFFFFF:FF,FFFF:FFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFF,F:FFFFFF:FFF:FFFFFFFFFFFFFFFFFFNM:i:0
   MD:Z:101        MC:Z:96M        MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10456:25770    83      NC_063383.1     52938   60      101M    =       52722   -317TTGTACTCGCTCTATATTCGAATACCATGGATTGAGGAAGCACATATGTTGCCG
CACCAGCGTCTGTTTTTGTTTCTACTCCATAATACTCCCACAAATACFFF:FFFFFFFFFF,FFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10456:25770    163     NC_063383.1     52722   60      101M    =       52938   317 CAACGACACCGACTAATGTTTCCGGAGATAGATTTCCAAAGATACACACATTAG
GATACAGACTGTTATAATCAAAGATTAATACATTATTACTAAACATTFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF,F:FFFFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10474:10645    99      NC_063383.1     32614   60      101M    =       32693   180 ATATATATCTTTTCAGTTCCCCTACGTATCTATAATCATCTGTAGGAAATGGAA
GATATTTCCATTTATCTACTGTTCCTAATATCATATGCGGTGGTGTAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF::FFFFFFFFFFFFFFFFFFFFFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10474:10645    147     NC_063383.1     32693   60      101M    =       32614   -180TAATATCATATGCGGTGGTGTAGAACCATTAAGCGCGAAAGATGTTATTTCGCA
TCGTATTTTAACTTCGCAATAATTTCTGGTTAGATAACGCACTCTACFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFNM:i:0
   MD:Z:101        MC:Z:101M       MQ:i:60 AS:i:101        XS:i:0
HG5N3DMXY:1:1101:10610:24158    83      NC_063383.1     136632  60      101M    =       136515  -218GAATATTAGTACATTCATTGTCTATTATTAACGTATTTCTAAACCCAAAATGTA
```

This is technically the output of *bwa mem*! But it's printing to the screen instead of printing to a file which we can use for our analyses!

We can try:

```
bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq > S1.sam
```

The ">" part of this command redirects the output from the screen – as we did previously, so it won't print to the terminal screen – into a file we name S1.sam. So, it should save all that confusing text on the screen to a new file in our directory named S1.sam!

Now, what is a SAM file? Simply put, a SAM file is a tab-delimited file with information of each read and where it mapped to in the genome. You can attempt to open this file in a text editor to look at the SAM format – which is best explained here: https://academic.oup.com/bioinformatics/article/25/16/2078/204688?login=true.

However, for now, we're going to move on to talk about the output format we really want – a BAM. A BAM is basically a compressed binary (the B in BAM) version of the SAM! It's compressed to save storage space, and can be indexed so we can efficiently access the read data.

So instead of:

```
bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq > S1.sam
```

We're going to convert the SAM output we get from *bwa mem* into a BAM. We'll use *samtools* for this! Again, our full mapping command is:

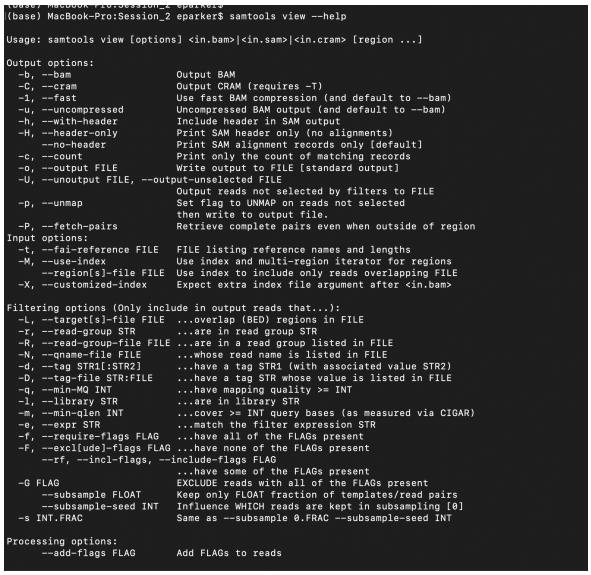
```
bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq | samtools view -u -@ 3  - | samtools sort -@ 3 -o S1.bam
```

So we know what the first part of that command does. Let's focus on the second part:

```
bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq | samtools view -u -@ 3  -
```

You may notice the "|" character here. This is what we call a pipe – basically, it takes everything that comes in front of it (the output of that first command) and gives it as input to the command that follows the pipe. So, it takes the output of *bwa mem* (the SAM output) and "pipes" that as input to the next tool – which here is *samtools* and its command *view. Samtools* is the part of the code that converts the SAM into BAM format. You can see all the options with the *–help* function, as usual:

```
samtools -help
```



Let's take another look at our command:

```
bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq | samtools view -u -@ 3  -
```

- The "-@ 3" part refers to the number of threads *samtools* needs to use.
- We're going to use "-u" for now, which keep it uncompressed – but remember our full command is:

```
bwa mem -t 3 NC_063383.fasta S1_R1_001_P.fastq S1_R2_001_P.fastq | samtools view -u -@ 3  - | samtools sort -@ 3 -o S1.bam
```

So, we're again "piping" our output from the command *samtools view* as input into *samtools sort. Samtools sort* sorts our BAM file – you can sort by e.g. read name, location on the chromosome. The type of sorting may differ across different tools downstream in the pipeline, but the default sort will work for us!

- "-@ 3" part again refers to the number of threads *samtools* needs to use.
- "-o" tells *samtools sort* to output the result as a file called S1.bam!

If we run the command above, our screen output will look a bit like:

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ bwa mem -t 3 NC_063383.f
asta S1_R1_001_P.fastq S1_R2_001_P.fastq | samtools view -u -@ 3
- | samtools sort -@ 3 -o S1.bam
[M::bwa_idx_load_from_disk] read 0 ALT contigs
[M::process] read 298358 sequences (30000045 bp)...
[M::process] read 298362 sequences (30000014 bp)...
[M::mem_pestat] # candidate unique pairs for (FF, FR, RF, RR): (14
, 131194, 22, 20)
[M::mem_pestat] analyzing insert size distribution for orientation
 FF...
[M::mem_pestat] (25, 50, 75) percentile: (107, 231, 239)
[M::mem_pestat] low and high boundaries for computing mean and std
.dev: (1, 503)
[M::mem_pestat] mean and std.dev: (210.21, 103.06)
[M::mem_pestat] low and high boundaries for proper pairs: (1, 635)
[M::mem_pestat] analyzing insert size distribution for orientation
 FR...
[M::mem_pestat] (25, 50, 75) percentile: (185, 224, 282)
[M::mem_pestat] low and high boundaries for computing mean and std
.dev: (1, 476)
[M::mem_pestat] mean and std.dev: (236.12, 73.61)
[M::mem_pestat] low and high boundaries for proper pairs: (1, 573)
[M::mem_pestat] analyzing insert size distribution for orientation
 RF...
[M::mem_pestat] (25, 50, 75) percentile: (49, 133, 214)
[M::mem_pestat] low and high boundaries for computing mean and std
.dev: (1, 544)
[M::mem_pestat] mean and std.dev: (126.90, 104.23)
[M::mem_pestat] low and high boundaries for proper pairs: (1, 709)
[M::mem_pestat] analyzing insert size distribution for orientation
 RR...
[M::mem_pestat] (25, 50, 75) percentile: (156, 184, 313)
[M::mem_pestat] low and high boundaries for computing mean and std
.dev: (1, 627)
[M::mem_pestat] mean and std.dev: (217.40, 78.29)
[M::mem_pestat] low and high boundaries for proper pairs: (1, 784)
[M::mem_pestat] skip orientation FF
[M::mem_pestat] skip orientation RF
```

Let's try:

```
ls
```

We should see that we now have a file called S1.bam!

```
(base) MacBook-Pro:Session_2 eparker$ ls
NC_063383.fasta      NC_063383.fasta.ann  NC_063383.fasta.pac  S1.bam            S1_R2_001_P.fastq  fastqs
NC_063383.fasta.amb  NC_063383.fasta.bwt  NC_063383.fasta.sa   S1_R1_001_P.fastq  fastqc_report      intermediates
```

Remember, the BAM is a compressed binary file – so we can't open it in a text editor or in the terminal.

We can first try and get some metrics about our BAM with *samtools:*

```
samtools flagstat S1.bam
```

This shows us some metrics on the number of reads that mapped, paired etc.

```
(base) MacBook-Pro:Session_2 eparker$ samtools flagstat S1.bam
833836 + 0 in total (QC-passed reads + QC-failed reads)
833366 + 0 primary
0 + 0 secondary
470 + 0 supplementary
0 + 0 duplicates
0 + 0 primary duplicates
820556 + 0 mapped (98.41% : N/A)
820086 + 0 primary mapped (98.41% : N/A)
833366 + 0 paired in sequencing
416683 + 0 read1
416683 + 0 read2
816182 + 0 properly paired (97.94% : N/A)
818918 + 0 with itself and mate mapped
1168 + 0 singletons (0.14% : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

We have to use specialized software, such as Geneious (which is proprietary) or IGV (which is free), to view our BAMS. You can download IGV from: https://software.broadinstitute.org/software/igv/download. IGV is a great way to visualize your genomic data

Let's try and view the BAM in IGV.

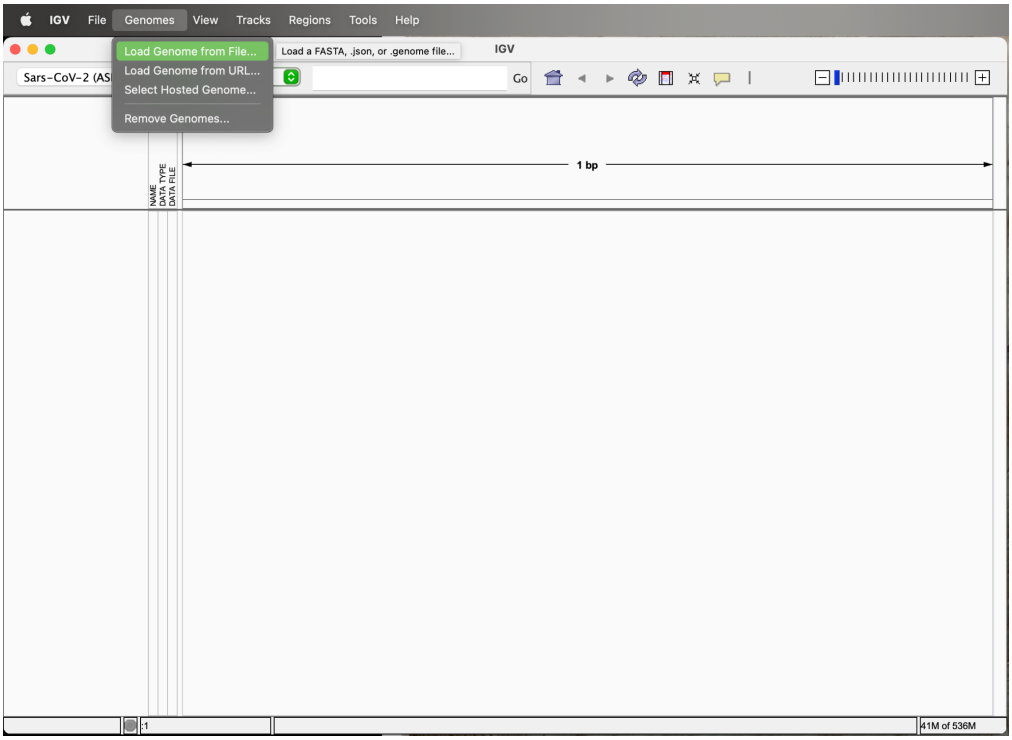
First, we need to index the BAM for viewing. We will use *samtools* for this too.

```
samtools index S1.bam
```

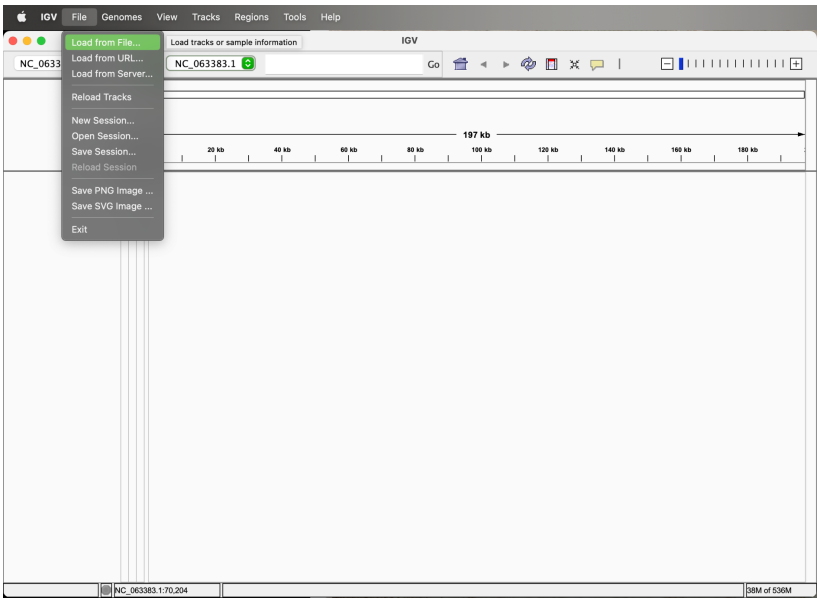
If we use *ls* we can now see the index file S1.bam.bai was created:

```
(Mpox) MacBook-Pro:Mpox_workshop eparker$ ls
NC_063383.fasta          S1.bam.bai
NC_063383.fasta.amb      S1_R1_001_P.fastq
NC_063383.fasta.ann      S1_R2_001_P.fastq
NC_063383.fasta.bwt      fastqc_report
NC_063383.fasta.pac      fastqs
NC_063383.fasta.sa       intermediates
S1.bam
```

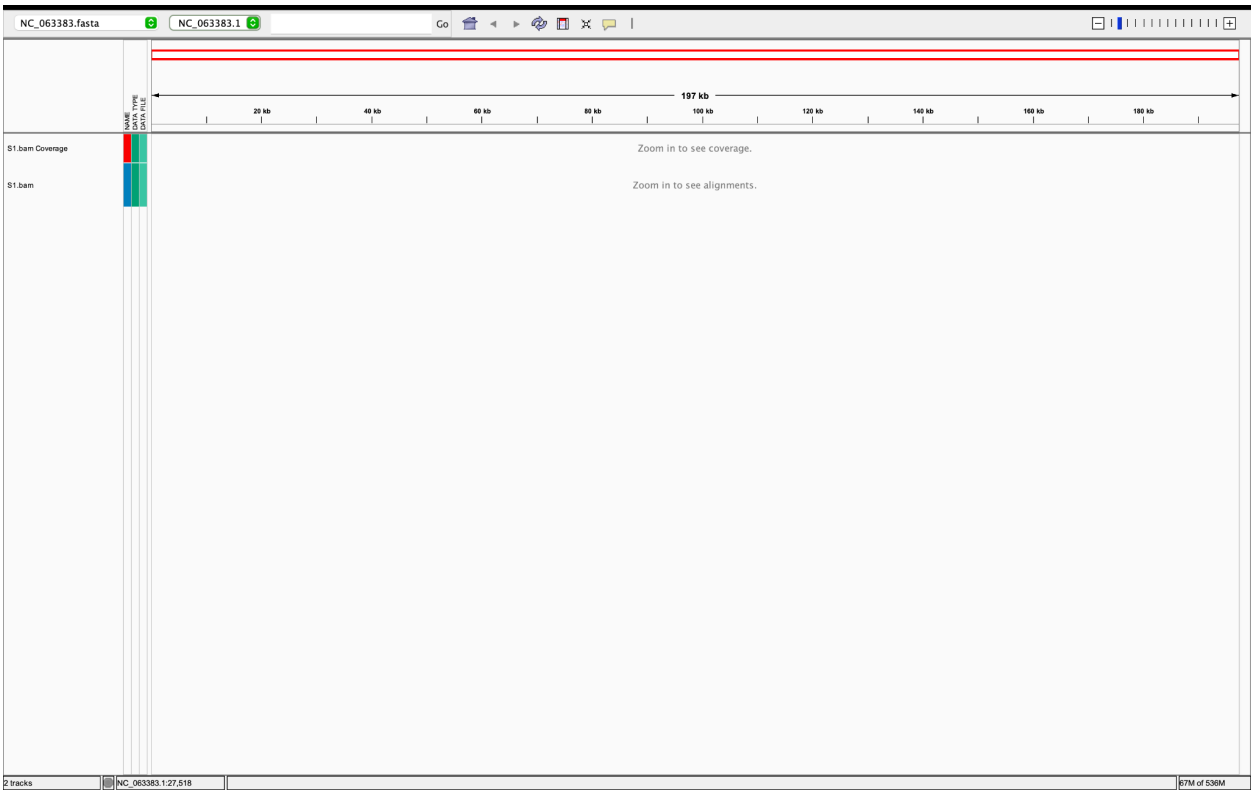
Let's open IGV! The first thing we need to do is load the reference genome (the one the reads in the BAM is mapped to, so NC_063383.fasta) – the index files we previously created need to be in the same folder too! Go to Genomes and then Load Genome from File and choose NC_063383.fasta in your directory.



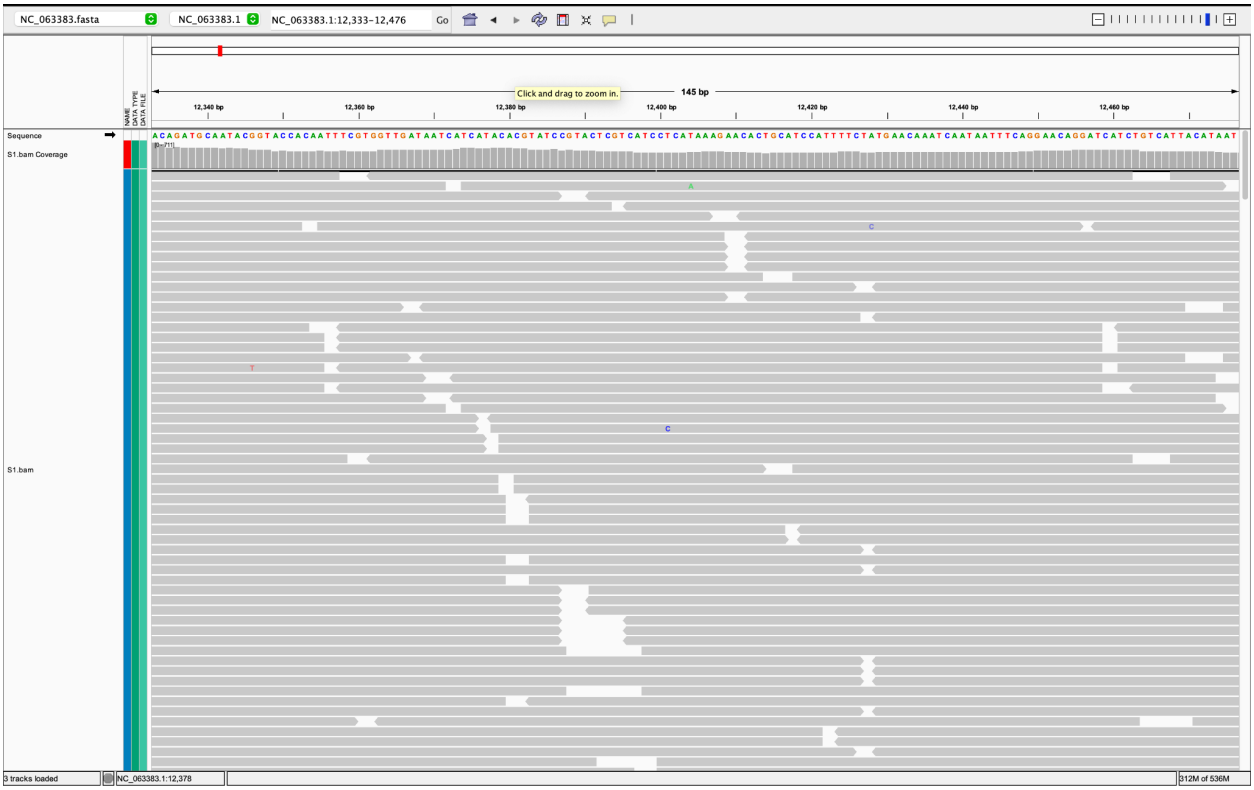
Now, let's open our BAM, by going to File and Load from File.



Our IGV screen should look like:



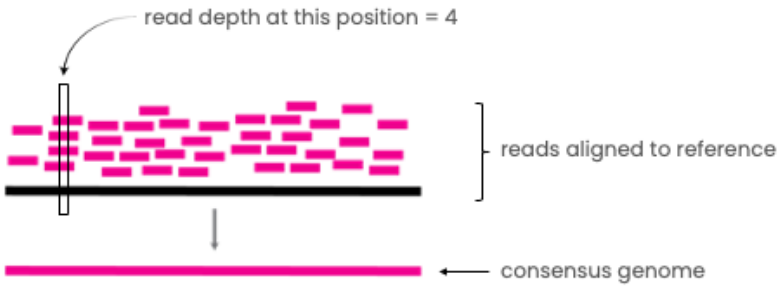
We can zoom in on any portion of the genome to see the reads mapped to the reference – you can use your cursor to drag a little box around your region of interest to zoom in or you can use the coordinates bar up top (the third box in the very top track, e.g. NC_063383:12,333-12,476) to go to the portion of the genome you want to look at.



Each horizontal gray bar represents a read – try scrolling up and down with the bar on the right to get a feeling for how many reads cover the region you're looking at! You can also change the formatting so that reverse and forward reads are different colors (we'll do this in a bit) – here's a good place to start on customizing IGV to your preferences: https://software.broadinstitute.org/software/igv/userguide

Remember, read depth/coverage from our previous training?

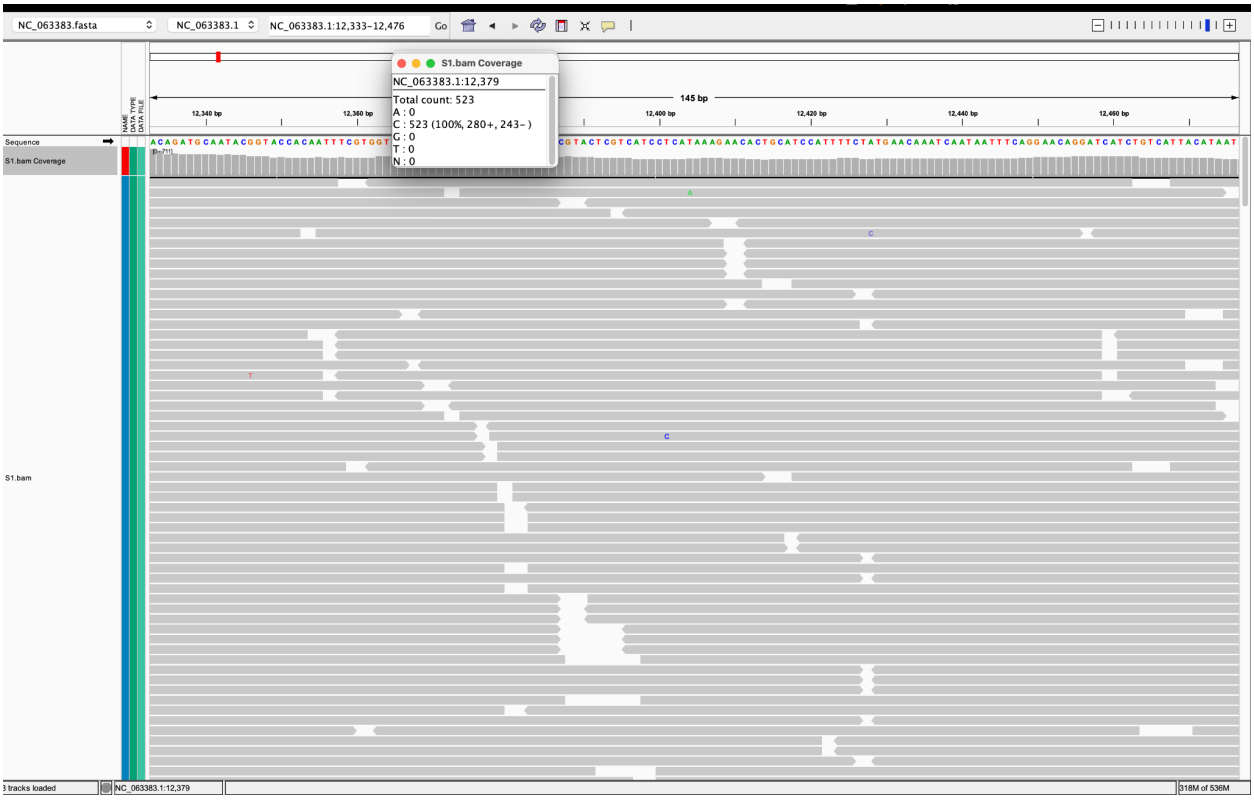
- *Read depth*: how many reads cover each position?



You can see there's a track called "S1.bam coverage" in your IGV – this reflects our read coverage for every position. As you may recall, the number of reads that cover a position tells us how confident we are in our base call at that position.

When we call consensus to generate our consensus sequence or call variants relative to a reference genome, we're going to use a read depth threshold to make sure we only include nucleotides or variants at positions where we have enough reads to be confident of the nucleotide to include i.e that it is not a sequencing error. For Illumina sequence data, a frequently used minimum read depth threshold is 10 – i.e. does the position in the genome you're looking at have at least ten reads covering it? You can get a feel for this conceptually by scrolling down with the bar on the right of the IGV screen.

However, it's easier to click on the histogram bar in the "S1.bam Coverage" track. It should bring up a window that tells you the total count of reads at that position, and also how many of each base is found at that position across all the reads at that position (see Figure below). E.g. For position 12379, we have a depth of 523 reads and 100% of them are a C – so we're pretty sure at that position in the consensus genome we should have a C!



We'll get back to IGV a bit later on when we try to understand how to troubleshoot a pipeline.

Now that we have our BAM file, which will be the input for all of our downstream processes, let's move the trimmed and paired reads somewhere to tidy up.

Let's compress them first:

```
gzip *.fastq
```

Let's move them to the directory we've already created for fastqs:

```
mv *.fastq.gz fastqs
```

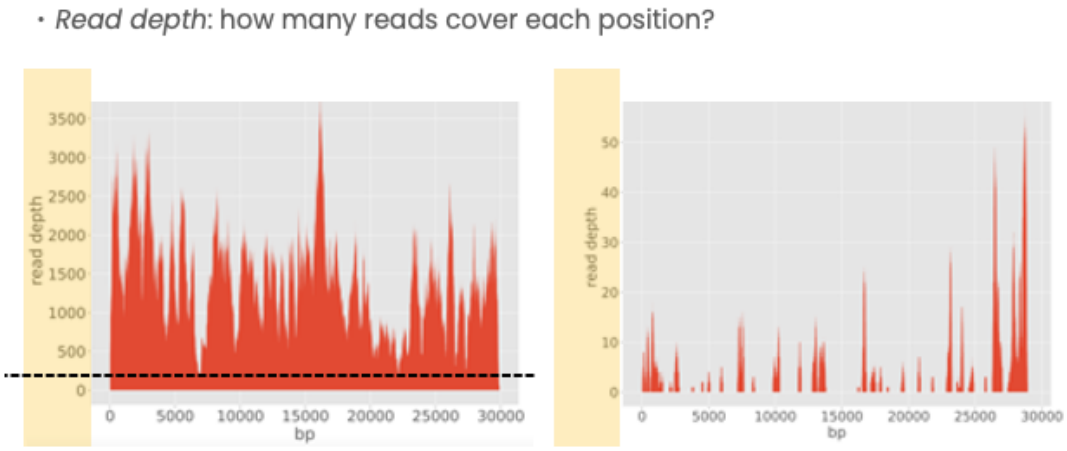
6. Coverage assessment

Coverage is definitely a key metric for us to understand the quality of our assemblies. However, we can't open every BAM file in IGV and click on every bar in the coverage track to see the coverage for every site in the genome! So, let's take a look at how to assess the coverage of our BAM a little more systematically.

Again, we're going to assess the coverage of our BAM (our reads mapped to our reference genome) as input, right?

Step	Read quality	Read filtering	Read mapping	Coverage
Tool	Fastqc	Trimmomatic	Bwa + Samtools	Samtools
Input	R1.fastq R2.fastq	R1.fastq R2.fastq	R1_trim.fastq R2_trim.fastq	Bam
Output	R1.html R2.html	R1_trim.fastq R2_trim.fastq	Bam	CSV

We want information or a plot that can help us understand the read depth distribution across the all genomic positions, like the plots below.



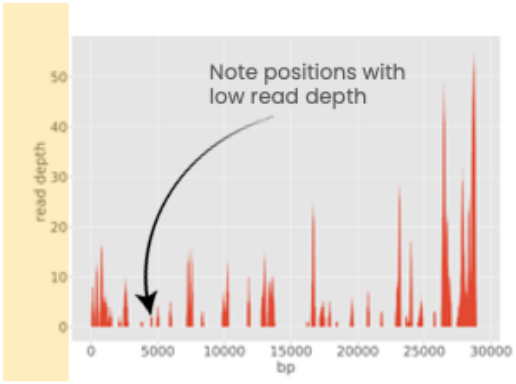
The first plot has a dashed line drawn on the y-axis at 10. This is the minimum depth threshold to call consensus or a variant on Illumina data that is recommended. We can see that no positions in the genome fall below that threshold in the first figure (although there's a region between 5000-10000 that comes close!). So, we expect that we can call consensus or variants at all positions with some confidence for this BAM, as all positions have at least ten reads covering them. In fact, some positions e.g. around 15000 have ~3500 read depth. The higher the read depth, the more confident we can be in our nucleotide or variant calls.

In the second plot, we can see there is pretty poor read depth at most sites. In fact, there are several regions where no reads map to. As these positions fall under our threshold of 10, we would fill them with an ambiguous nucleotide such as "N" (we'll revisit this later on too).

We need to set this minimum read threshold as with very few reads at a position, we can't be sure of a nucleotide call. For example:

- *Read depth*: how many reads cover each position?

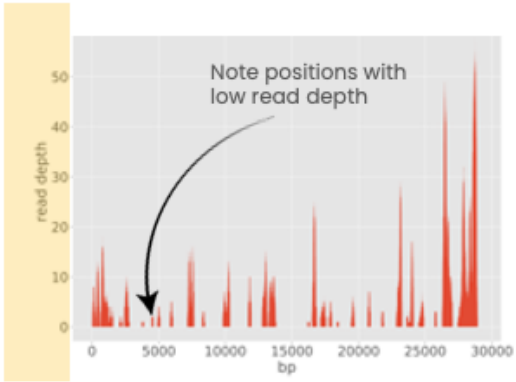
ATGCCTAATCCTAGTTCAAGA
ATGCGTAATCCTAGTTCAAGA
ATGCGTAATCCTAGTTCAAGA



We would say there's a G at that position 5 (in colour), right? But if we had more reads covering position 5:

- *Read depth*: how many reads cover each position?

ATGCCTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA
ATGCGTAATCCTAGTTCAAGA
ATGCGTAATCCTAGTTCAAGA
ATGCGTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA
ATGCCTAATCCTAGTTCAAGA



A C is clearly best supported at that position – maybe the Gs are a true within-host variant, or maybe its sequencing error… This is why we need good read depth!

There could be a few things that went wrong in the second plot. The most likely is that the sample simply did not generate enough reads to cover the genome (which here is a SARS-CoV genome, just as an example). As you can imagine, we need a lot more reads to get good coverage on a Mpox genome of 197 000 bases (vs 30 000 for SARS-CoV-2) – with a minimum coverage of 10, with a standard read length of 150 nt, allowing for read overlap… How many reads do you think we need to get a high quality Mpox genome? That's why we broke out the Novaseq and the big flow cells! Or maybe the reads just did not map to the reference genome – maybe those regions are divergent to the reference

genome? Or maybe, if we're using amplicon sequencing protocols, there was amplicon drop out owing to primer mismatches?

Remember: we performed metagenomic sequencing, not targeted sequencing! So not every read in our fastqs is from the Mpox virus – and we did not deplete host reads so far. So, there is no easy way for us to look at our original unmapped fastqs and see how many Mpox reads we should expect to map based on file size. We can use a program like *kraken,* as we did in the Terra section, and classify all reads by organism to help us understand how many Mpox reads there are in our samples. However, we would need access to a very large database to run *kraken* against… So we'll skip this for now, as you got familiar with it via Terra.

Let's try to replicate the figure above for our BAM – first, we need to generate a file where we have the read depth for every position in the genome. If we then have a strange mutation in our genome or a base that we're unsure of, we can go back to this file and see the coverage for that position.

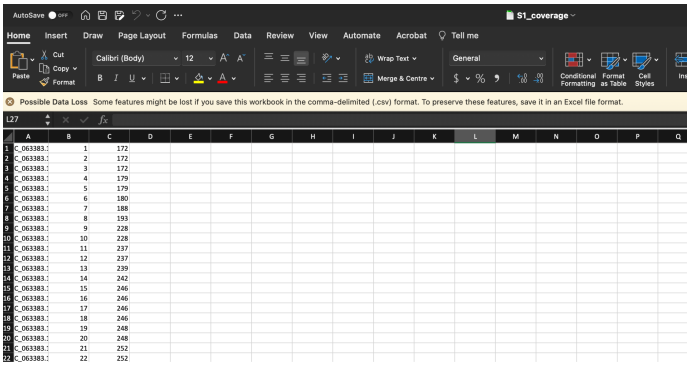
Let's try:

```
samtools depth S1.bam > S1_coverage.tsv
```

Remember, ">" redirect the output of the command in front of it to print it to a file (here named S1_coverage.tsv). So, we're going to use the *samtools depth* command on the S1.bam and then print the output to a file named S1_coverage.tsv.

```
ls
```

Let's open the tab-delimited file we just created. We can see the file has three columns: 1) The reference 2) The genome position 3) The coverage.

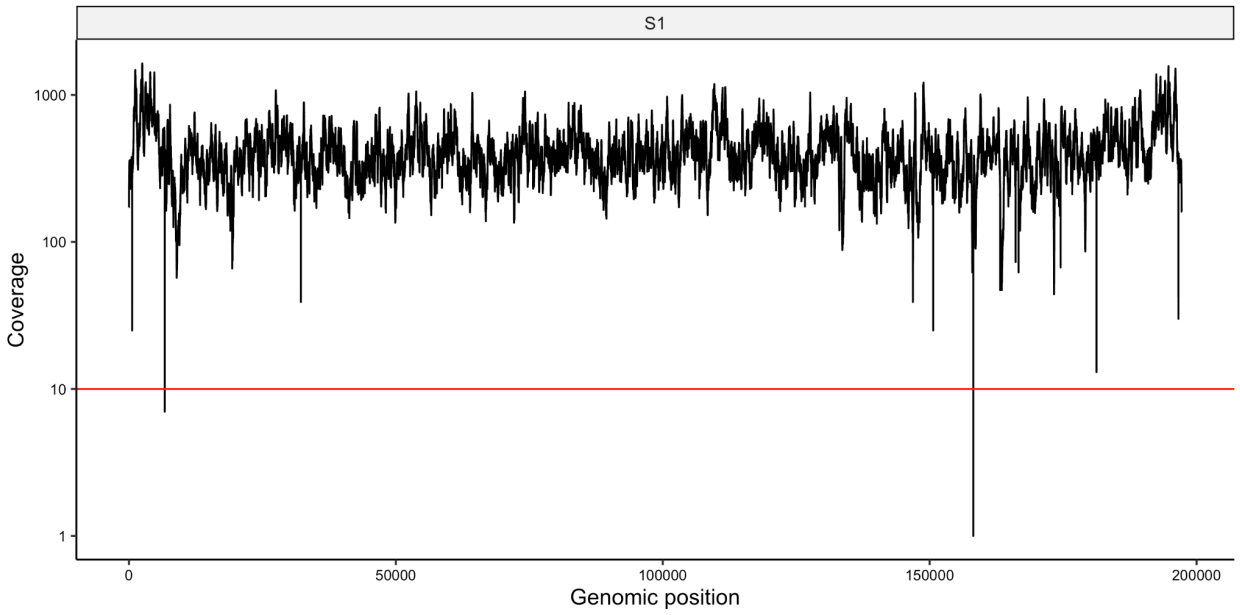


I very simply sorted the third column (the Coverage) from smallest to largest. We can see that only two positions have coverage below 10 – so we expect that this genome will not have too many ambiguous nucleotides (Is this true? What else can introduce an ambiguous nucleotide?)

G11

	A	B	C	D
1	NC_063383.1	158166	1	
2	NC_063383.1	6685	7	
3	NC_063383.1	181246	13	
4	NC_063383.1	597	25	
5	NC_063383.1	150652	25	
6	NC_063383.1	150653	25	
7	NC_063383.1	150654	25	
8	NC_063383.1	150655	25	
9	NC_063383.1	598	26	
10	NC_063383.1	196604	30	
11	NC_063383.1	595	32	
12	NC_063383.1	596	32	
13	NC_063383.1	196605	32	
14	NC_063383.1	150651	35	
15	NC_063383.1	196606	38	
16	NC_063383.1	594	39	
17	NC_063383.1	32209	39	
18	NC_063383.1	146860	39	
19	NC_063383.1	146861	39	
20	NC_063383.1	146862	39	
21	NC_063383.1	196603	40	
22	NC_063383.1	593	41	

You can use this tab-delimited file (S1_coverage.tsv) to make plots – either in Excel, if you're comfortable with that, or in R or python! If you're comfortable with R/R studio, there should be a script called Cov_plot.R in the Dropbox that can generate the following figure as Coverage.png (you just need to change the working directory):



As we can see from the Figure, there's only two positions that cross the red line (read depth minimum threshold of 10)! The remaining positions all have coverage 100-1000 – so we are confident that we can discern the nucleotides at these positions – so we should get a pretty full genome from consensus calling!

Don’t worry about generating this figure now, for the sake of time. Try it at home, and if it doesn’t make sense – hit me up via email!

Let’s tidy up a bit, and move all of our coverage files to a new folder:

```
mkdir coverage
```

```
mv *.R coverage/
```

```
mv *.png coverage/
```

```
mv *_coverage.tsv coverage/
```

Remember, the wildcard “*” is going to grab all files that match the pattern – so if you have more than one sample in the directory, it’s going to move all coverage files to the new directory, not just S1… Just keep this in mind when you’re working on multiple samples.

7. Generating a consensus sequence

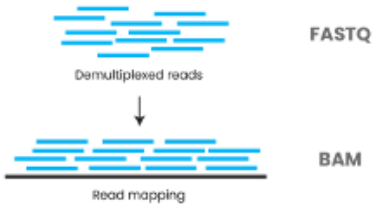
Finally, we’re ready to generate our consensus sequence! For research questions around the origin of a novel pathogen, transmission dynamics during an outbreak or the evolutionary history of a pathogen, we are most likely going to use the consensus genome for our investigation with e.g. phylogenetics.

Remember, a consensus sequence represents the “average” sequence or variant at the “highest frequency” in the potentially diverse within-host population. When we work at the consensus level, we’re not capturing all of the intrahost minor variants that may be present at lower frequencies (sub-consensus) in the population. These minor variants, their frequencies and evolutionary time scale may help us understand how the pathogen is evolving within the host and under what stochastic and selective forces. We’ll take a look at these in the next step, but for now, we’ll focus on the consensus genome.

As a reminder, our input into this step is our BAM, with our filtered paired reads mapped to our reference genome, and our output will be a consensus sequence in the fasta format:

Step	Read quality	Read filtering	Read mapping	Coverage	Consensus calling
Tool	Fastqc	Trimmomatic	Bwa + Samtools	Samtools	Samtools + iVar
Input	R1.fastq R2.fastq	R1.fastq R2.fastq	R1_trim.fastq R2_trim.fastq	Bam	Bam
Output	R1.html R2.html	R1_trim.fastq R2_trim.fastq	Bam	CSV	Fasta

As a reminder, we have the reads mapped to a specific position in the reference genome in our BAM – remember, we took a look at this in IGV:



There are many tools we can use to generate consensus sequences, but we're going to use a combination of *samtools* and *ivar*. *ivar* is, like *samtools*, a very handy and general tool; manual at https://andersen-lab.github.io/ivar/html/manualpage.html

We're going to call the consensus base for every position in the BAM along the reference genome. Well, every position that has a read depth above a certain threshold, so we can be confident in our calls, right?

How do we define what is the consensus nucleotide at a position? We will use a threshold rule – any nucleotide that is present in 50% of the reads (majority rules) will be the consensus nucleotide and will be included in the consensus sequence at that position.

So if at position 1378 in the genome we have: 80% A, 0.1% T and 19.9% G, we will have A at position 1378 in our consensus genome in the fasta - if position 1378 has 10 or more reads covering it! However, if there was no base at 50% frequency e.g. 45% A, 35% T, 20% G – then we don't really know what base we should include in the consensus genome, right? So instead of including an A when we're not sure, we're going to include an ambiguous nucleotide "N". We will also include "N" if there's less than 10 reads covering that position as mentioned.

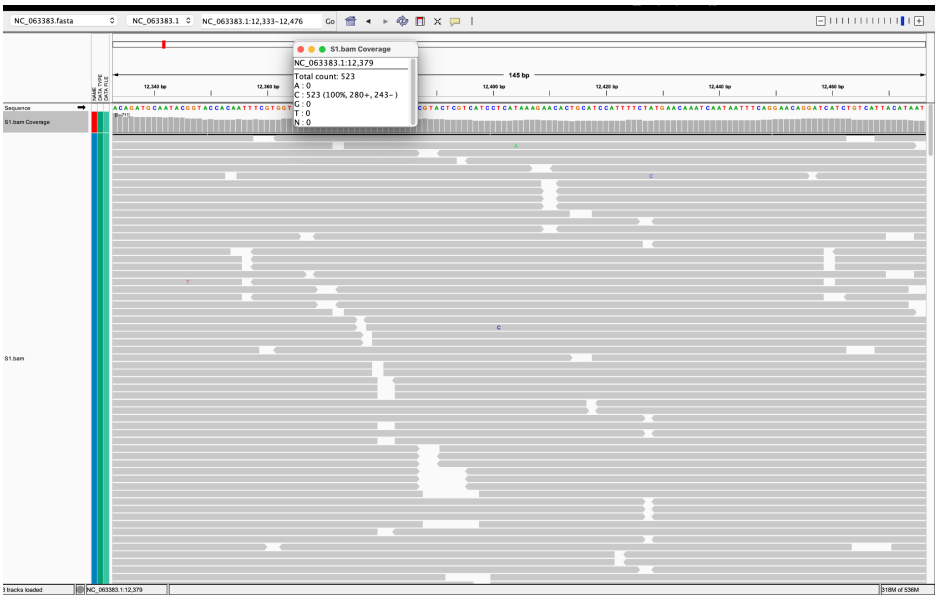
Consider a position with 6As, 3Ts and 1C. The table below shows the consensus nucleotide called with *ivar* at different frequencies:

Minimum frequency threshold	Consensus
0	A
0.5	A
0.6	A
0.7	W(A or T)
0.9	W (A or T)
1	H (A or T or C)

If there are two nucleotides at the same frequency, both nucleotides are used to call an ambiguous base as the consensus. As an example, consider a position with 6 Ts, 2As and 2 Gs. The table below shows the consensus nucleotide called at different frequencies.

Minimum frequency threshold	Consensus
0	T
0.5	T
0.6	T
0.7	D(A or T or G)
0.9	D(A or T or G)
1	D(A or T or G)

Remember, we have looked at how we can see the frequency break down for each position in the BAM in IGV, by clicking on the histogram bar of the position:



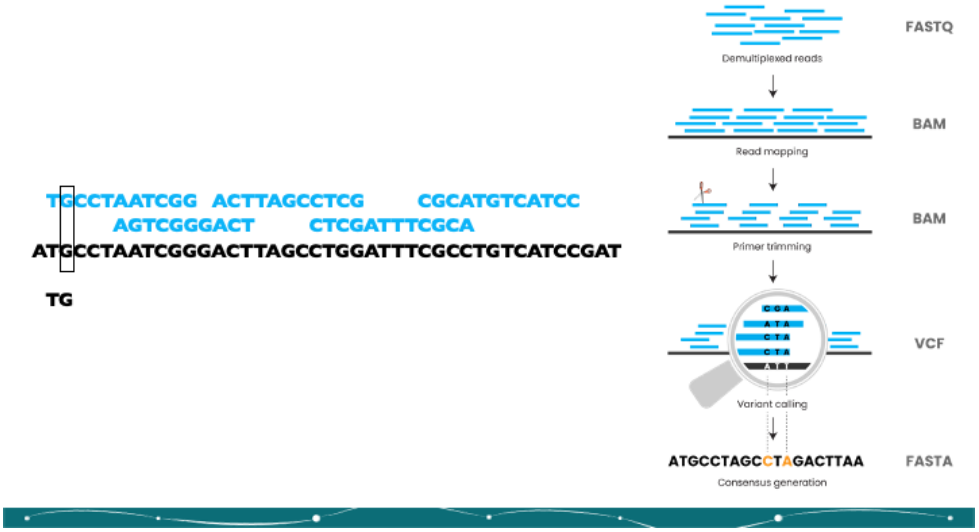
So, if you ever need to see what nucleotide is best supported at a position – or, if there are any other nucleotides close in frequency to the consensus call – you can open the BAM in IGV to double check. E.g. If you get an A at a position, and you're suspicious of it, you can go see what is happening at that position – maybe A is just at 50.1%, and T is at 49%...

Now, is 50% the right threshold? Can we be very confident that a nucleotide is the right call if there's 10 reads and 5 of them are A's? Good question. For some research questions, technologies or pathogens, we might want to have more stringent thresholds – 50% is the default in *ivar* and a read depth of 10 is the rule of thumb for Illumina data (100 for ONT, as it is more error prone) – but we should never just accept default parameters without a bit more thinking and reading!

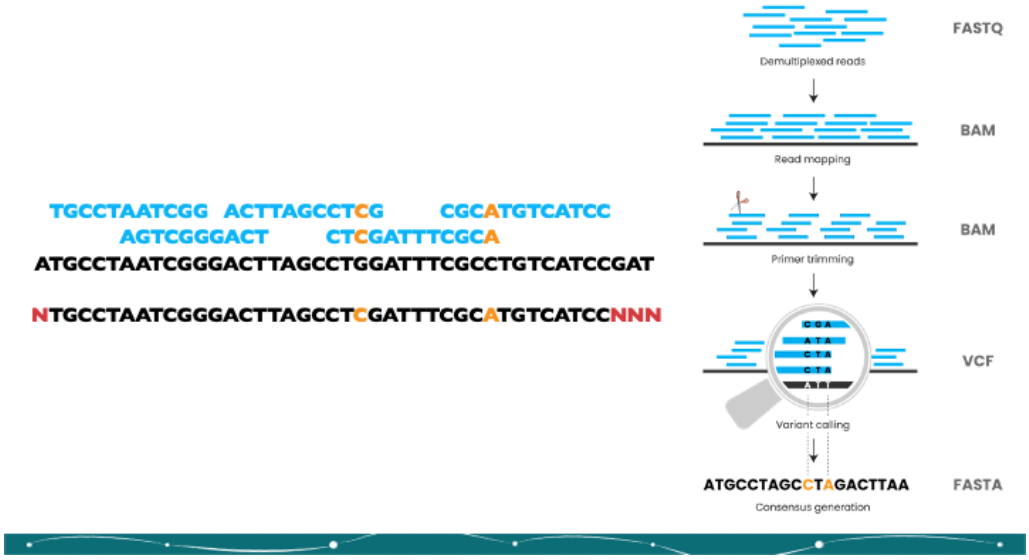
If we have 50% A and 49% T, it's possibly indicative of contamination, as you have a mixed population at that site. It's entirely possible it is a true within-host variant, but it's important to rule out contamination when you find mixed populations at high frequency – so check your BAMS in IGV as above if you're suspicious of mutations!

For now, we're going to use a minimum nucleotide frequency threshold of 50% to call consensus, and a minimum read depth of 10.

The operation of calling consensus works a little like: We move along the genome in the BAM (the reference genome is in black, the reads in blue). We find the base at at least 50% for every nucleotide position with at least a read depth above 10:



When we have a read depth below our threshold (e.g. 10) or no base that reaches our consensus threshold (e.g. 50%), we put a “N” (an ambiguous nucleotide):



The output of consensus calling will be our one consensus sequence in the fasta format. As a reminder:

Let's try calling consensus! We are, of course, in the same directory we've been performing all of our commands in.

The command we'll run looks like (don't run right now!):

```
samtools mpileup -A -d 0 -Q 0 -B S1.bam | ivar consensus -p S1 -t 0.5 -m 10
```

The first part of the command again relies on *samtools.*

```
samtools mpileup -A -d 0 -Q 0 -B S1.bam
```

The command *mpileup* is a bit complicated, but basically creates a new formatted file called the pileup format (http://www.htslib.org/doc/samtools-mpileup.html), which makes it easier and faster for us to understand what nucleotides are at each position.So this part of the command will reformat our S1.bam file into the pileup format with the following parameters:

- "-A" just makes sure we do not skip anomalous read pairs in variant calling.
- "-d 0" sets the maximum read depth at 0, which just means we don't want to lose any data. Sometimes, to save memory you can set a limit to depth e.g. remove everything after 10000 reads at a position. But we don't want to lose data, and our genomes are not that large.
- "-Q 0" sets base-quality to 0, which is used as a filtering mechanism for overlap removal which marks bases as having quality zero and lets the base quality filter remove them.

Now the second part:

```
samtools mpileup -A -d 0 -Q 0 -B  S1.bam | ivar consensus -p S1 -t 0.5 -m 10
```

The rest of the command basically tells us we take that pileup S1.bam file and “pipe” (remember “|”?) it as input to *ivar.* With *ivar*, we’re going to use the command *consensus* to call consensus with the following parameters:

There are five parameters that can be set in *ivar*:

- Minimum base quality
    - Default: 20 – probably an okay default!
    - Minimum quality is the minimum quality of a base to be considered in calculations of variant frequencies at a given position.
- Minimum frequency threshold
    - Default: 0 – so remember to set!
    - Minimum frequency threshold is the minimum frequency that a base must match to be called as the consensus base at a position. If one base is not enough to match a given frequency, then an ambiguous nucleotide is called at that position.
- Minimum depth to call a consensus
    - Default: 10 – okay for Illumina!
    - Minimum depth is the minimum required depth to call a consensus.
- A flag to exclude nucleotides from regions with depth less than the minimum depth
- A character to call in regions with coverage lower than the specified minimum depth
    - Default: 'N'.

Reminder of the parameters we’ll use:

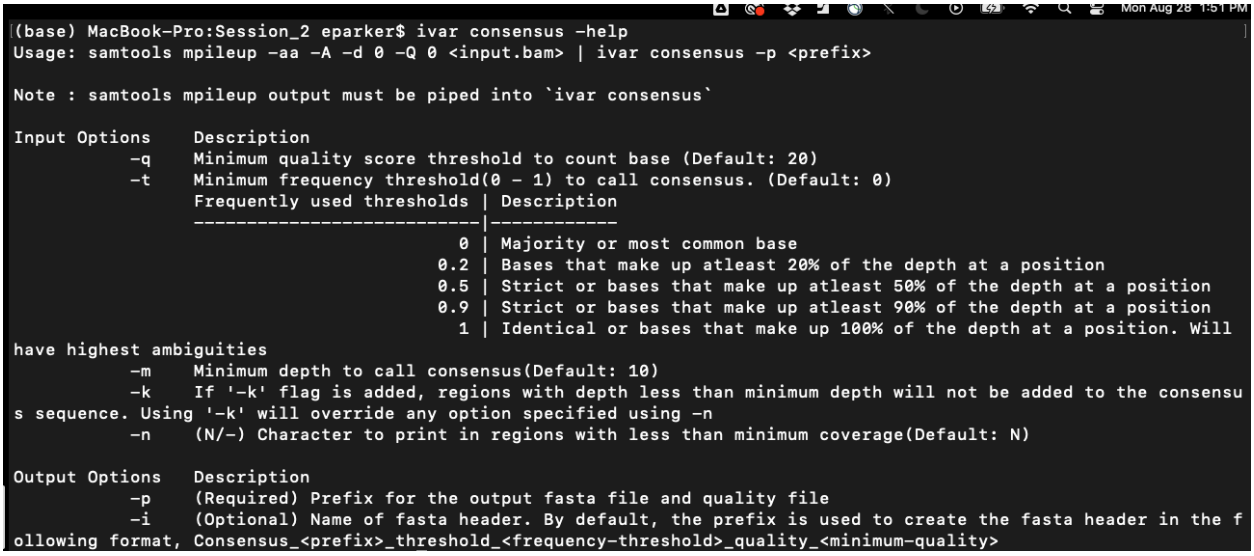
```
samtools mpileup -A -d 0 -Q 0 -B  S1.bam | ivar consensus -p S1 -t 0.5 -m 10
```

- “-p S1” is the prefix for our output file – which will be saved as S1.fasta.
- “-t 0.5” – we’re setting the minimum frequency threshold to call consensus to 0.5 Aka a 50% frequency threshold.
- “-m 10” – we’re setting the minimum read depth to call consensus to 10 (the default, but just to emphasize how you can tweak the parameter).

We’re going to keep the rest at default, so that the ambiguous nucleotide is “N”. Some pipelines insert the character “-” – but this is wrong! Can you think of why? What else does “-” represent in sequences?

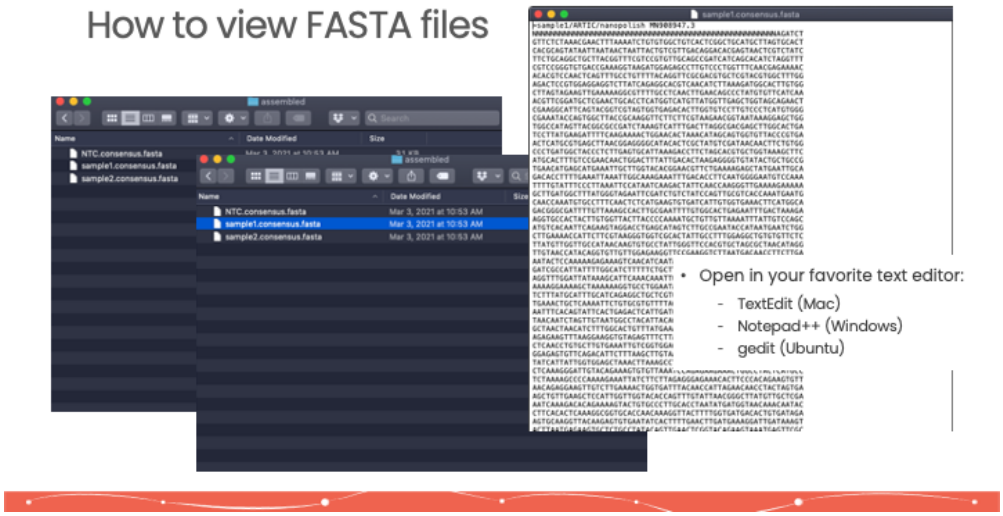
Remember, when we don't know what a tools or command does, we type:

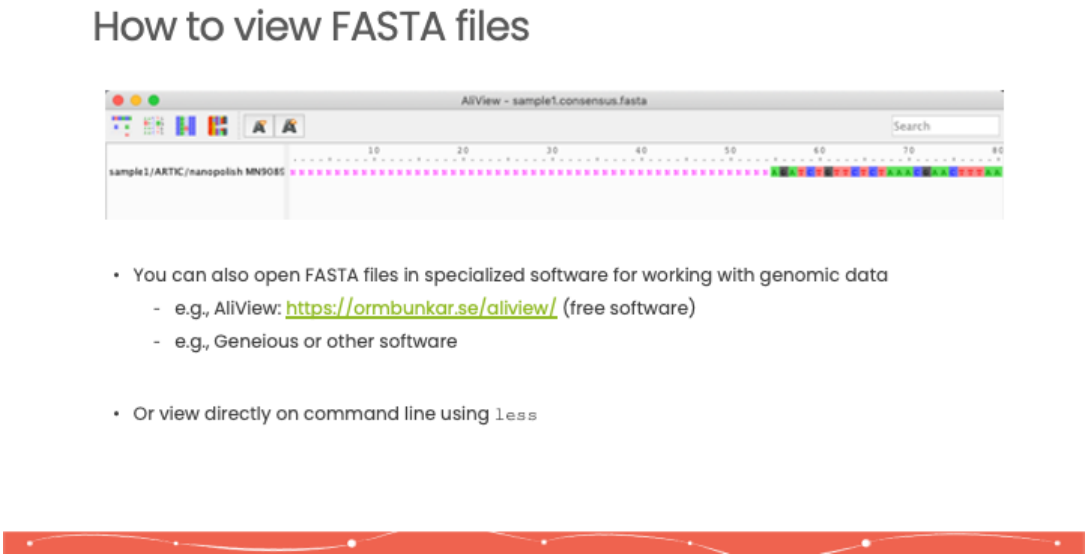
```
ivar consensus -help
```



We should now have a file called S1.fa in our folder – this is our consensus sequence! We also have a text file that has the quality score associated with each base we used to call consensus.

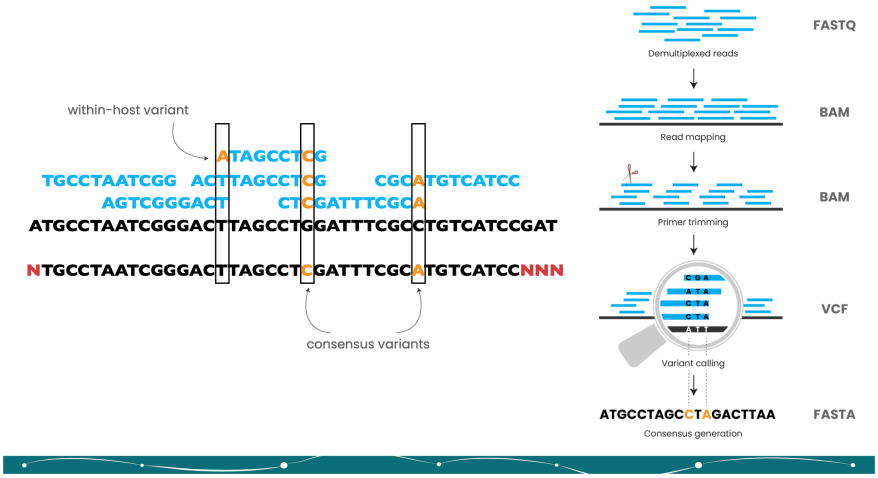
As you know, we can open the fasta in Aliview or in a text editor.





8.Calling variant

The final step in our pipeline is calling variants – this will tell us if there's any single nucleotide variants (SNVs) relative to the reference genome in our BAM, including at a sub-consensus level! We can have consensus level variants in our consensus genome that differ from the reference genome (i.e. variant relative to the reference), but we can also have within-host, minor variants that exist at frequencies sub-consensus level in a few reads that represent true pathogen diversity evolving within-host, e.g.:



We can also call indels – insertions or deletions relative to the reference genome.

There are a number of tools we can use to call variants, but we'll again use a combination of *samtools* and *ivar.* Our input here is our mapped BAM file and our reference genome it is mapped against, and our output will be a tab-delimited file (TSV, not CSV!) with all variants by position, e.g.:

Step	Read quality	Read filtering	Read mapping	Coverage	Consensus calling	Variant calling
Tool	Fastqc	Trimmomatic	Bwa + Samtools	Samtools	Samtools + iVar	Samtools + iVar
Input	R1.fastq R2.fastq	R1.fastq R2.fastq	R1_trim.fastq R2_trim.fastq	Bam	Bam	Bam + reference
Output	R1.html R2.html	R1_trim.fastq R2_trim.fastq	Bam	CSV	Fasta	VCF/ CSV

Let's break down the command again (don't run for now!):

```
samtools mpileup -aa -A  -B -Q 0 S1.bam  | ivar variants  -p S1_var -t 0.03 -m 10 -r NC_063383.fasta
```

Again, in the first part of the command, we're creating a pileup formatted file with *samtools* – as we did when calling consensus.

```
samtools mpileup -aa -A  -B -Q 0 S1.bam
```

- "-aa" tells *samtools* to output all positions, including unused reference sequences.
- "-A" tells *samtools* to not skip anomalous read pairs in variant calling.
- "-B" disables the base alignment quality computation
- "-Q 0" sets the minimum base quality for a base to be considered to 0. Note base-quality 0 is used as a filtering mechanism for overlap removal which marks bases as having quality zero and lets the base quality filter remove them.

Again, the manual has additional parameters and explanations you'd want to explore: http://www.htslib.org/doc/samtools-mpileup.html

Then we're "piping" the pileup format to *ivar* for variant calling:

```
samtools mpileup -aa -A  -B -Q 0 S1.bam | ivar variants  -p S1_var -t 0.03 -m 10 -r NC_063383.fasta
```

As with consensus calling, there are a few parameters we need to set for *ivar*.

- Minimum quality
    - Default: 20 − decent default!
    - The minimum quality for a base to be counted towards the ungapped depth to calculate iSNV frequency at a given position.
    - For insertions, the quality metric is discarded and the mpileup depth is used directly.
- Minimum frequency
    - Default: 0.03 − what is a good frequency threshold for minor variants…
    - The minimum frequency required for a SNV or indel to be reported.

The parameters we'll set for *ivar:*

- "-r NC_063383.fasta" – we need to pass *ivar* the reference that our reads are mapped to in the BAM.
- "-p" – the prefix for the file name of the output e.g. "S1_var.tsv"
- "-t 0.03" – We're setting the minimum frequency to call variants to 0.03 – i.e. any variant at 3% of reads at a position will be reported in our output. Is this an appropriate threshold?
- "-m 10" – We're setting the minimum read depth to call convariants to 10 (the default, but just as emphasis)
- We'll keep the minimum base quality score to 20
    - The default, so no need to specify

This command might take some time, depending on how many reads we have.

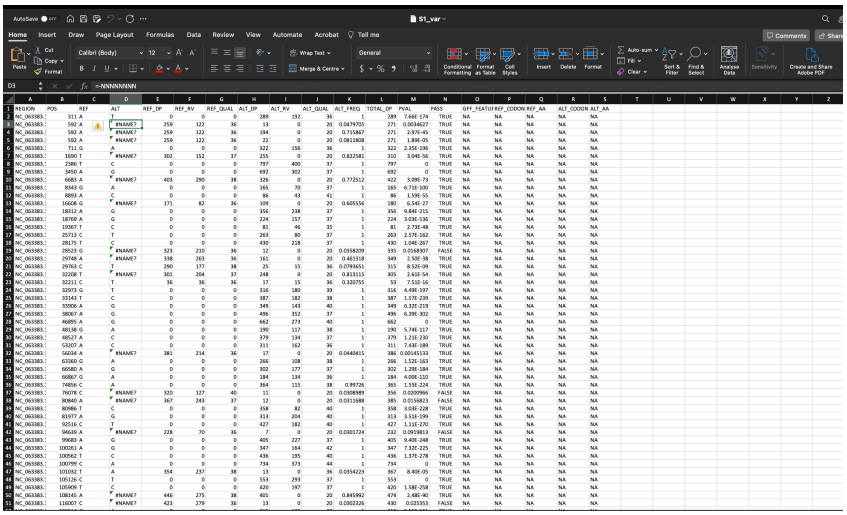
Now, let's take a look at the output! We'll generate a tab-delimited file we can open in Excel. It'll look a little like:

REGION	POS	REF	ALT	REF_DP	REF_RV	REF_QUAL	ALT_DP	ALT_RV	ALT_QUAL	ALT_FREQ	TOTAL_DP	PVAL	PASS	GFF_FEATURE	REF_CODON	REF_AA	ALT_CODON	ALT_AA
test	42	G	T	0	0	0	1	0	49	1	1	1	FALSE	id-test3		AGG	R	ATG	M
test	42	G	T	0	0	0	1	0	49	1	1	1	FALSE	id-test4		CAG	Q	CAT	H
test	320	A	T	1	1	35	1	1	46	0.5	2	0.666667	FALSE	NA	NA	NA	NA	NA	NA
test	365	A	T	0	0	0	1	1	27	1	1	1	FALSE	NA	NA	NA	NA	NA	

Here is the description of the columns (also see: https://andersen-lab.github.io/ivar/html/manualpage.html)

Field	Description
REGION	Region from BAM file
POS	Position on reference sequence
REF	Reference base
ALT	Alternate Base
REF_DP	Ungapped depth of reference base
REF_RV	Ungapped depth of reference base on reverse reads
REF_QUAL	Mean quality of reference base
ALT_DP	Ungapped depth of alternate base.
ALT_RV	Ungapped deapth of alternate base on reverse reads
ALT_QUAL	Mean quality of alternate base
ALT_FREQ	Frequency of alternate base
TOTAL_DP	Total depth at position
PVAL	p-value of fisher's exact test
PASS	Result of p-value <= 0.05
GFF_FEATURE	ID of the GFF feature used for the translation
REF_CODON	Codong using the reference base
REF_AA	Amino acid translated from reference codon
ALT_CODON	Codon using the alternate base
ALT_AA	Amino acid translated from the alternate codon

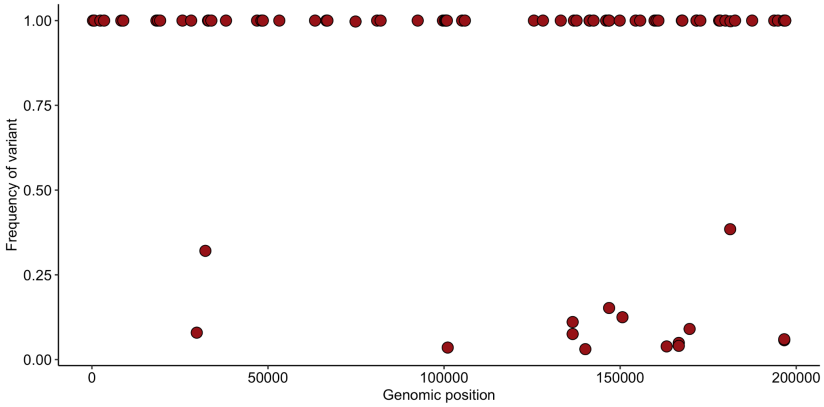
When we open our file "S1_var.tsv", it'll look a little like:



Based on the ALT_FREQ (alternative frequency) column, we can see that we have a few consensus level variants relative to the reference genome (we expect all positions with a frequency > 0.5 to be included in the consensus genome, right? But we can also see that we have a few minor variants at e.g. 4% of reads.

Importantly, in the ALT column, we can see that we have some stretches of “N”s compared to the reference. You can see my Excel fails to read them correctly – also the indels can’t be read (insertions starts with a +, or - for deletions) – but click on the cell to see the text!

We can plot the variants to see their distribution across the genome. There should be an R script in the dropbox called Var_plot.R, but I’m sure you can figure out how to plot it in Excel too!



Here we can see we have quite a few “minor” or sub-consensus variants – can you identify them in the TSV?

Do we think these represent true diversity? How do we consider this?

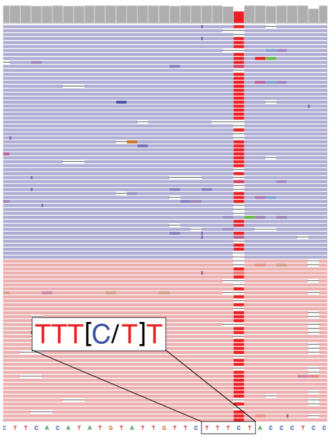
Examining new or suspicious mutations

ATGCCAAATCGGGACTTAGCCTGGATTTCGCCGGTTACCCGAT

- Examine new mutations (not seen before during a particular outbreak)
- Examine regions of the genome with many variants close together
- Examine any frameshifting mutations

- To explore mutations, visualize reads in **BAM** file

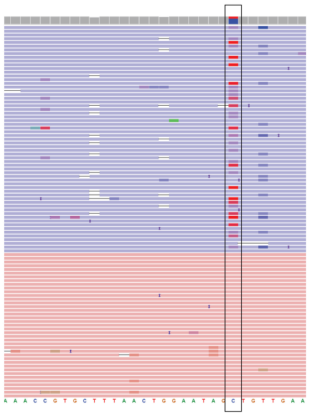
Examining new or suspicious mutations



- Mutations in homopolymer regions often represent sequencing errors

- Spurious one-base insertions and deletions in homopolymer regions are common errors

Examining new or suspicious mutations

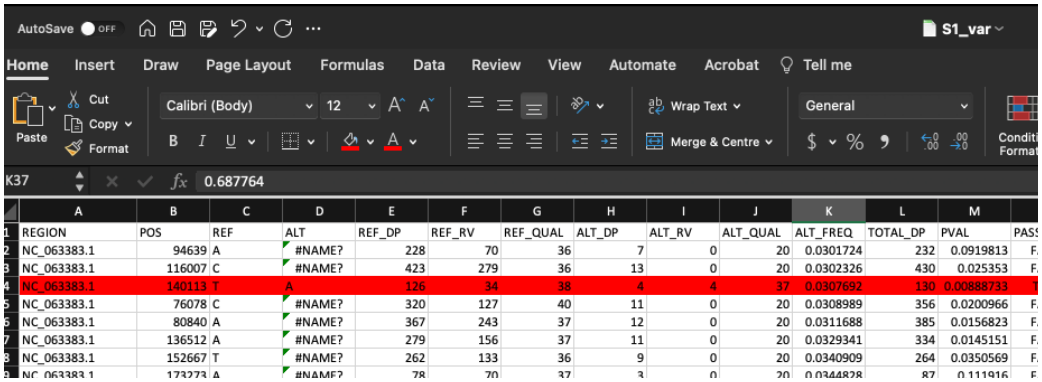


- Note any mutations that occur on only the forward or reverse reads – these are likely errors

- Most variant calling software check for strand bias, but always validate new or odd mutations

We can go look at any suspicious positions in our BAM in IGV, as mentioned previously. Remember, you need to load the indexed fasta and indexed BAM for IGV (see earlier section).

Let's look at this variant:

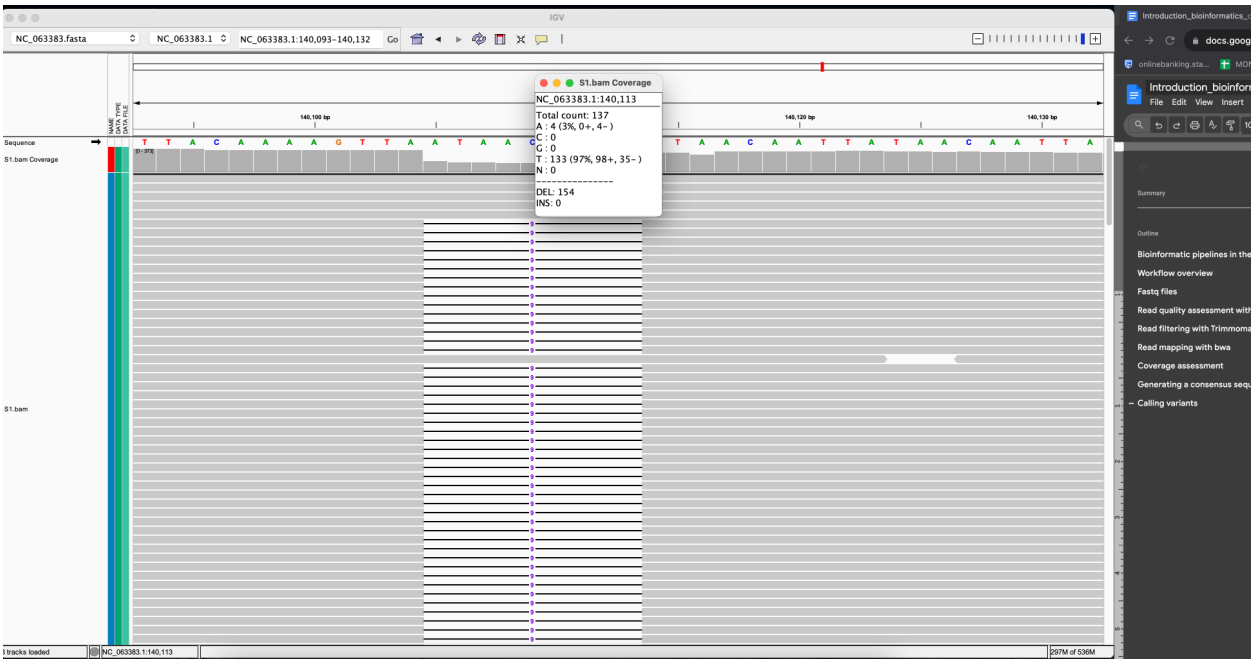


	A	B	C	D	E	F	G	H	I	J	K	L	M	
1	REGION	POS	REF	ALT	REF_DP	REF_RV	REF_QUAL	ALT_DP	ALT_RV	ALT_QUAL	ALT_FREQ	TOTAL_DP	PVAL	PASS
2	NC_063383.1	94639	A	#NAME?	228	70	36	7	0	20	0.0301724	232	0.0919813	F
3	NC_063383.1	116007	C	#NAME?	423	279	36	13	0	20	0.0302326	430	0.025353	F
4	NC_063383.1	140113	T	A	126	34	38	4	4	37	0.0307692	130	0.00888733	T
5	NC_063383.1	76078	C	#NAME?	320	127	40	11	0	20	0.0308989	356	0.0200966	F
6	NC_063383.1	80840	A	#NAME?	367	243	37	12	0	20	0.0311688	385	0.0156823	F
7	NC_063383.1	136512	A	#NAME?	279	156	37	11	0	20	0.0329341	334	0.0145151	F
8	NC_063383.1	152667	T	#NAME?	262	133	36	9	0	20	0.0340909	264	0.0350569	F
9	NC_063383.1	173273	A	#NAME?	78	70	37	3	0	20	0.0344828	87	0.111916	F

We can go to the position of the variant in the genome by going to the co-ordinates up top:

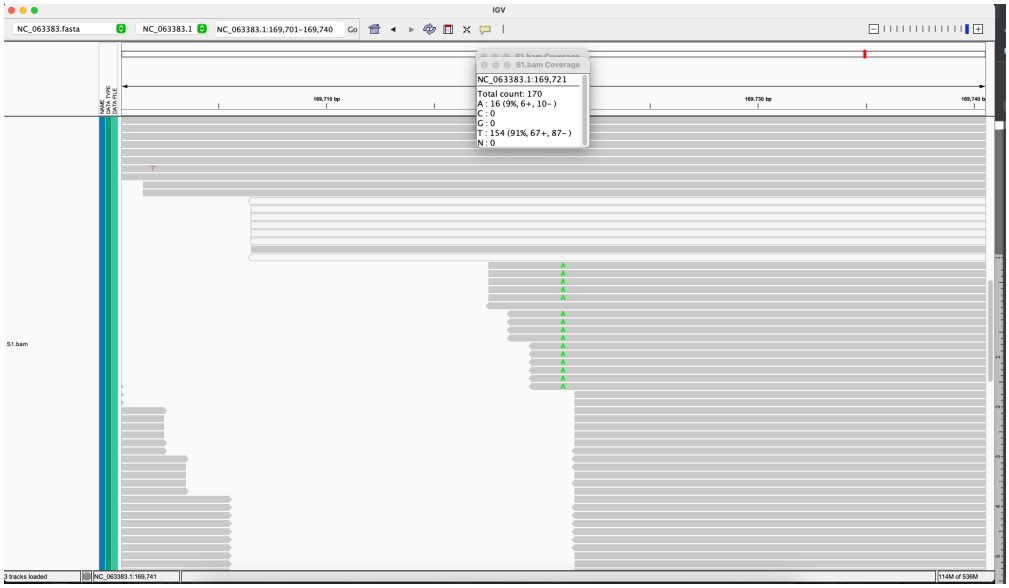
And typing the co-ordinate:

We can click on the Histogram for that position in the coverage track. We have 137 reads that cover that position (so we pass the threshold!). In 133 reads, we have a T there, and we have about 4 A's. So, pretty well supported as a T at that position. But, there's also something strange happening before that region… Do you think this variant is real?



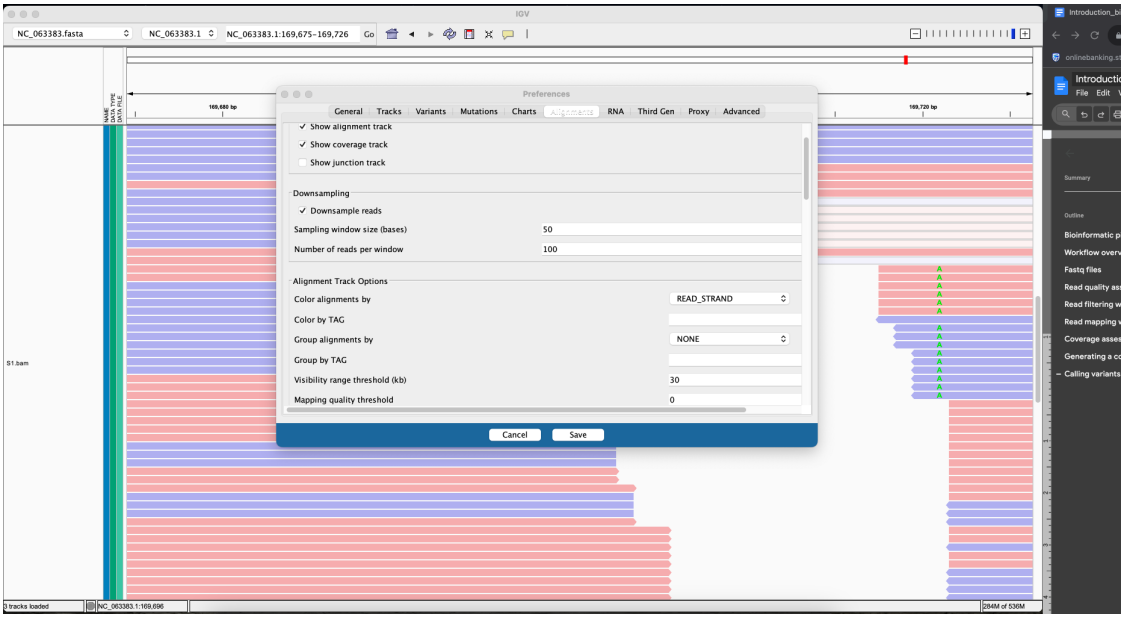
Let's try another one: position 169 721.

The variant (A – 9% of reads) of that position sits at the start of the read – do we think the minor variant is likely true?

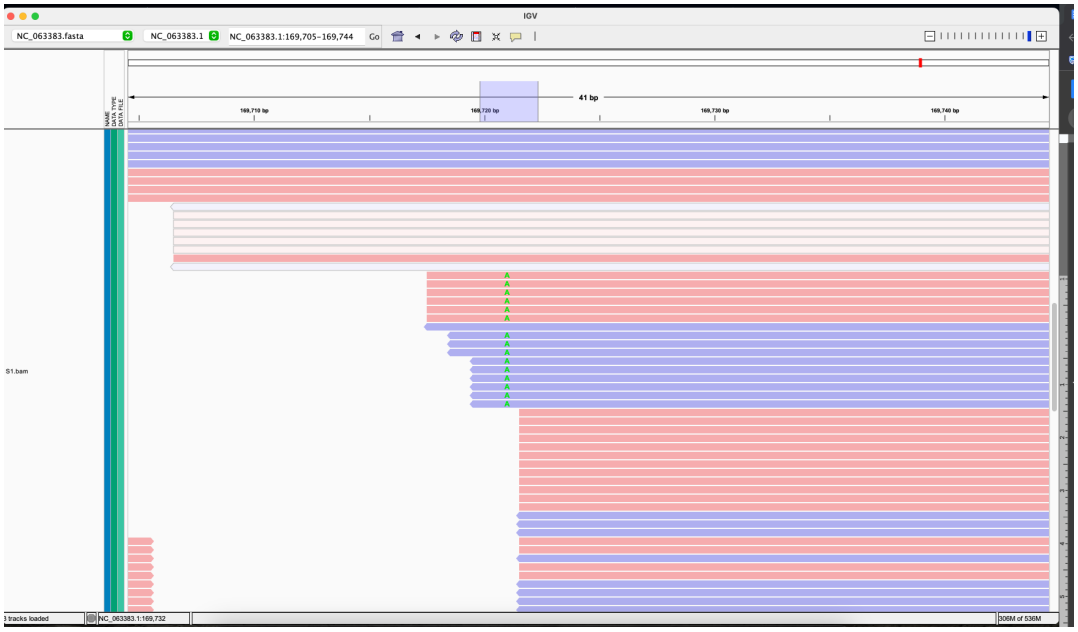


Let's see if it's on the forward and reverse reads – remember, variants present on both are less likely to be a sequencing error, that's why we perform paired end sequencing. But what is a reverse or forward read here?

First, let's go to View and toggle the read colour:



So is the variant A present on both forward and reverse reads (in different colours)? Do we trust the variant?



As discussed, short insertions or deletions in homopolymer stretches are often artefactual, so if we ever see any of those in our consensus sequence – especially if they introduce a fameshift – we can open our BAM in IGV and look at the surrounding regions to see if the indels are in a homopolymer stretch. We can also see if the indel is the middle of the read – less likely to be an artifact, or towards the beginning or end of reads, which is more likely artefactual!

So, in summary, BAM diagnoses in IGV is a very good idea for anything that looks a little weird in your consensus genome!

Let's clean up a bit again:

```
mkdir output
```

```
mv S1* output/
```

```
mv Variants.png output/
```

```
mv NC_063383.fasta* reference/
```

```
mv Var_plot.R output/
```

9. Within-host diversity variants

If we're interested in the within-host diversity of our sample, we can already see the minor alleles present relative to the reference from the variant calling step just above. However, we can also map our reads against our own consensus sequence as a reference and call variants on that – that should give us a variant file of all the minor variants relative to our own consensus!

We've done these operations before, so let's see if we can remember how to 1) map reads to a reference (now our own consensus sequence) and 2) call variants on that new mapped BAM!

We first need to retrieve our 1) paired, trimmed reads in R1.fastq and R2.fastq and 2) Our consensus genome, to map against.

We can retrieve our fastqs:

```
cd fastqs
```

```
mv S1_R*_001_P.fastq.gz ../
```

```
cd ../
```

```
gunzip *
```

And our consensus sequence is here:

```
cd output/
```

```
cp S1.fa ../
```

```
cd ../
```

Now, let's map our reads to our consensus – we'll do this like we generated the BAM mapped to reference in our previous section – so go back if you need to!

We'll again map with *bwa mem*! But what do we need to do first? We need to index the reference genome – which is now our consensus!

```
bwa index S1.fa
```

Remember: we need to keep these index files in the directory we're performing our operations in!

```
-bash: bwa: command not found
(base) MacBook-Pro:Session_2 eparker$ /Applications/bwa/bwa index S1.fa
[bwa_index] Pack FASTA... 0.00 sec
[bwa_index] Construct BWT for the packed sequence...
[bwa_index] 0.02 seconds elapse.
[bwa_index] Update BWT... 0.00 sec
[bwa_index] Pack forward-only FASTA... 0.00 sec
[bwa_index] Construct SA from BWT and Occ... 0.01 sec
[main] Version: 0.7.17-r1198-dirty
[main] CMD: /Applications/bwa/bwa index S1.fa
[main] Real time: 0.037 sec; CPU: 0.040 sec
(base) MacBook-Pro:Session_2 eparker$ ls
S1.fa      S1.fa.ann  S1.fa.pac  S1_R1_001_P.fastq  coverage       fastqs         output
S1.fa.amb  S1.fa.bwt  S1.fa.sa   S1_R2_001_P.fastq  fastqc_report  intermediates  reference
```

Now we can map the paired and trimmed reads (the fastqs) to our consensus sequence (S1.fa), to create a new BAM called S1_selfmap.bam (or whatever you want to call it!).

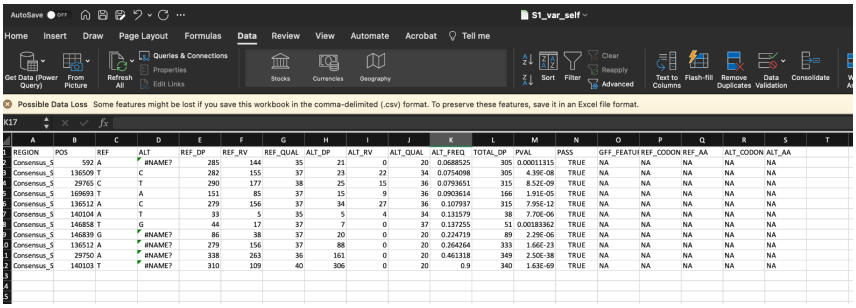
Do you remember what all of these arguments mean? Go back to the previous section if need be!

```
bwa mem -t 3 S1.fa S1_R1_001_P.fastq S1_R2_001_P.fastq | samtools view -u -@ 3 - | samtools sort -@ 3 -o S1_selfmap.bam
```

Now we have a BAM with our reads mapped to our consensus, so we can investigate sub-consensus variation by calling variants again on this BAM! Let's call our new variant file something different (S1_var_self.tsv) to distinguish it from the other variant file.

```
samtools mpileup -aa -A  -B -Q 0 S1_selfmap.bam  | ivar variants  -p S1_var_self -t 0.05 -m 10 -r S1.fa
```

Now, we have a file with our minor variant at >5% frequency in our sample!



REGION	POS	REF	ALT	REF_DP	REF_RV	REF_QUAL	ALT_DP	ALT_RV	ALT_QUAL	ALT_FREQ	TOTAL_DP	PVAL	PASS	GFF_FEATURE	REF_CODON	REF_AA	ALT_CODON	ALT_AA
Consensus_S	592	A	#NAME?	285	144	35	21	0	20	0.0688525	305	0.00011315	TRUE	NA	NA	NA	NA	NA
Consensus_S	136509	T	C	282	155	37	23	22	34	0.0754098	305	4.39E-08	TRUE	NA	NA	NA	NA	NA
Consensus_S	29765	C	T	290	177	38	25	15	36	0.0793651	315	8.52E-09	TRUE	NA	NA	NA	NA	NA
Consensus_S	169693	T	A	151	85	37	15	9	36	0.0903614	166	1.91E-05	TRUE	NA	NA	NA	NA	NA
Consensus_S	136512	A	C	279	156	37	34	27	36	0.107937	315	7.95E-12	TRUE	NA	NA	NA	NA	NA
Consensus_S	140104	A	T	33	5	35	5	4	34	0.131579	38	7.70E-06	TRUE	NA	NA	NA	NA	NA
Consensus_S	146858	T	G	44	17	37	7	0	37	0.137255	51	0.00183362	TRUE	NA	NA	NA	NA	NA
Consensus_S	146839	G	#NAME?	86	38	37	20	0	20	0.224719	89	2.29E-06	TRUE	NA	NA	NA	NA	NA
Consensus_S	136512	A	#NAME?	279	156	37	88	0	20	0.264264	333	1.66E-23	TRUE	NA	NA	NA	NA	NA
Consensus_S	29750	A	#NAME?	338	263	36	161	0	20	0.461318	349	2.50E-38	TRUE	NA	NA	NA	NA	NA
Consensus_S	140103	T	#NAME?	310	109	40	306	0	20	0.9	340	1.63E-69	TRUE	NA	NA	NA	NA	NA

Now we can investigate these variants! We need to ask ourselves a few questions:

1) How much within-host diversity do we expect, if we believe all variants are real? This is going to be very different for different pathogens, different hosts, different lengths of infection (e.g. acute infection vs chronic) etc
2) Do we expect more within-host diversity in certain genomic regions depending on pathogen or host factors or selective forces?
    a) Can we plot the positions against the genome organization?
3) Is there any reason for us to suspect contamination? Contamination will present as mixed populations (e.g. 20% A, 30% T, 50% G at a site) across a few sites depending on the sequencing prep and protocol e.g. amplicon vs metagenomics.
    a) So if there's contamination, we might expect to see a lot of variants at sub-consensus level!
    b) If we suspect contamination and want to see if the variants are e.g. on both reads, we can open our BAM with IGV and investigate as we did previously
    c) We should also check the coverage at these positions to see how well supported things are!
4) Are there any indels sub-consensus?
    a) As discussed, short indels in homopolymer stretches are often artefactual, so we can open our BAM in IGV and look at the surrounding genomic region, and where the indel falls in the reads

Finally, we should clean up a bit:

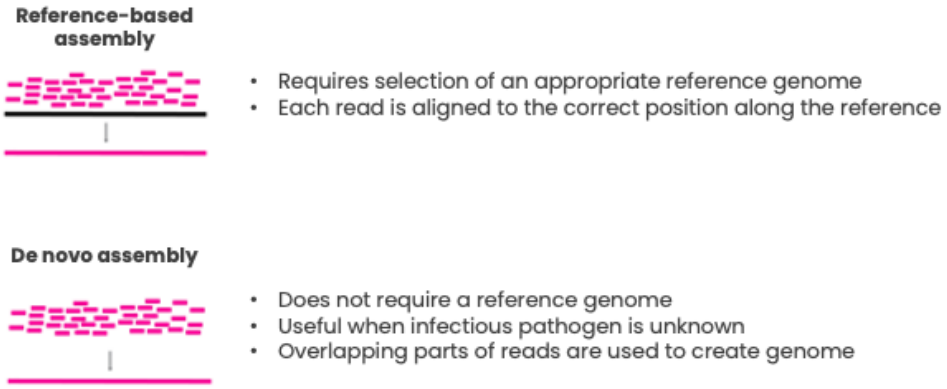
```
mv S1.fa.* reference/
mv S1_var_self.tsv output/
mv S1_selfmap.bam output/
```

10. De novo assembly (with human read depletion)

As you discussed, reference-based assembly is not appropriate when we don't know the target organism e.g. if we're doing metagenomics of fevers of unknown origin, if there is no decent reference sequence available (if its a suspected novel pathogen or something that's not well sequenced) or if we suspect our pathogen might be very divergent from the reference (e.g. a new lineage of LASV in a country previously unsampled for LASV, or if we're not sure if its Mpox Clade 2 and Clade 1 in countries like Cameroon where we know there may be overlap!).

You've performed *de novo* assembly in Terra, so you should be acquainted with the underlying concept. And, as we have metagenomic sequencing of Mpox from diverse countries, it may be best to perform *de novo* assembly if we're not sure if it's Clade 1 or Clade 2. Again, we can map against references from both clades and see what generates a full genome too.

As a recap:



De novo assembly is a little more computationally intensive, especially as we have a lot of reads to work with. Remember, we performed metagenomic sequencing, not targeted sequencing! So not every read in the fastqs is from the Mpox virus. There will be host reads and reads from potential co-infections, the microbiome etc. You can of course deplete human reads in the lab; but we can also perform a step to deplete host e.g. human reads in our pipelines to save memory and space, as you did in Terra. Otherwise we spend a lot of our computational time assembling contigs or contiguous sequences from the host reads – we'll get large human contigs out of our *de novo* assembler!

To deplete host reads, we'll simply map reads against the human genome and then remove the reads that map to the human genome. I say "simply" because it requires us to download the human genome and its indices (which is 3.75 Gigabytes compressed…). So, I'm not sure we can all attempt that right now on a shared internet bandwidth. Also, we don't technically have to deplete human reads before *de novo* assembly – we'll just assemble some human contigs then too.

But let's take a look at how we'd run this step if we wanted to:

First, we'll download the human genome and its indexes from the internet. We'll unzip the folder, and then move its content to the directory we're working in – we need the indices in the same directory we're working in.

```
wget -c https://genome-idx.s3.amazonaws.com/bt/GRCh38_noalt_as.zip
```

```
unzip GRCh38_noalt_as.zip
```

```
cd GRCh38_noalt_as
```

```
mv * ../
```

```
cd ../
```

We'll see a few files there, all with the format GRCh38_noalt_as* – these are our human reference indices!

Now, we'll map our reads (S1_R1_001_P.fastq and S1_R2_001_P.fastq) against the human genome. We'll use a new tool called *bowtie2* – this is also a read mapper, like *bwa*. I use *bowtie2* for this step because of its *--un-conc-gz* function, but you can perform this with any read mapper of your choice.

We can install it via Anaconda, as usual!

```
conda install -c bioconda bowtie2
```

We'll try (if you can't download the human genome indices for now, don't worry):

```
bowtie2 -p 4 -x GRCh38_noalt_as -1 S1_R1_001_P.fastq -2 S1_R2_001_P.fastq  --un-conc-gz S1_human_depleted > S1_mapped_and_unmapped.sam
```

The *bowtie* manual is here: https://bowtie-bio.sourceforge.net/bowtie2/manual.shtml#output-options

Let's see the arguments to *bowtie2*:

- "-p 4" is telling *bowtie2* to use four processors – remember, you should adapt to your machine (and definitely increase if you're on a HPC/server!)
- "-x GRCh38_noalt_as" is the basename of our index for the reference genome.
- "-1" and "-2" are our read files (R1 and R2).
- "--un-conc-gz" – this command writes paired-end reads that fail to align concordantly to our reference to the path that follows e.g. "S1_human_depleted"
- "> S1_mapped_and_unmapped.sam" redirects the reads to a SAM file.

Our output should look a little like:

```
(Mpox) MacBook-Pro:Session_2 eparker$ bowtie2 -p 4 -x GRCh38_noalt_as -1 S1_R1_001_P.fastq -2 S1
_R2_001_P.fastq  --un-conc-gz S1_human_depleted > S1_mapped_and_unmapped.sam
416683 reads; of these:
  416683 (100.00%) were paired; of these:
    414287 (99.42%) aligned concordantly 0 times
    363 (0.09%) aligned concordantly exactly 1 time
    2033 (0.49%) aligned concordantly >1 times
    ----
    414287 pairs aligned concordantly 0 times; of these:
      160 (0.04%) aligned discordantly 1 time
    ----
    414127 pairs aligned 0 times concordantly or discordantly; of these:
      828254 mates make up the pairs; of these:
        825674 (99.69%) aligned 0 times
        1148 (0.14%) aligned exactly 1 time
        1432 (0.17%) aligned >1 times
0.92% overall alignment rate
(Mpox) MacBook-Pro:Session_2 eparker$ ls
GRCh38_noalt_as                 S1.fa.ann                       S1_selfmap.bam
GRCh38_noalt_as.1.bt2           S1.fa.bwt                       S1_var_self.tsv
GRCh38_noalt_as.2.bt2           S1.fa.fai                       coverage
GRCh38_noalt_as.3.bt2           S1.fa.pac                       fastqc_report
GRCh38_noalt_as.4.bt2           S1.fa.sa                        fastqs
GRCh38_noalt_as.rev.1.bt2       S1_R1_001_P.fastq               intermediates
GRCh38_noalt_as.rev.2.bt2       S1_R2_001_P.fastq               metaspades_out
GRCh38_noalt_as.zip             S1_human_depleted.1             output
S1.fa                           S1_human_depleted.2             reference
S1.fa.amb                       S1_mapped_and_unmapped.sam
```

We can see our files of interest are now S1_human_depleted.1 and S1_human_depleted.2 – let's rename them to show they're fastqs (and they're compressed as .gz files)!

```
mv S1_human_depleted.1 S1_human_depleted_R1.fastq.gz
```

```
mv S2_human_depleted.1 S1_human_depleted_R2.fastq.gz
```

Now, we have read files without human reads in there to serve as input to our *de novo* assembler.

But let's tidy up first! Let's compress our original read files and move them back to their directory:

```
gzip S1_R*_001_P.fastq
```

```
mv S1_R*_001_P.fastq.gz fastqs
```

Let's move our human genome reference back to our original folder and actually delete that folder – we still have the compressed .zip file, so can just unzip that if we need it again. Note: I'm just doing this to free up space on your machines for now! If you're running multiple samples in a row, and you want to deplete each, you'll need the indices and should not delete them, but save them somewhere accessible to all samples!

```
mv GRCh38_noalt_as*.bt* GRCh38_noalt_as
```

```
rm -r GRCh38_noalt_as
```

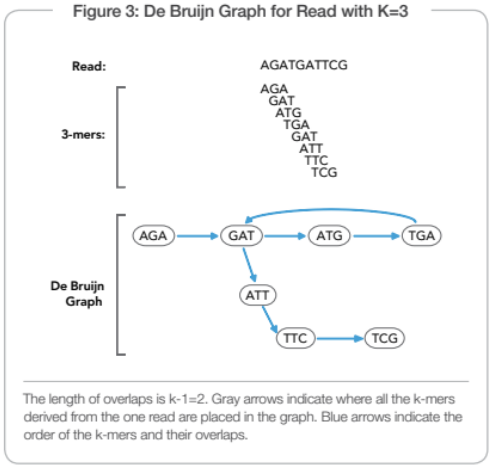
We're also not going to use that new SAM, so let's delete it for now:

```
rm S1_mapped_and_unmapped.sam
```

Okay, now we have host depleted, paired and trimmed read files ready for *de novo* assembly.

As always, there are many tools to perform *de novo* assembly, but we'll use a tool called *spades:* https://github.com/ablab/spades#sec2.1. Spades is an iterative short-read genome assembly tool that works with something called de Bruijn graphs. To borrow from:https://www.illumina.com/Documents/products/technotes/technote_denovo_assembly_ecoli.pdf

"De Bruijn graphs reduce the computational effort by breaking reads into smaller sequences of DNA, called k-mers, where the parameter k denotes the length in bases of these sequences. The de Bruijn graph captures overlaps of length k-1 between these k-mers and not between the actual reads. By reducing the entire data set down to k-mer overlaps the de Bruijn graph reduces the high redundancy in short-read data sets. The maximum efficient k-mer size for a particular assembly is determined by the read length as well as the error rate. The value of the parameter k has a significant influence on the quality of the assembly."



The great thing about *Spades* is that it can automatically select the value of *K (for the k-mer)* based on read length and data set.

Here is a few citations to help you understand:
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3342519/ and
https://genome.cshlp.org/content/27/5/824.short.

We can install *Spades* from conda, but I think we should rather install it from the source as my conda install threw up a few conflict:We can follow the download instructions here if we haven't already: https://github.com/ablab/spades#sec2.1

E.g., if you're on a Linux system, you can run:

```
wget -c http://cab.spbu.ru/files/release3.15.5/SPAdes-3.15.5-Linux.tar.gz
```

```
tar -xzf SPAdes-3.15.5-Linux.tar.gz
```

```
cd SPAdes-3.15.5-Linux/bin/
```

Our script spades.py should be in the folder SPAdes-3.15.5-Linux/bin/. We should export this folder to our path, so we can run it from anywhere in our system. Remember our path is basically a list of addresses where our terminal will go looking for the program to run!

So, when you're in the folder SPAdes-3.15.5-Linux/bin/ type:

```
pwd
```

This should print out the full address of the folder e.g. ~/Downloads/SPAdes-3.15.5-Linux/bin/ .

Now, we need to export that to our PATH:

```
export PATH=$PATH:~/Downloads/SPAdes-3.15.5-Linux/bin/
```

It should be in our PATH now; you can check with

```
echo $PATH
```

If we don't see the address there, we can technically hard code the PATH e.g. run spades as:

```
~/Downloads/SPAdes-3.15.5-Linux/bin/spades.py –help
```

This way, spades.py is not in our path, but we're explicitly telling the terminal where it can be found. If, however, we want to make sure it's always in our PATH, we can change our bash_profile – this is basically the settings your terminal loads whenever it restarts. If we change the path PATH, it'll always add *spades* to the PATH.

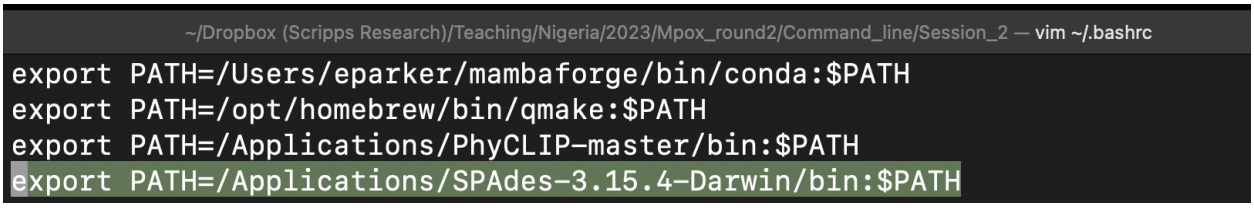
Don't worry about this for now, but for future use: To change this file, we need to open a hidden file named ~/.bashrc (it should be in your home directory). If you're not sure how to do this, it might be worth not playing around with it and just hard coding the spades.py path as e.g. ~/Downloads/SPAdes-3.15.5-Linux/bin/spades.py when you run it. Otherwise you can open the bash profile with

```
vim ~/.bashrc
```

And then you can copy and paste the address to spades.py in there e.g. export PATH=$PATH:~/Downloads/SPAdes-3.15.5-Linux/bin/. Remember, this should be the output of the pwd command above – your *spades* directory may have been installed somewhere else!

Vim is a little complicated, so press "**I**" (capital i) to activate input mode. Then you can paste it in there. Make sure it is on one line on its own, so enter the next line. Then press "**Escape**", then "**:**" then "**x**" then "**Enter**" – without the quotations, so just the characters/keys. This should save the changes made.

Here's how my ~/.bashrc looks like, with the spades directory added to my path.



Now, I need to reload that bash_profile so my current terminal has all those paths too.

```
source ~/.bashrc
```

If you can't figure this out, don't worry! Just hard code your path to spades.

Okay end of that homework section!

So, back to assembly! We're going to try:

```
/Applications/spades-spades_3.15.4/assembler/spades.py -o S1_denovo -1 S1_human_depleted_R1.fastq.gz -2 S1_human_depleted_R2.fastq.gz -t 4 --cov-cutoff 10
```

So here "/Applications/spades-spades_3.15.4/assembler/spades.py" is either "spades.py" if you added it to your PATH, or it's the full address to your spades.py file e.g. ~/Downloads/SPAdes-3.15.5-Linux/bin/spades.py. Also remember, your *Spades* version might be different if you're on a Mac/Linux! So make sure you know what your folder names are, don't just copy and paste mine here!

Let's break down the command – but definitely take a look at the manual (https://github.com/ablab/spades#sec2.1) as there may be other parameters you want to play around with, including *metaspades*, which is specifically for metagenomics:

- "-o" here is the name of our output folder, which I called S1_denovo/
- "-1" and "-2" are our paired end reads – R1 and R2 respectively. Remember we're using the fastqs depleted of human reads.
- "-t 4" is telling *Spades* to use 4 threads or processors
- "--cov-cutoff 10" is the read coverage cutoff value.

We're not going to give spades the K-mer size, as we'll allow it to tune it automatically (see above). You'll see *spades* attempt different K's in the output!

Running this on my Mac on 4 threads took about ten minutes – this may take a lot longer on your local machines – but can be sped up on HPC with more cores!

Our output should look a little like:

```
(base) MacBook-Pro:Session_2 eparker$ /Applications/spades-spades_3.15.4/assembler/spades.py -o S1_denovo -1 S1_human_depleted_R1.fastq.gz -2 S1_human_depleted_R2.fastq.gz -t 4 --cov-cutoff 10


== Warning ==  output dir is not empty! Please, clean output directory before run.



== Warning ==  No assembly mode was specified! If you intend to assemble high-coverage multi-cell/isolate data, use '--isolate' option.


Command line: /Applications/spades-spades_3.15.4/assembler/spades.py    -o      /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo      -1      /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_human_depleted_R1.fastq.gz  -2      /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_human_depleted_R2.fastq.gz  -t      4       --cov-cutoff    10

System information:
  SPAdes version: 3.15.4
  Python version: 3.10.12
  OS: macOS-13.4.1-arm64-arm-64bit

Output dir: /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo
Mode: read error correction and assembling
Debug mode is turned OFF

Dataset parameters:
  Standard mode
  For multi-cell/isolate data we recommend to use '--isolate' option; for single-cell MDA data use '--sc'; for metagenomic data use '--meta'; for RNA-Seq use '--rna'.
  Reads:
    Library number: 1, library type: paired-end
      orientation: fr
      left reads: ['/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_human_depleted_R1.fastq.gz']
      right reads: ['/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_human_depleted_R2.fastq.gz']
      interlaced reads: not specified
      single reads: not specified
      merged reads: not specified
Read error correction parameters:
  Iterations: 1
  PHRED offset will be auto-detected
  Corrected reads will be compressed
Assembly parameters:
  k: automatic selection based on read length
  Repeat resolution is enabled
```

It'll start with read error correction – you can read more about that here: https://cab.spbu.ru/files/release3.15.2/manual.html

We then start the assembly with K21 – remember, this is the K-mer length (see above). We might want to set this to a longer length, like 55, but it's dependent on our data, and *spades* will run through a few K-mer lengths:

```
===== Assembling started.


===== K21 started.


== Running: /Applications/spades-spades_3.15.4/assembler/bin/spades-core /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/K21/configs/config.info

  0:00:00.000     1M / 8M    INFO    General                 (main.cpp                  :  99)   Loaded config from /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/K21/configs/config.info
  0:00:00.000     1M / 8M    WARN    General                 (memory_limit.cpp          :  52)   Failed to limit memory to 250 Gb, setrlimit(2) call failed, errno = 22 (Invalid argument). Watch your memory consumption!
  0:00:00.000     1M / 8M    INFO    General                 (main.cpp                  : 107)   Starting SPAdes, built from N/A, git revision N/A
  0:00:00.000     1M / 8M    INFO    General                 (main.cpp                  : 108)   Maximum k-mer length: 128
  0:00:00.000     1M / 8M    INFO    General                 (main.cpp                  : 109)   Assembling dataset (/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/dataset.info) with K=21
  0:00:00.000     1M / 8M    INFO    General                 (main.cpp                  : 110)   Maximum # of threads to use (adjusted due to OMP capabilities): 1
  0:00:00.000     1M / 8M    INFO    General                 (pipeline.cpp              : 212)   SPAdes started
  0:00:00.000     1M / 8M    INFO    General                 (pipeline.cpp              : 225)   Starting from stage: read_conversion
  0:00:00.000     1M / 8M    INFO    General                 (pipeline.cpp              : 231)   Two-step repeat resolution disabled
  0:00:00.000     1M / 8M    INFO    GraphCore               (graph_core.hpp            : 672)   Graph created, vertex min_id: 3, edge min_id: 3
  0:00:00.000     1M / 8M    INFO    GraphCore               (graph_core.hpp            : 673)   Vertex size: 48, edge size: 40
  0:00:00.000     1M / 8M    INFO    General                 (edge_index.hpp            : 115)   Size of edge index entries: 12/8
  0:00:00.000     1M / 9M    INFO    StageManager            (stage.cpp                 : 185)   STAGE == Binary Read Conversion (id: read_conversion)
  0:00:00.000     1M / 9M    INFO    General                 (read_converter.cpp        :  72)   Converting reads to binary format for library #0 (takes a while)
  0:00:00.000     1M / 9M    INFO    General                 (read_converter.cpp        :  73)   Converting paired reads
  0:00:00.042    22M / 22M   INFO    General                 (binary_converter.cpp      :  96)   16384 reads processed
  0:00:00.083    26M / 26M   INFO    General                 (binary_converter.cpp      :  96)   32768 reads processed
  0:00:00.216    22M / 30M   INFO    General                 (binary_converter.cpp      :  96)   65536 reads processed
  0:00:00.421    25M / 36M   INFO    General                 (binary_converter.cpp      :  96)   131072 reads processed
  0:00:00.865    21M / 36M   INFO    General                 (binary_converter.cpp      :  96)   262144 reads processed
  0:00:01.368    19M / 36M   INFO    General                 (binary_converter.cpp      : 111)   414180 reads written
  0:00:01.368     1M / 36M   INFO    General                 (read_converter.cpp        :  86)   Converting single reads
  0:00:01.369     9M / 36M   INFO    General                 (binary_converter.cpp      : 111)   102 reads written
  0:00:01.369     1M / 36M   INFO    General                 (read_converter.cpp        :  92)   Converting merged reads
  0:00:01.369     9M / 36M   INFO    General                 (binary_converter.cpp      : 111)   0 reads written
  0:00:01.369     1M / 36M   INFO    StageManager            (stage.cpp                 : 185)   STAGE == de Bruijn graph construction (id: construction)
  0:00:01.369     1M / 36M   INFO    General                 (construction.cpp          : 153)   Max read length 101
  0:00:01.370     1M / 36M   INFO    General                 (construction.cpp          : 159)   Average read length 100.564
  0:00:01.370     1M / 36M   INFO    General                 (stage.cpp                 : 117)   PROCEDURE == k+1-mer counting (id: construction:kpomer_co
```

Then K33

```
 0:02:49.212     3M / 164M  INFO    Simplification          (parallel_processing.hpp   : 167)   Running Tip clipper
 0:02:49.212     3M / 164M  INFO    Simplification          (parallel_processing.hpp   : 170)   Tip clipper triggered 0 times
 0:02:49.212     3M / 164M  INFO    Simplification          (parallel_processing.hpp   : 167)   Running Bulge remover
 0:02:49.212     3M / 164M  INFO    Simplification          (parallel_processing.hpp   : 170)   Bulge remover triggered 0 times
 0:02:49.212     3M / 164M  INFO     General                (simplification.cpp        : 327)   Disrupting self-conjugate edges
 0:02:49.212     3M / 164M  INFO    Simplification          (parallel_processing.hpp   : 167)   Running Removing isolated edges
 0:02:49.212     3M / 164M  INFO    Simplification          (parallel_processing.hpp   : 170)   Removing isolated edges triggered 86 times
 0:02:49.212     3M / 164M  INFO     General                (simplification.cpp        : 337)   Removing all the edges having coverage 7.91177 and less
 0:02:49.212     3M / 164M  INFO     General                (simplification.cpp        : 495)   After simplification:
 0:02:49.212     3M / 164M  INFO     General                (simplification.cpp        : 496)     Average coverage = 325.093
 0:02:49.212     3M / 164M  INFO     General                (simplification.cpp        : 497)     Total length = 201062
 0:02:49.212     3M / 164M  INFO     General                (simplification.cpp        : 498)     Median edge length: 103334
 0:02:49.212     3M / 164M  INFO     General                (simplification.cpp        : 499)     Edges: 165
 0:02:49.212     3M / 164M  INFO     General                (simplification.cpp        : 500)     Vertices: 166
 0:02:49.212     3M / 164M  INFO    StageManager            (stage.cpp                 : 185)   STAGE == Contig Output (id: contig_output)
 0:02:49.212     3M / 164M  INFO     General                (read_converter.cpp        : 107)   Outputting contigs to /Users/eparker/Dropbox (Scripps Res
earch)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/K21/simplified_contigs
 0:02:49.213     6M / 164M  INFO     General                (binary_converter.cpp      : 111)   84 reads written
 0:02:49.213     3M / 164M  INFO     General                (pipeline.cpp              : 287)   SPAdes finished
 0:02:49.213     1M / 164M  INFO     General                (main.cpp                  : 136)   Assembling time: 0 hours 2 minutes 49 seconds

==== K21 finished.


==== K33 started.


= Running: /Applications/spades-spades_3.15.4/assembler/bin/spades-core /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Comma
d_line/Session_2/S1_denovo/K33/configs/config.info

 0:00:00.000     1M / 8M    INFO     General                (main.cpp                  :  99)   Loaded config from /Users/eparker/Dropbox (Scripps Resear
h)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/K33/configs/config.info
 0:00:00.000     1M / 8M    WARN     General                (memory_limit.cpp          :  52)   Failed to limit memory to 250 Gb, setrlimit(2) call faile
, errno = 22 (Invalid argument). Watch your memory consumption!
 0:00:00.000     1M / 8M    INFO     General                (main.cpp                  : 107)   Starting SPAdes, built from N/A, git revision N/A
 0:00:00.000     1M / 8M    INFO     General                (main.cpp                  : 108)   Maximum k-mer length: 128
 0:00:00.000     1M / 8M    INFO     General                (main.cpp                  : 109)   Assembling dataset (/Users/eparker/Dropbox (Scripps Resea
ch)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/dataset.info) with K=33
 0:00:00.000     1M / 8M    INFO     General                (main.cpp                  : 110)   Maximum # of threads to use (adjusted due to OMP capabili
ies): 1
 0:00:00.000     1M / 8M    INFO     General                (pipeline.cpp              : 212)   SPAdes started
 0:00:00.000     1M / 8M    INFO     General                (pipeline.cpp              : 225)   Starting from stage: read_conversion
 0:00:00.000     1M / 8M    INFO     General                (pipeline.cpp              : 231)   Two-step repeat resolution disabled
 0:00:00.000     1M / 8M    INFO    GraphCore               (graph_core.hpp            : 672)   Graph created, vertex min_id: 3, edge min_id: 3
 0:00:00.000     1M / 8M    INFO    GraphCore               (graph_core.hpp            : 673)   Vertex size: 48, edge size: 40
```

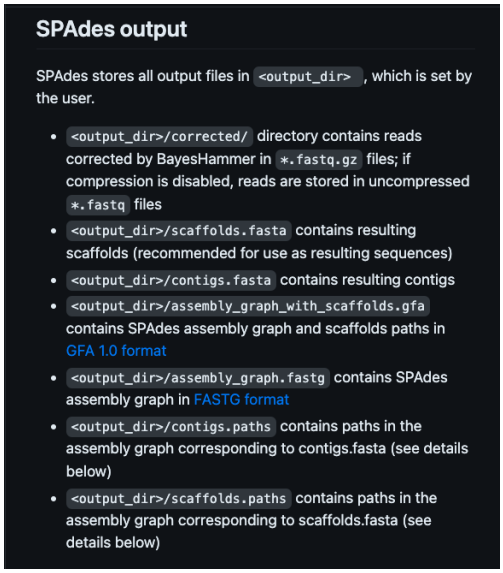
Then K55

```
  0:02:31.909     2M / 163M  INFO     General                (simplification.cpp        : 498)     Median edge length: 126703
  0:02:31.909     2M / 163M  INFO     General                (simplification.cpp        : 499)     Edges: 122
  0:02:31.909     2M / 163M  INFO     General                (simplification.cpp        : 500)     Vertices: 142
  0:02:31.909     2M / 163M  INFO    StageManager            (stage.cpp                 : 185)   STAGE == Contig Output (id: contig_output)
  0:02:31.909     2M / 163M  INFO     General                (read_converter.cpp        : 107)   Outputting contigs to /Users/eparker/Dropbox (Scripps Res
earch)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/K33/simplified_contigs
  0:02:31.909     5M / 163M  INFO     General                (binary_converter.cpp      : 111)   62 reads written
  0:02:31.909     2M / 163M  INFO     General                (pipeline.cpp              : 287)   SPAdes finished
  0:02:31.909     1M / 163M  INFO     General                (main.cpp                  : 136)   Assembling time: 0 hours 2 minutes 31 seconds

===== K33 finished.


===== K55 started.


== Running: /Applications/spades-spades_3.15.4/assembler/bin/spades-core /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Comma
nd_line/Session_2/S1_denovo/K55/configs/config.info

  0:00:00.000     1M / 8M    INFO     General                (main.cpp                  :  99)   Loaded config from /Users/eparker/Dropbox (Scripps Resear
ch)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/K55/configs/config.info
  0:00:00.000     1M / 8M    WARN     General                (memory_limit.cpp          :  52)   Failed to limit memory to 250 Gb, setrlimit(2) call faile
d, errno = 22 (Invalid argument). Watch your memory consumption!
  0:00:00.000     1M / 8M    INFO     General                (main.cpp                  : 107)   Starting SPAdes, built from N/A, git revision N/A
  0:00:00.000     1M / 8M    INFO     General                (main.cpp                  : 108)   Maximum k-mer length: 128
  0:00:00.000     1M / 8M    INFO     General                (main.cpp                  : 109)   Assembling dataset (/Users/eparker/Dropbox (Scripps Resea
rch)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/dataset.info) with K=55
  0:00:00.000     1M / 8M    INFO     General                (main.cpp                  : 110)   Maximum # of threads to use (adjusted due to OMP capabili
ties): 1
  0:00:00.000     1M / 8M    INFO     General                (pipeline.cpp              : 212)   SPAdes started
  0:00:00.000     1M / 8M    INFO     General                (pipeline.cpp              : 225)   Starting from stage: read_conversion
  0:00:00.000     1M / 8M    INFO     General                (pipeline.cpp              : 231)   Two-step repeat resolution disabled
  0:00:00.000     1M / 8M    INFO    GraphCore               (graph_core.hpp            : 672)   Graph created, vertex min_id: 3, edge min_id: 3
  0:00:00.000     1M / 8M    INFO    GraphCore               (graph_core.hpp            : 673)   Vertex size: 48, edge size: 40
  0:00:00.000     1M / 8M    INFO     General                (edge_index.hpp            : 115)   Size of edge index entries: 12/8
  0:00:00.000     1M / 8M    INFO     General                (pipeline.cpp              : 242)   Will need read mapping, kmer mapper will be attached
  0:00:00.000     1M / 9M    INFO    StageManager            (stage.cpp                 : 185)   STAGE == Binary Read Conversion (id: read_conversion)
  0:00:00.000     1M / 9M    INFO     General                (read_converter.cpp        :  53)   Binary reads detected
  0:00:00.000     1M / 9M    INFO    StageManager            (stage.cpp                 : 185)   STAGE == de Bruijn graph construction (id: construction)
  0:00:00.000     1M / 9M    INFO     General                (construction.cpp          : 118)   Contigs from previous K will be used: /Users/eparker/Drop
box (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/K33/simplified_contigs
  0:00:00.000     1M / 9M    INFO     General                (construction.cpp          : 153)   Max read length 101
  0:00:00.000     1M / 9M    INFO     General                (construction.cpp          : 159)   Average read length 100.564
  0:00:00.000     1M / 9M    INFO     General                (stage.cpp                 : 117)   PROCEDURE == k+1-mer counting (id: construction:kpomer_co
```

When it’s finished with K55, it’ll generate the output folder, which we named S1_denovo:

```
===== Terminate finished.

 * Corrected reads are in "/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/corrected/"
 * Assembled contigs are in "/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/contigs.fasta"
 * Assembled scaffolds are in "/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/scaffolds.fast
a"
 * Paths in the assembly graph corresponding to the contigs are in "/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_li
ne/Session_2/S1_denovo/contigs.paths"
 * Paths in the assembly graph corresponding to the scaffolds are in "/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_
line/Session_2/S1_denovo/scaffolds.paths"
 * Assembly graph is in "/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/assembly_graph.fastg
"
 * Assembly graph in GFA format is in "/Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/assemb
ly_graph_with_scaffolds.gfa"

======= SPAdes pipeline finished WITH WARNINGS!

=== Error correction and assembling warnings:
 * 0:00:00.000     1M / 6M    WARN    General                 (memory_limit.cpp          :  52)   Failed to limit memory to 250 Gb, setrlimit(2) call fail
ed, errno = 22 (Invalid argument). Watch your memory consumption!
 * 0:03:18.200     6M / 174M  WARN    General                 (launcher.cpp              : 178)   Your data seems to have high uniform coverage depth. It
is strongly recommended to use --isolate option.
======= Warnings saved to /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/warnings.log

SPAdes log can be found here: /Users/eparker/Dropbox (Scripps Research)/Teaching/Nigeria/2023/Mpox_round2/Command_line/Session_2/S1_denovo/spades.log

Thank you for using SPAdes!
Wed Aug 30 14:10:11 PDT 2023
(base) MacBook-Pro:Session_2 eparker$ ls
GRCh38_noalt_as.zip             S1_human_depleted_R2.fastq.gz   fastqs                          reference
S1_denovo                       coverage                        intermediates
S1_human_depleted_R1.fastq.gz   fastqc_report                   output
```

Here's what we expect as output in the folder S1_denovo:



SPAdes output

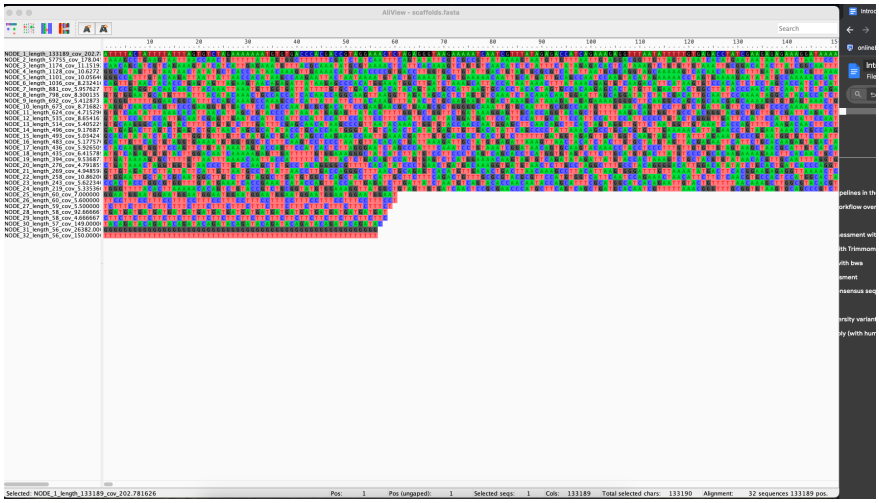
SPAdes stores all output files in `<output_dir>`, which is set by the user.

- `<output_dir>/corrected/` directory contains reads corrected by BayesHammer in `*.fastq.gz` files; if compression is disabled, reads are stored in uncompressed `*.fastq` files
- `<output_dir>/scaffolds.fasta` contains resulting scaffolds (recommended for use as resulting sequences)
- `<output_dir>/contigs.fasta` contains resulting contigs
- `<output_dir>/assembly_graph_with_scaffolds.gfa` contains SPAdes assembly graph and scaffolds paths in GFA 1.0 format
- `<output_dir>/assembly_graph.fastg` contains SPAdes assembly graph in FASTG format
- `<output_dir>/contigs.paths` contains paths in the assembly graph corresponding to contigs.fasta (see details below)
- `<output_dir>/scaffolds.paths` contains paths in the assembly graph corresponding to scaffolds.fasta (see details below)

We should work with the scaffolds.fasta file in the directory we created (S1_denovo/) for all of our downstream processes – no need to dig around in the subdirectories for now! There is also a file named contigs.fasta. This is the assembled contiguous sequences, but *spades* also tries to join contigs together based on read pairs and the assembly graph to form longer scaffolds – which is why we'll use the scaffolds, not the contigs for now!

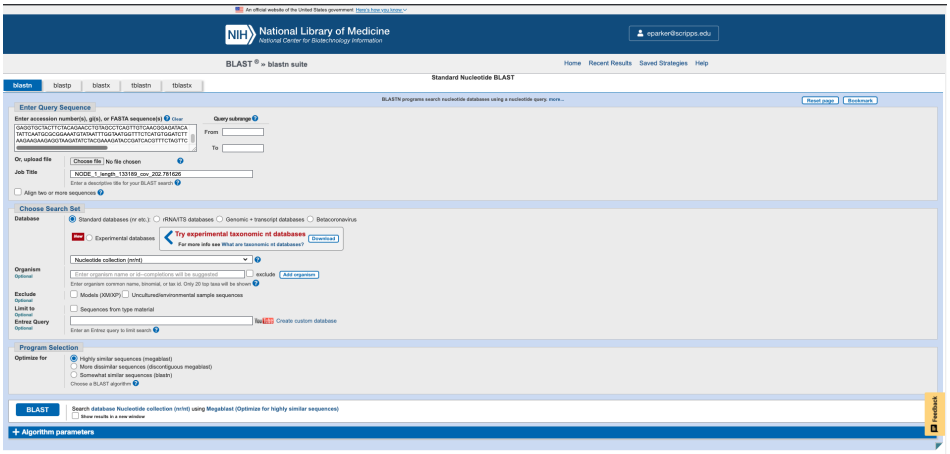
Here is a good explainer on contig vs scaffold: https://www.pacb.com/blog/genomes-vs-gennnnes-difference-contigs-scaffolds-genome-assemblies/

So, if we open the scaffolds.fasta file, we can see our scaffolded contigs or scaffolds! Let's open the scaffolds.fasta in Aliview:

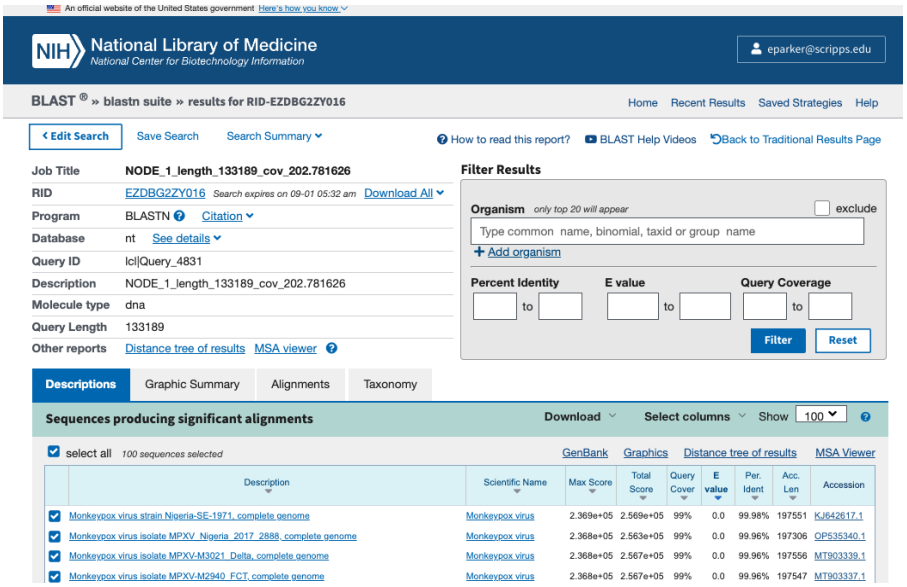


We have a sequence up top named NODE_1_length_133189_cov_202.781626. This is a very long sequence at 133 189 nt, so it's probably Mpox! However, it is clearly partial, as we expect ~197 000 nt for Mpox. It also tells us the coverage in the name (202.781626) – which is decent!

We can confirm the identity of this sequence with BlastN – let's copy and paste it into BlastN. You can literally click on the sequence in Aliview and use the shortcut for copy (control + C or command + C) and then paste (control + V or command + V) that into the BlastN query box.

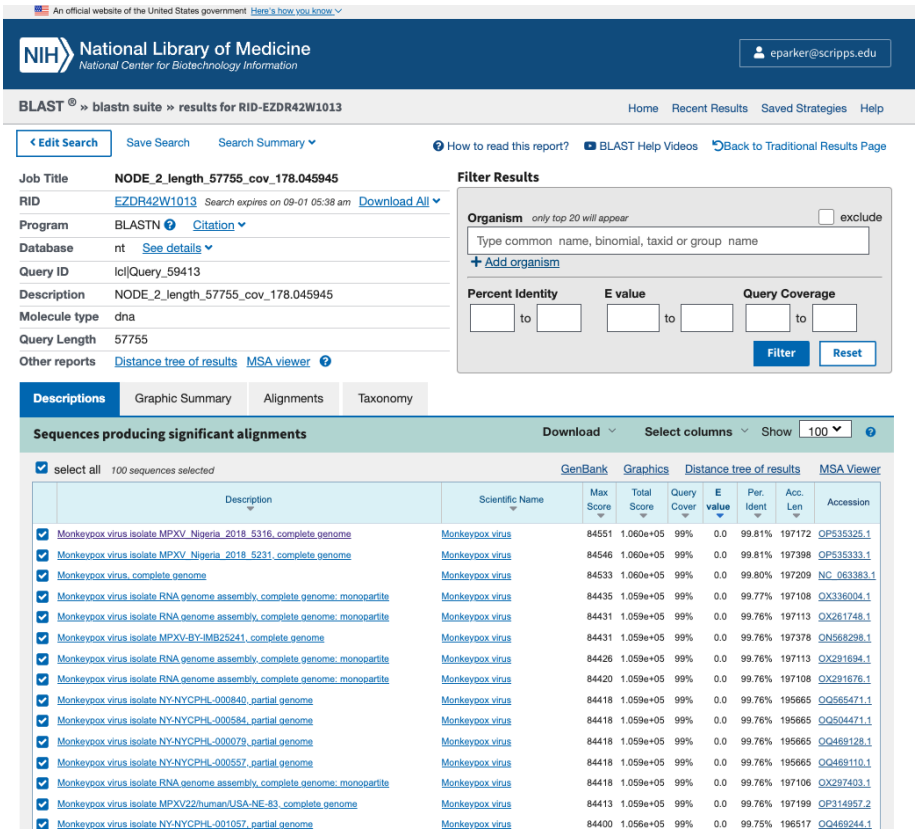


If we run Blast, we can see:



It is Mpox! And we see it's highly similar to sequences circulating in Nigeria from 1971 to more recent – which means it is most likely Clade 2B (as we know from our previous analyses…).

We also had a second long-ish scaffold NODE_2_length_57755_cov_178.045945 (with decent coverage at ~178!).



It too is Mpox! Remember, this is two scaffolded contigs from the same sample – but *spades* did not produce a full assembled contig of around 197 000 here, so it could clearly not figure out how to join these two contigs. This is where reference-based scaffolding comes in, like you performed in Terra. You should try and compare these two scaffolds and see what regions they overlap…