This document contains instructions for the data analysis session for the workshop.

You should have installed the IGV viewer before the workshop. If this was not done, please download **Tablet** (<https://ics.hutton.ac.uk/tablet/download-tablet/> ) right now and make sure it is installed.

In this guide commands that are to be typed and entered in the command prompt are highlighted in **bold**. Messages that may be seen in the terminal window are in Courier New.

### Starting the tutorial in Windows

1. Open Bitvise SSH Client listed in the programmes/apps available
2. Use the host name provided to you at the workshop (trainxx.us.nanopore.me)
3. Use the key as provided (pc\_key\_jackson.ppk)

### Starting the tutorial on a Mac

1. Use the ssh command below along with your host name and the provided key.

**ssh -o ServerAliveInterval=10 -o TCPKeepAlive=no -i /Users/username/Downloads/mac\_key\_jackson.txt ubuntu@trainxx.us.nanopore.me**

### Starting tmux

1. Once connected, initiate a tmux session before running any commands by entering

**ubuntu@ip-10-152-12-61:~$ tmux**

This will ensure any active jobs will keep running even if your connection to the remote machine is somehow interrupted. If this should happen, log back in using ssh (press up arrow to bring back the last command in the terminal) or with Bitwise and reattach to the tmux session using:

**ubuntu@ip-10-152-12-61:~$ tmux attach**

### Basecalling with Albacore

Now we will use Albacore, our standard basecaller package, to basecall the raw data in the my\_data directory.

1. We will first make a new directory for albacore, then download and install it:

**ubuntu@ip-10-152-11-122:~$ cd**

**ubuntu@ip-10-152-11-122:~$ mkdir albacore**

**ubuntu@ip-10-152-11-122:~$ cd albacore**

**ubuntu@ip-10-153-11 122:~/albacore$ wget** **https://mirror.oxfordnanoportal.com/software/analysis/ont\_albacore-2.2.7-cp36-cp36m-manylinux1\_x86\_64.whl**

You can also list directory contents with the “ls” command:

**ubuntu@ip-10-153-11 122:~/albacore$ ls**

ont\_albacore-2.1.3-cp36-cp36m-manylinux1\_x86\_64.whl

1. You should see an installer file ending in .whl as shown above. To run the installer we must first install Python 3, the programing language used by Albacore. Many standard Linux installations such as the one you are using on this Amazon machine come with Python 2 as standard. However we have preloaded **miniconda**, a Python distribution that will let us create custom python environments as needed, including Python 3 environments. To do so use the following command:

**ubuntu@ip-10-153-11-241:~/albacore$ conda create -n albacore python=3**

Fetching package metadata ...........

Solving package specifications: .

Package plan for installation in environment /home/ubuntu/miniconda/envs/albacore:

The following NEW packages will be INSTALLED:

ca-certificates: 2017.08.26-h1d4fec5\_0

certifi: 2017.7.27.1-py36h8b7b77e\_0

libedit: 3.1-heed3624\_0

libffi: 3.2.1-hd88cf55\_4

libgcc-ng: 7.2.0-h7cc24e2\_2

libstdcxx-ng: 7.2.0-h7a57d05\_2

ncurses: 6.0-h9df7e31\_2

openssl: 1.0.2m-h26d622b\_1

pip: 9.0.1-py36h6c6f9ce\_4

python: 3.6.3-h1284df2\_4

readline: 7.0-ha6073c6\_4

setuptools: 36.5.0-py36he42e2e1\_0

sqlite: 3.20.1-hb898158\_2

tk: 8.6.7-hc745277\_3

wheel: 0.29.0-py36he7f4e38\_1

xz: 5.2.3-h55aa19d\_2

zlib: 1.2.11-ha838bed\_2

Proceed ([y]/n)?

**Please type a lower case “y” (no quotes) and press Enter to proceed with the environment setup.**

openssl-1.0.2m 100% |##############################################################################################################################################| Time: 0:00:00 46.37 MB/s

#

# To activate this environment, use:

# > source activate albacore

#

# To deactivate an active environment, use:

# > source deactivate

Let’s activate our new environment:

**ubuntu@ip-10-153-11-241:~/albacore$ source activate albacore**

(albacore) ubuntu@ip-10-153-11-241:~$

You can now see that the prompt begins with (albacore) indicating you are operating inside the new environment. Let’s switch to the albacore directory and launch the albacore installer using the pip command:

**(albacore) ubuntu@ip-10-153-11-241:~/albacore$ pip install ont\_albacore-2.1.3-cp36-cp36m-manylinux1\_x86\_64.whl**

You should see a series of install progress messages ending with the one below indicating albacore was successfully installed:

Successfully installed h5py-2.7.1 numpy-1.13.3 ont-albacore-2.1.3 ont-fast5-api-0.4.1 progressbar33-2.4 python-dateutil-2.6.1 six-1.11.0

1. We will next basecall the raw data in the my\_data directory. Albacore contains basecalling pipelines for 1D and 1D2 data. As we are using a 1D dataset, we will use the read\_fast5\_basecaller.py python script which handles 1D basecalling. Let’s first learn more about the various configuration options possible with read\_fast5\_basecaller.py by intiating the script followed by the “-h” flag as shown below:

**(albacore) ubuntu@ip-10-153-11 241:~/albacore$ read\_fast5\_basecaller.py -h**

You should see usage information as shown below followed by a more detailed description of each available option:

usage: read\_fast5\_basecaller.py [-h] [-l] [-v] [-i INPUT] -t WORKER\_THREADS -s

SAVE\_PATH [--resume [X]] [-f FLOWCELL]

[-k KIT] [--barcoding] [-c CONFIG]

[-d DATA\_PATH] [-b] [-r]

[-n FILES\_PER\_BATCH\_FOLDER] [-o OUTPUT\_FORMAT]

[-q READS\_PER\_FASTQ\_BATCH]

[--disable\_filtering] [--disable\_pings]

An important set of parameters required by the basecaller will be the flow cell model and sequencing kit used to generate the data. You can list all of the available flow cell and kit combinations using the -l option flag:

**(albacore) ubuntu@ip-10-153-11-241:~/albacore$ read\_fast5\_basecaller.py -l**

You will see a list like the one below:

Parsing config files in /home/ubuntu/miniconda/envs/albacore/lib/python3.6/site-packages/albacore/data\_versioned.

Available flowcell + kit combinations are:

flowcell kit barcoding config file

FLO-MIN106 SQK-DCS108 r94\_450bps\_linear.cfg

FLO-MIN106 SQK-LSK108 r94\_450bps\_linear.cfg

FLO-MIN106 SQK-LWB001 included r94\_450bps\_linear.cfg

FLO-MIN106 SQK-LWP001 r94\_450bps\_linear.cfg

FLO-MIN106 SQK-PCS108 r94\_450bps\_linear.cfg

FLO-MIN106 SQK-RAB201 included r94\_450bps\_linear.cfg

We know the dataset in /my\_data was generated using a FLO-MIN106 flowcell and a SQK-LSK108 library preparation kit. To basecall the data in the /my\_data directory we will use the following options:

**-i ~/my\_data** -- specifies the path to where the raw data is located

**-s ~/my\_data** -- specifies where the basecalling output will be saved. A new subfolder named “workspace” will be created for the output.

**-f FLO-MIN106** -- specifies the flowcell model

**-k SQK-LSK108** -- specifies the sequencing kit

**-o fast5,fastq** -- specifies we want both fast5 and fastq output

**-r** -- recursive option, if the raw data directory contains multiple subfolders, albacore will automatically search each one for data to basecall

**-t** -- specifies the number of compute threads to use for basecalling. Since we have 16 threads available on this machine, we will use 14.

The command should look like the one below:

**(albacore) ubuntu@ip-10-153-11-241:~/albacore$ read\_fast5\_basecaller.py -i ~/my\_data -s ~/my\_data -f FLO-MIN106 -k SQK-LSK108 -o fast5,fastq -r -t 14**

### Exploring read length & quality score distribution

1. Let’s take a look at the basecalling output by plotting the quality score and read length distribution. To do so we will need to first deactivate the albacore environment:

**(albacore) ubuntu@ip-10-153-11-252:~/albacore$ source deactivate**

First let’s take a look at the albacore output. Albacore will deposit successfully basecalled reads above a Q-score of 7 inside a /workspace/pass subfolder inside the parent output directory (/my\_data in our case). Let’s switch to this directory now and list the contents:

**ubuntu@ip-10-153-11-252:~/albacore$ cd ~/my\_data/workspace/pass/**

**ubuntu@ip-10-153-11-252:~/albacore$ ls**

1. fastq\_runid\_25867780442cd34270bea76488b29fe01ba3be54\_0.fastq

We have two items inside this directory – a subdirectory named “0” which contains all of the basecalled fast5 files and a fastq file that holds all of the aggregated fastq data from these fast5s. We will be using the fastq file to explore the read length and quality score distribution. First let’s rename it to something easier to manage:

**ubuntu@ip-10-153-11-252:~/my\_data/workspace/pass$ mv fastq\_runid\_25867780442cd34270bea76488b29fe01ba3be54\_0.fastq pass.fastq**

The move command (mv) takes the name of the file we want to rename and then its new name, in this case “pass.fastq”

Let’s plot the read and quality score distribution of the data in pass.fastq. To do this, we will use a python script preloaded in the ~/scripts directory

**ubuntu@ip-10-153-11-252:~/my\_data/workspace/pass$ python ~/scripts/plot\_fastq\_v5.py**

1. The script will fail because our python installation lacks a module called **biopython**. Let’s quickly install this module with the following command:

**ubuntu@ip-10-153-11-252:~/my\_data/workspace/pass$ conda install biopython**

1. Press **Enter** when prompted to “Proceed ([y]/n)?”
2. Now let’s try running the script again. You don’t have to type the command again, simply press the up arrow key to cycle through previously entered commands and run the script command again:

**ubuntu@ip-10-153-11-252:~/my\_data/workspace/pass$ python ~/scripts/plot\_fastq\_v5.py**

1. Once the script prompts you for the path to the input, enter the path to the pass.fastq file as below (just the file name is sufficient since we are already in the correct directory).

Enter path to fastq file: **pass.fastq**

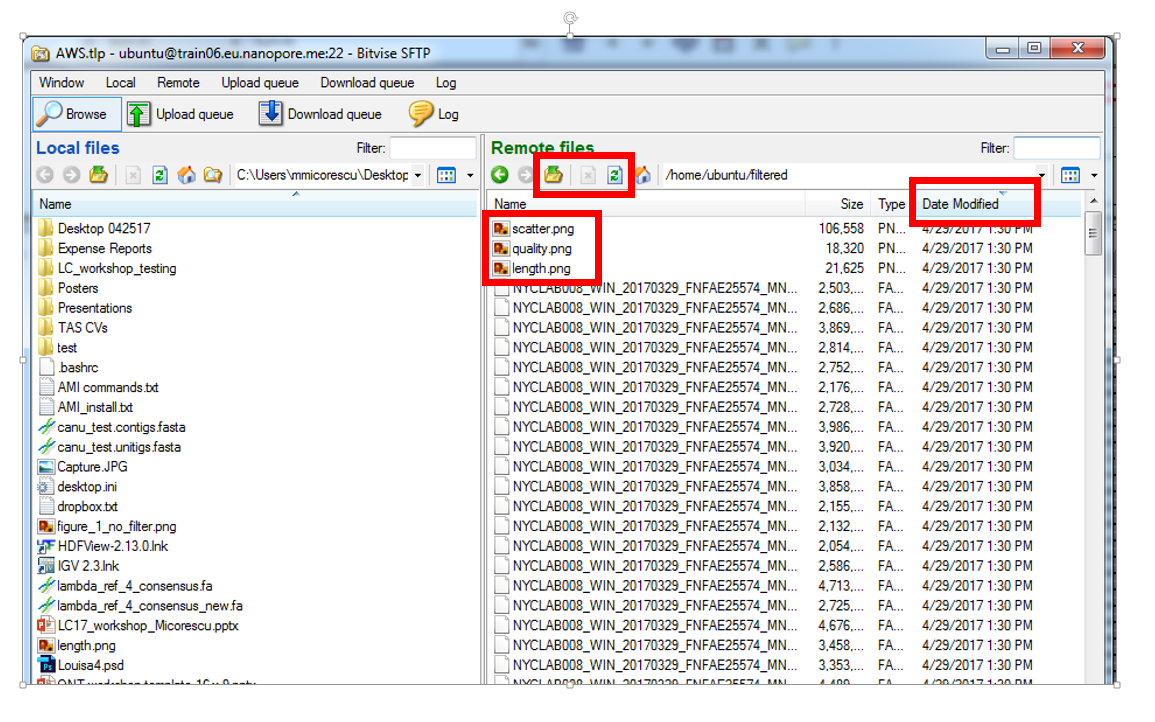
The script will process the reads in the fastq file and output 3 plot files in the current directory:

3317 reads in fastq file

1. Now let’s take a look at the output generated by the script.

### **Windows**

Use the Bitvise file browser to open the **/my\_data/workspace/pass** folder. Sort by date by clicking twice on the "Date Modified" header. Double-click on the length, quality and scatter png files to examine the length, quality score distributions and a scatter plot of quality vs length.



## Mac

Open a second terminal window and connect to the machine using the sftp command below. To input the path to key file you can simply drag the file onto to the terminal window. This path may be different than below depending on where you saved the key file.

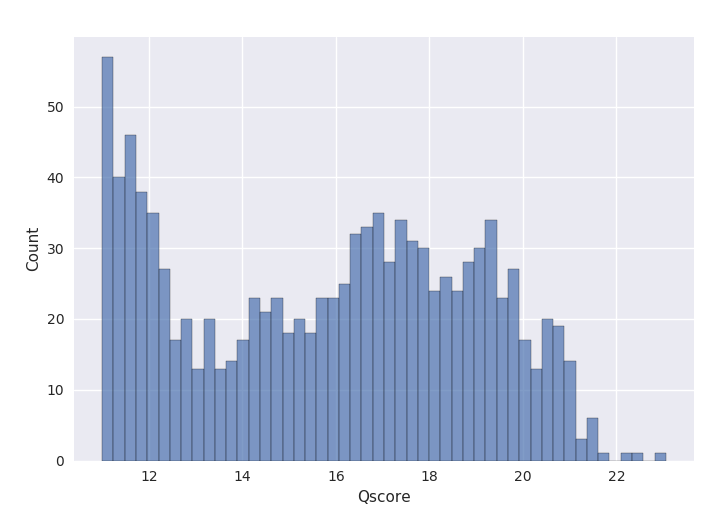
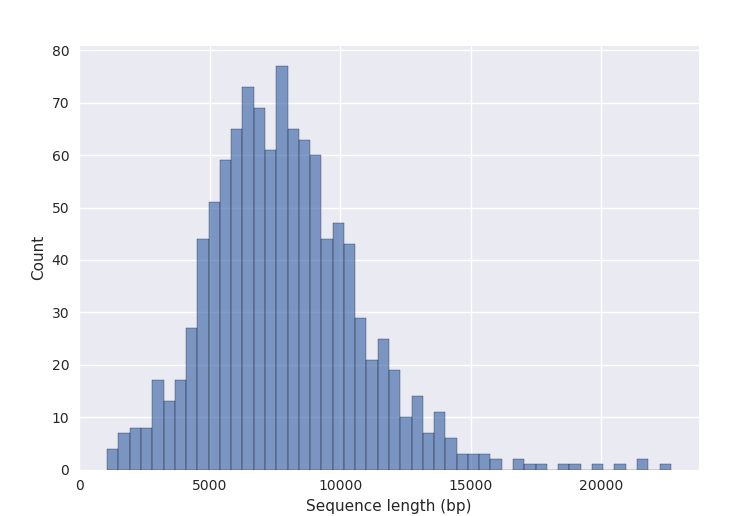
**sftp -o ServerAliveInterval=10 -o TCPKeepAlive=no -i /Users/username/Downloads/demo\_key.txt ubuntu@trainxx.us.nanopore.me**

Once connected download the script output files with the get command:

**sftp> get my\_data/workspace/pass/\*.png**

The png files will be downloaded to your home directory.

1. Double-click on the length, quality and scatter png files to examine the length, quality score distributions and a scatter plot of quality vs length.



1. Now let’s filter the data by both quality and read-length. First we will make a new folder called “filtered” to store the filtered dataset:

**ubuntu@ip-10-152-11-89:~$ mkdir ~/filtered**

1. We will now use another python script located in ~/scripts called filter\_fastq\_v2.py to retain only reads above a Q-score of 8 and longer than 3000 bp. Once the script prompts you for the path to the input and output folders, enter the required information as below:

**ubuntu@ip-10-153-11-252:~/my\_data/workspace/pass$ python ~/scripts/filter\_fastq\_v2.py**

path to file: **pass.fastq**

enter minimum length: **3000**

enter maximum length: **50000**

Enter quality score: **8**

2762 filtered reads saved

1. The script will filter the data and save the output to a new file named filtered.fq inside the current directory. Let’s move the filtered fastq into the **~/filtered** directory:

**ubuntu@ip-10-153-11-252:~/my\_data/workspace/pass$ mv filtered.fq ~/filtered/filtered.fq**

Let’s plot the filtered data to confirm the process was successful. We will use the same plot script as before, but first switch to the **~/filtered** directory:

**ubuntu@ip-10-153-11-252:~/my\_data/workspace/pass$ cd ~/filtered/**

and run the plot script again on the filtered.fastq file:

**ubuntu@ip-10-153-11-252:~/filtered$ python ~/scripts/plot\_fastq\_v5.py**

Enter path to fastq file **filtered.fq**

2762 reads in fastq file

## Windows

Use the Bitvise file browser window to take a look at the filtered folder. You will have to press the “Up” button, then the refresh button before the new filtered folder will show up.

## Mac

Use your second terminal window and download the files with the get command:

**sftp> get filtered/\*.png**

Take a look at the plots and confirm we have only retained data above our filtering thresholds.

### Mapping the filtered data to a reference

We are next going to map the filtered data to the lambda reference. We will use the **Pomoxis** pipeline developed by Oxford Nanopore. It is available via the Oxford Nanopore Github space.

Pomoxis chains together some very effective tools developed by the nanopore community such as **minimap2**, **miniasm**, **racon** and **porechop**.

Pomoxis contains a **mini\_align** module we will use for mapping and a mini\_assemble module we will use for *de novo* assembly later.

Pomoxis has been preloaded on this machine and runs in its own virtual environment.

1. Let’s start by activating the Pomoxis environment:

**ubuntu@ip-10-153-11-241:~/filtered$ cd ~/git/pomoxis/**

Be sure to include the space between to the two periods in the command:

**ubuntu@ip-10-153-11-241:~/git/pomoxis$ . ./venv/bin/activate**

(venv) ubuntu@ip-10-153-11-241:~/git/pomoxis$

To find out more about Pomoxis mini\_align options, use the **-h** flag:

**(venv) ubuntu@ip-10-153-11-241:~/git/pomoxis$ mini\_align -h**

mini\_align [-h] -r <reference> -i <fastq>

Align fastq/a formatted reads to a genome using minimap2.

-h show this help text.

-r reference, should be a fasta file. If correspondng bwa indices

do not exist they will be created. (required).

-i fastq/a input reads (required).

-a aggresively extend gaps (sets -A1 -B2 -O2 -E1 for minimap2).

-P filter to only primary alignments (i.e. run samtools view -F 2308)

-n sort bam by read name.

-c chunk size. Input reads/contigs will be broken into chunks

prior to alignment.

-t alignment threads (default: 1).

-p output file prefix (default: reads).

We will switch to the **~/filtered** directory and use the following command to map the filtered data to the lambda reference (available in **~/refs/lambda\_ref.fasta**):

**(venv) ubuntu@ip-10-153-11-241:~/filtered$ mini\_align -i filtered.fq -r ~/refs/lambda\_ref.fasta -P -p lambda -t 14**

Found minimap files.

[M::main::0.003\*1.18] loaded/built the index for 1 target sequence(s)

[M::mm\_mapopt\_update::0.004\*1.16] mid\_occ = 2

[M::mm\_idx\_stat] kmer size: 15; skip: 10; is\_HPC: 0; #seq: 1

[M::mm\_idx\_stat::0.004\*1.14] distinct minimizers: 9040 (99.99% are singletons); average occurrences: 1.000; average spacing: 5.365

[M::worker\_pipeline::1.357\*10.34] mapped 2762 sequences

[M::main] Version: 2.3-r531

[M::main] CMD: minimap2 -L -x map-ont -t 14 -a /home/ubuntu/refs/lambda\_ref.fasta.mmi filtered.fq

[M::main] Real time: 1.358 sec; CPU: 14.029 sec

1. Pomoxis used minimap2 to very quickly create an alignment file named lambda.bam as well as an alignment index name lambda.bam.bai. Let’s download both onto your local machine:

## Windows

Use the Bitvise file browser to find the lambda.bam and lambda.bam.bai as well as the lambda reference file (from /home/Ubuntu/refs). Drag these from the browser window onto a location on your local machine (desktop works fine).

## Mac

Use the following commands:

**sftp> get filtered/lambda.bam**

**sftp> get filtered/lambda.bam.bai**

**sftp> get refs/lambda\_ref.fasta**

### Visualizing the alignment using Tablet

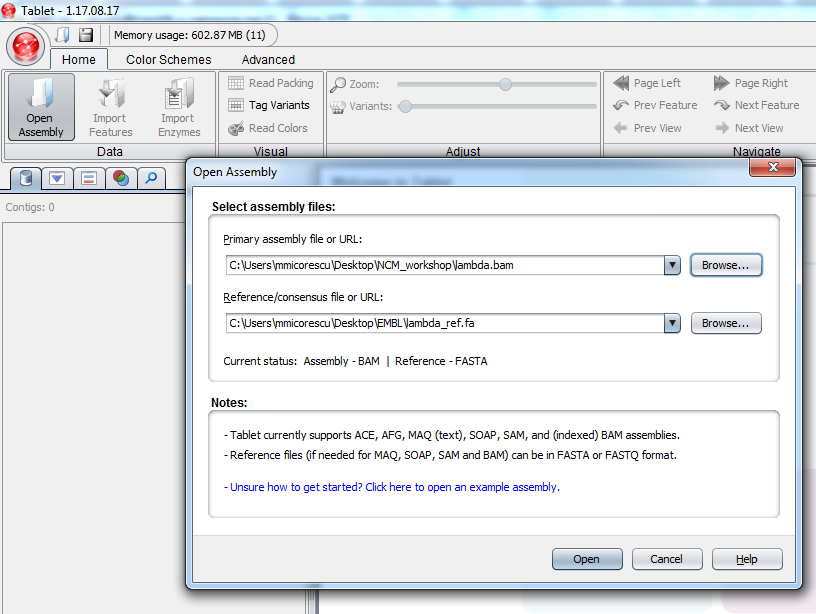
If you have already downloaded and installed the IGV viewer before the workshop, please see the next section for instructions on using IGV to visualize the alignment. If you do not have IGV installed, please download and use Tabled as described below:

1. Navigate to a browser window

**https://ics.hutton.ac.uk/tablet/download-tablet/**

Please download the version of Tablet appropriate for your machine and launch the application.

1. Once Tablet starts use the “Open Assembly” button to browse and select the lambda.bam and lambda\_ref.fasta file using the assembly and reference browse buttons repectevely. Once the alignment loads you should be able to zoom in using the controls at the top right and to pan right and left by clicking on the alignment and dragging left or right.



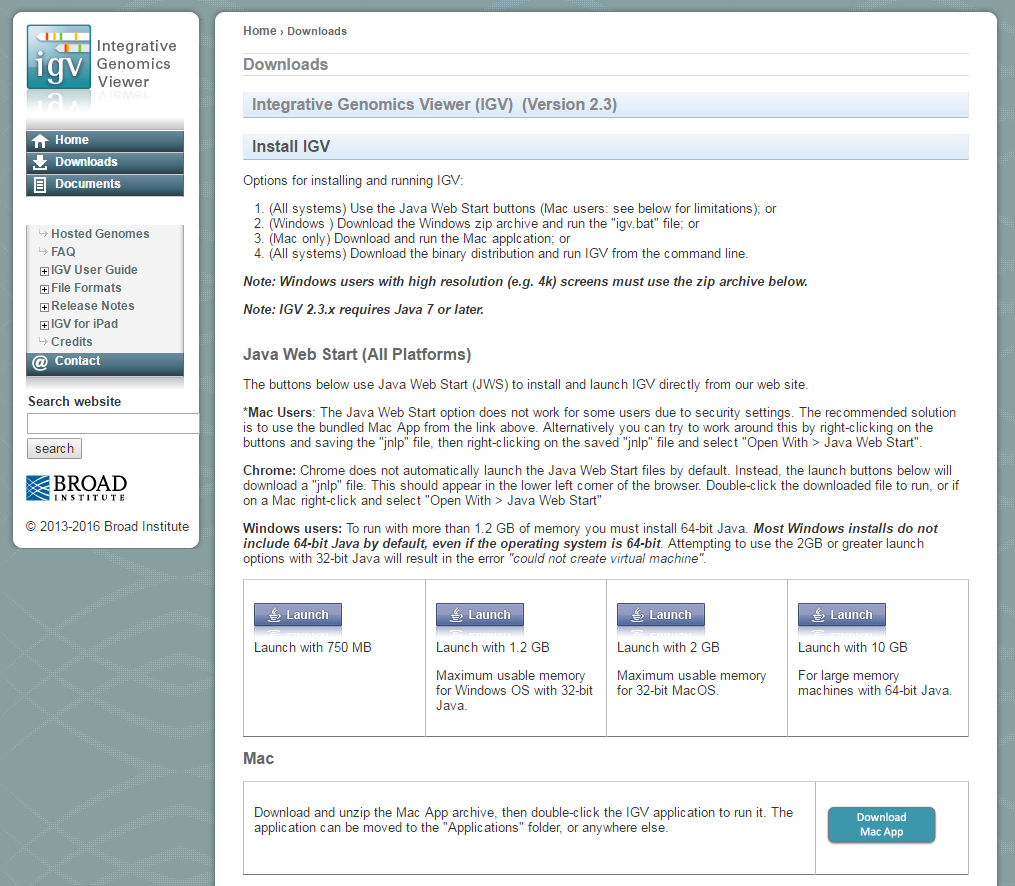
Although we only used ~2700 reads you should see even coverage across the entire lambda genome.

### Visualizing the alignment using IGV

1. Navigate to a browser window

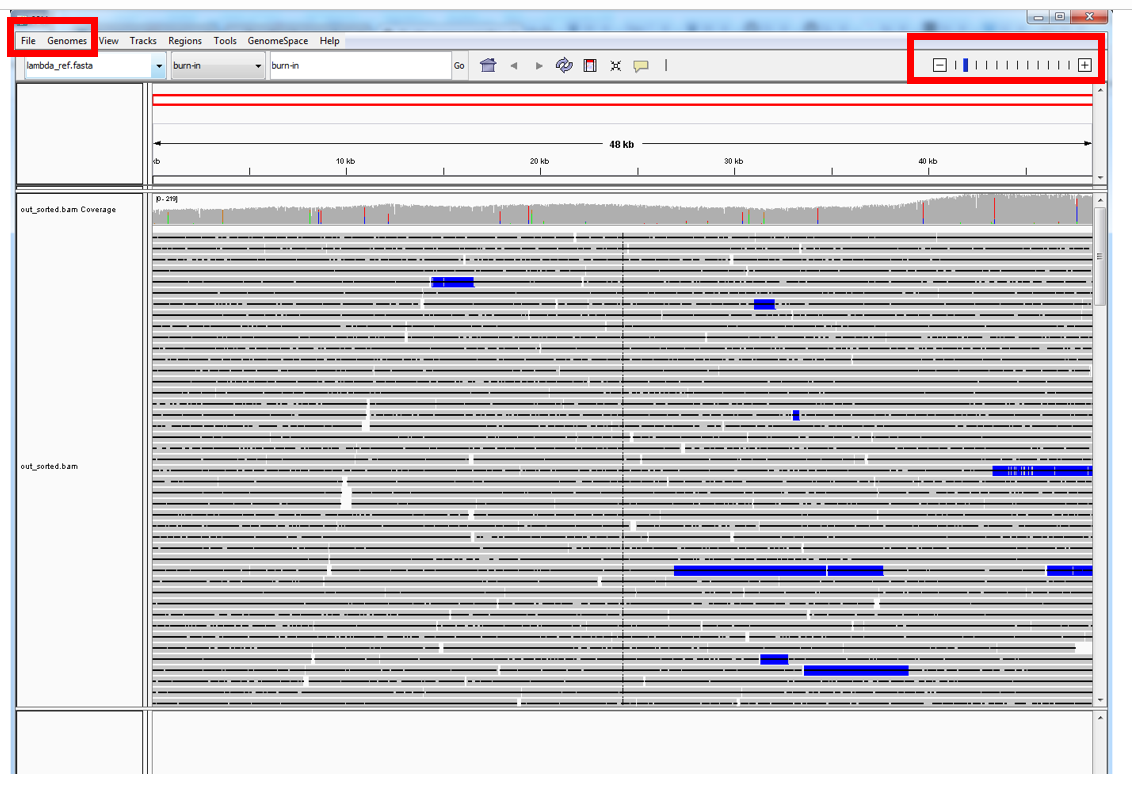
**https://software.broadinstitute.org/software/igv/download**

Please launch the 1.2 GB version of the viewer. Mac users will have to download the Mac App. Java will be required by IGV, so if you don’t have it installed, see the url below:



**https://java.com/en/download**/

1. Once the viewer starts, load the lambda reference by going to Genomes>Load genomes form file via the menu bar. Then load the lambda.bam file using File>Load from file. Once the alignment loads you should be able to zoom in using the controls at the top right and to pan right and left by clicking on the alignment and dragging left or right.



Although we only used ~2700 reads you should see even coverage across the entire lambda genome.

### *de novo* assembly

Now let’s see if this dataset is enough for a de novo assembly of the lambda genome. We will use the Pomoxis **mini\_assemble** package for this.

1. First let’s take a look at the mini\_assemble options:

**(venv) ubuntu@ip-10-153-11-241:~/git/pomoxis$ mini\_assemble -h**

mini\_assemble [-h] -i <fastq>

Assemble fastq/fasta formatted reads and perform POA consensus.

-h show this help text.

-i fastx input reads (required).

-q use qualities as is (default: false).

-r reference fasta for reference-guided consensus (instead of de novo assembly)

-o output folder (default: assm).

-p output file prefix (default: reads).

-t number of minimap and racon threads (default: 1).

-c trim adapters from reads prior to everything else.

-e error correct longest e% of reads prior to assembly.

1. Let’s make sure you are in your home directory. Type “cd” and press enter to get there – this will return you to the home directory from any other directory, in the example below I was in ~/git/pomoxis:

**(venv) ubuntu@ip-10-153-11-241:~/git/pomoxis$ cd**

Now let’s run mini\_assemble as follows:

**(venv) ubuntu@ip-10-153-11-241:~$ mini\_assemble -i ~/filtered/filtered.fq -o denovo -p lambda\_denovo -t 14 -c**

Running mini\_assemble will take a few minutes. The -c option will run porechop first and trim away any remaining adapter sequences. We are creating a new output folder named “denovo” in the home directory for our results using -o and files therein will be labeled using the –p flag with the “lambda\_denovo” tag. Lastly, we are using the –i flag to specify where the reads are located and –t for the number of compute threads.

The assembly should finish with the following message:

Final assembly written to denovo/lambda\_denovo\_final.fa. Have a nice day.

## Windows

Navigate to the **/denovo** folder and download the **lambda\_denovo\_final.fa** file.

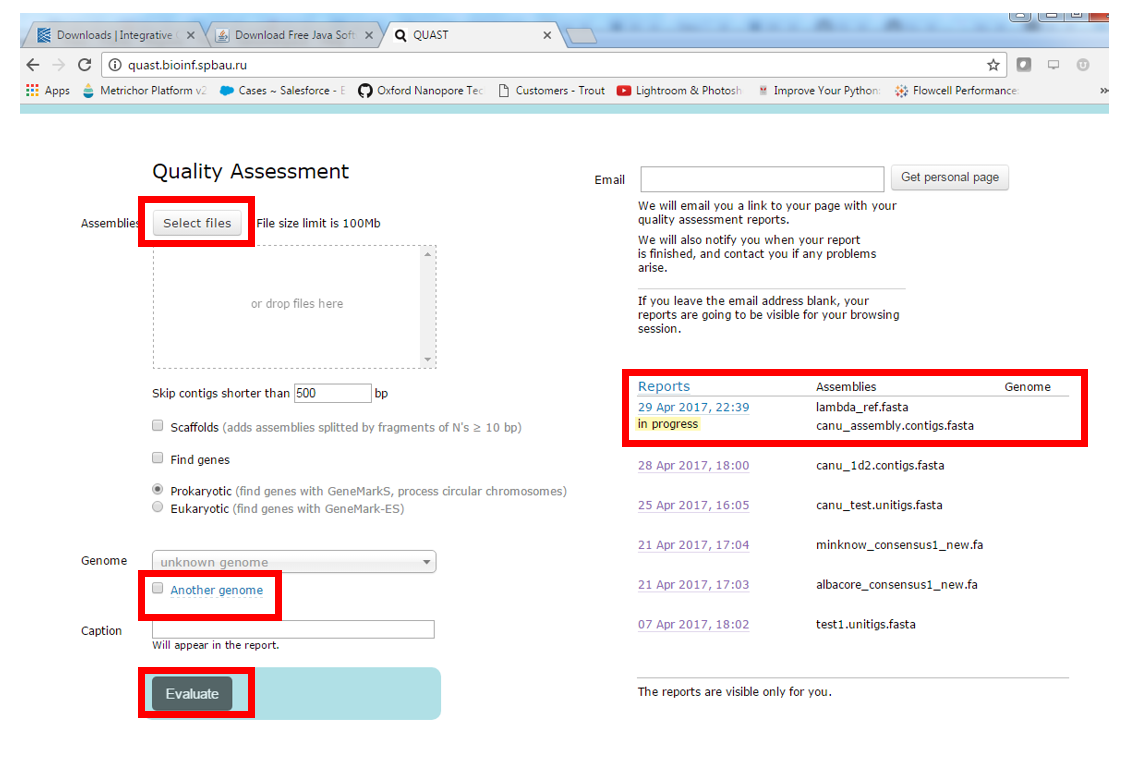
## Mac

Use another get command in your sftp window:

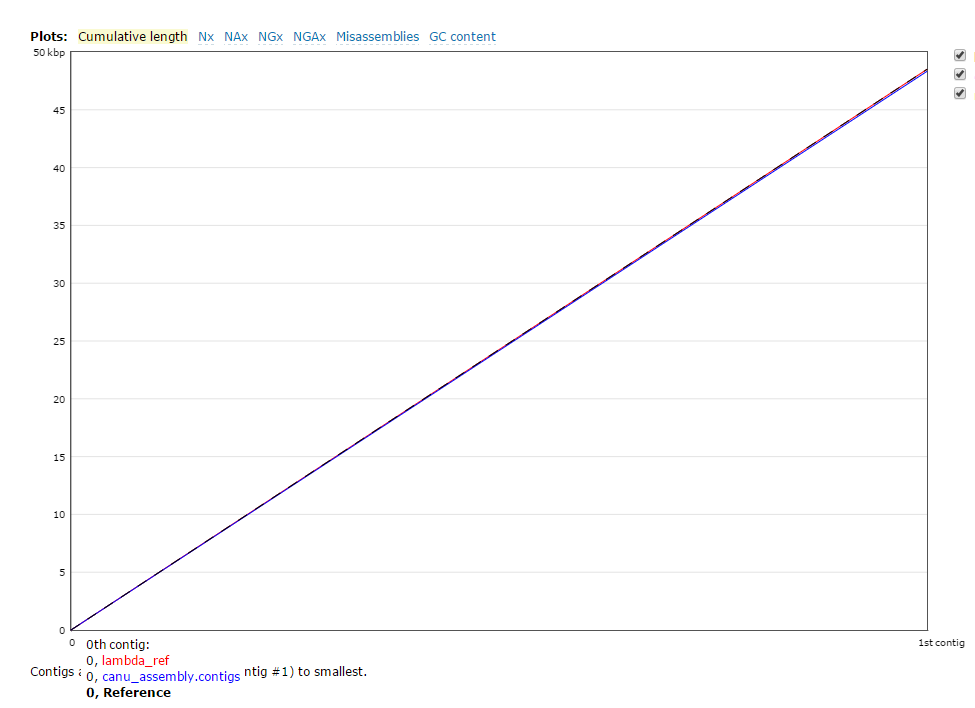
**sftp> get denovo/lambda\_denovo\_final.fa**

### Quality of the assembly

1. From a browser please navigate to [**http://quast.bioinf.spbau.ru/**](http://quast.bioinf.spbau.ru/)
2. Use the “select files” button to load **lambda\_denovo\_final.fa**, the “Another genome” link to load the lambda reference (lambda.fasta) and then press “Evaluate”. In a few minutes the output can be loaded via the “Reports” tab on the right.



You should now see a plot like the one below confirming that the assembly generate a full length contig covering the entire reference.



This is the end of the tutorial and all terminals should now be closed.