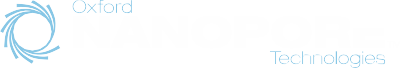
**Data Analysis Gui****de**



Oxford Nanopore Technologies

Oxford Science Park, Oxford OX4 4GA, UK [support@nanoporetech.com](mailto:support@nanoporetech.com)

[**www.nanoporetech.com**](http://www.nanoporetech.com/)

# INTRODUCTION

Welcome to this Data Analysis Workshop. In these 3 sessions we will cover:

1. The essentials of command line
2. Installation of analysis tools (for reference only)
3. Starting the tutorial
4. Basecalling data using Guppy
5. Exploring the read length and quality score distributions of the basecalled data, then filtering the reads based on these metrics
6. Mapping the filtered reads to a reference
7. Visualizing the alignment using Tablet
8. Performing a *de novo* assembly using Pomoxis (minimap2+miniasm+racon OLC pipeline)
9. Assessing assembly quality with QUAST
10. Error correcting the assembly using Medaka and comparing error corrected assembly quality using mummer (dnadiff)
11. Detecting methylated bases using Nanopolish

This document contains detailed step-by-step instructions for the above.

**STEP 1: A BRIEF COMMAND LINE TUTORIAL**

The command line is a text interface, which allows a user to interact with the computer directly by typing in commands from a keyboard, rather than using a mouse to click on buttons and menus. The advantages of using command line over a graphic user interface include the former being faster to use, requiring less memory, and lower CPU power, to name a few.

For those unfamiliar with it, we have designed a few tutorials to help get you up and running with using the command line. Please click on the hyperlinks to access the different sessions:

1. [Introduction to command line](https://www.katacoda.com/amblina/scenarios/tutorial1)
2. [Beginner’s guide to pagers](https://www.katacoda.com/amblina/scenarios/tutorial2)
3. [Familiarising yourself with the unfamiliar](https://www.katacoda.com/amblina/scenarios/tutorial3)
4. [Opening and searching files](https://www.katacoda.com/amblina/scenarios/tutorial4)

**How to follow this tutorial**

In this tutorial, when we want you to enter a command, we’ve included the whole command prompt (in **grey**) and the command (in **bold black** and in **monospace font**). To copy and paste a command into the terminal, just copy the black monospace command, and paste it into your terminal by right-clicking. Then press enter.

Command prompts generally take the following form:

**username@machine\_name:present\_working\_directory$**

For example, if you are logged into machine **ip-10-11-12-14** using the **ubuntu** username and your present working directory is **~** (shortcut for the home directory, **/home/ubuntu**), then your prompt will appear as such:

**ubuntu@ip-10-11-12-14:/home/ubuntu$**

Running the **pwd** command will verify that we are indeed currently in the home directory for the ubuntu username:

**ubuntu@ip-10-11-12-14:/home/ubuntu$ pwd**

/home/ubuntu/

Please note that some commands may be very long and will wrap around the edge of the screen; these are to be treated as *a* *single set of instructions* and may appear as a separate line from the **grey** command prompt.

**STEP 2: INSTALLATION OF SOFTWARE**

**NOTE:** The tools we will be using for this workshop are already pre-installed on these training machines to save on time. As such, none of the following tools need to be installed during the workshop itself, but the installation instructions are provided below for your reference.

**Guppy**

Guppy is a cross-platform bioinformatics toolkit that enables real-time basecalling and several post-processing features that works on Oxford Nanopore Technologies sequencing platforms. Download and installation instructions can be found on the Nanopore Community Downloads page:

<https://community.nanoporetech.com/downloads>

There are packaged installers available for Windows, Ubuntu 16, CentOS 6, and CentOS 7 operating systems. GPU basecalling is enabled on Ubuntu 16, CentOS 6, and CentOS 7, but may require you to manually install your GPU drivers. Installation instructions can be found on the downloads page linked above.

Archives containing precompiled binaries are available for macOS and other Linux distributions as **.zip** or **.tar.gz** files. These do not require installation and simply need to be extracted to run the binaries directly.

**Conda**

Conda is an open-source, cross-platform package manager and environment management system. A wide variety of bioinformatics and scientific computing software has been deployed within Conda and it provides a streamlined way to install both software packages and their required dependencies without the requirement for administrative rights. Many nanopore analysis tools are available through and can be install from conda repositories.

Downloads and installation instructions can be found on the following page: <https://docs.conda.io/en/latest/miniconda.html>

**NanoPlot**

NanoPlot is a third-party plotting tool for long read sequencing data and alignments. It can be installed through conda by running the following command. More details can be found on its GitHub page: <https://github.com/wdecoster/NanoPlot>

**ubuntu@ip-10-11-12-14:/home/ubuntu $ conda install -c bioconda nanoplot**

## Package Plan ##

environment location: /home/ubuntu/miniconda

added / updated specs:

- nanoplot

The following packages will be downloaded:

package | build

---------------------------|-----------------

biopython-1.71 | py36h637b7d7\_0 2.4 MB

libgcc-7.2.0 | h69d50b8\_2 304 KB

samtools-1.7 | 1 1.0 MB bioconda

nanoplotter-0.38.0 | py36\_0 20 KB bioconda

conda-4.5.4 | py36\_0 1.0 MB

htslib-1.7 | 0 1.1 MB bioconda

curl-7.59.0 | h84994c4\_0 140 KB

pauvre-0.1.86 | py36\_0 66 KB bioconda

nanomath-0.21.0 | py36\_0 10 KB bioconda

decorator-4.3.0 | py36\_0 15 KB

plotly-2.5.1 | py36\_0 19.9 MB

jsonschema-2.6.0 | py36h006f8b5\_0 62 KB

jupyter\_core-4.4.0 | py36h7c827e3\_0 61 KB

ipython\_genutils-0.2.0 | py36hb52b0d5\_0 39 KB

bcftools-1.7 | 0 2.7 MB bioconda

pysam-0.14.1 | py36\_htslib1.7\_0 7.9 MB bioconda

traitlets-4.3.2 | py36h674d592\_0 131 KB

nanoget-1.5.0 | py36\_0 17 KB bioconda

nanoplot-1.13.0 | py36\_0 22 KB bioconda

nbformat-4.4.0 | py36h31c9010\_0 137 KB

------------------------------------------------------------

Total: 37.2 MB

The following NEW packages will be INSTALLED:

bcftools: 1.7-0 bioconda

biopython: 1.71-py36h637b7d7\_0

curl: 7.59.0-h84994c4\_0

decorator: 4.3.0-py36\_0

htslib: 1.7-0 bioconda

ipython\_genutils: 0.2.0-py36hb52b0d5\_0

jsonschema: 2.6.0-py36h006f8b5\_0

jupyter\_core: 4.4.0-py36h7c827e3\_0

libgcc: 7.2.0-h69d50b8\_2

nanoget: 1.5.0-py36\_0 bioconda

nanomath: 0.21.0-py36\_0 bioconda

nanoplot: 1.13.0-py36\_0 bioconda

nanoplotter: 0.38.0-py36\_0 bioconda

nbformat: 4.4.0-py36h31c9010\_0

pauvre: 0.1.86-py36\_0 bioconda

plotly: 2.5.1-py36\_0

pysam: 0.14.1-py36\_htslib1.7\_0 bioconda

samtools: 1.7-1 bioconda

traitlets: 4.3.2-py36h674d592\_0

The following packages will be UPDATED:

conda: 4.5.2-py36\_0 --> 4.5.4-py36\_0

Proceed ([y]/n)?

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

Downloading and Extracting Packages

biopython 1.71###################################### | 100%

libgcc7.2.0######################################### | 100%

samtools 1.7####################################### | 100%

nanoplotter 0.38.0#################################### | 100%

conda 4.5.4######################################## | 100%

htslib 1.7######################################### | 100%

curl 7.59.0#########################################| 100%

pauvre 0.1.86####################################### | 100%

nanomath 0.21.0#####################################| 100%

decorator 4.3.0######################################| 100%

plotly 2.5.1#########################################| 100%

jsonschema 2.6.0#####################################| 100%

jupyter\_core 4.4.0####################################| 100%

ipython\_genutils 0.2.0################################# | 100%

bcftools 1.7########################################| 100%

pysam 0.14.1#######################################| 100%

traitlets 4.3.2#######################################| 100%

nanoget 1.5.0#######################################| 100%

nanoplot 1.13.0######################################| 100%

nbformat 4.4.0#######################################| 100%

Preparing transaction: done

Verifying transaction: done

Executing transaction: done

**Filtlong**

Filtlong is a third-party tool for filtering long reads by quality. It can take a set of long reads and produce a smaller, better subset. It uses both read length (longer is better) and read identity (higher is better) when choosing which reads pass the filter. It can be installed through conda by running the following command. More details can be found on its GitHub page: <https://github.com/rrwick/Filtlong>

**ubuntu@ip-10-11-12-14:/home/ubuntu $** **conda install -c bioconda filtlong**

Solving environment: done

## Package Plan ##

environment location: /home/ubuntu/miniconda

added / updated specs:

- filtlong

The following packages will be downloaded:

package | build

---------------------------|-----------------

filtlong-0.2.0 | 1 70 KB bioconda

The following NEW packages will be INSTALLED:

filtlong: 0.2.0-1 bioconda

Proceed ([y]/n)?

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

Downloading and Extracting Packages

filtlong-0.2.0 | 70 KB | ########################################################### | 100%

Preparing transaction: done

Verifying transaction: done

Executing transaction: done

**Tablet**

Tablet is a third-party graphical viewer for next generation sequence assemblies and alignments. It can be downloaded from its home page: <https://ics.hutton.ac.uk/tablet/>

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

**Nanopolish**

Nanopolish is a third-party tool for signal-level analysis of Oxford Nanopore sequencing data. Nanopolish can calculate an improved consensus sequence for a draft genome assembly, detect base modifications, call SNPs and indels with respect to a reference genome and other applications. It can be installed through conda by running the following command. More details can be found on its GitHub page: <https://github.com/jts/nanopolish>

**ubuntu@ip-10-11-12-14:/home/ubuntu$ conda install -c bioconda nanopolish**

**MUMmer**

MUMmer is a third-party system for rapidly aligning entire genomes, whether in complete or draft form. It contains a utility called dnadiff, which is a wrapper for nucmer and analysis utilities that provides detailed information on the differences between two genomes, and also provides a high level report file that quantifies the differences between the two inputs. It can be installed through conda by running the following command.More details can be found on its home page: <http://mummer.sourceforge.net/>

**ubuntu@ip-10-11-12-14:/home/ubuntu $ conda install -c bioconda mummer**

**STEP 3: STARTING THE TUTORIAL**

**Connecting to the host machine on a Windows**

1. Open **Bitvise** **SSH Client** listed in the programs/apps available
2. Enter the IP address provided to you at the workshop
3. Continue by using the key provided this morning (**ont\_pc\_key.ppk/Global1**)

**Connecting to the host machine on a Mac/Linux machine**

1. Use the ssh command below along with the new host name. The same key used during the pre-workshop connectivity test will work (**ont\_mac\_key.txt**). To input the path to the 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.
2. Type (being sure to replace **10.11.12.13** with your machine number and to replace the file path if the key was downloaded to a location other than your Do) and Enter the following:

**ssh -o ServerAliveInterval=10 -o TCPKeepAlive=no -i /Users/$(whoami)/Downloads/ont\_mac\_key ubuntu@10.11.12.13**

**Notes before we start**

Before you begin the tutorial, please follow the instructions below to disable your grammar checking (it can interfere with the commands):

* Click the File tab, and then click Options.
* Click Proofing.
* Click to clear the Check spelling as you type check box.
* Click to clear the Mark grammar errors as you type check box
* Click on ‘View’ in the Taskbar and open the Navigation pane to enable you to switch easily between different sections of this document

\*Remember to turn these back on after the data analysis session

\*\* There are some URLs which may be difficult to type out, if you are struggling with these, it is recommended that you copy and paste the commands from the document opened on your own machine and paste directly into the terminal on the ubuntu machine. In order to do that please copy as normal and paste on the ubuntu machine by right clicking inside the terminal window. It is not recommended to open the whole document within the Virtual Machine as there may be formatting issues.

\*\*\*The command line is case sensitive, so always type commands exactly as given. For more complex commands spaces are important as well.

**Useful information**

* When you're typing file names (and many other things) on the command line, use the **TAB** key to autocomplete the word. This will save you tons of time. It will automatically fill in the remainder of the word once you've typed enough to uniquely identify it.
* Pressing the **UP** key will cycle through the history of recently used commands. Very handy.

**Initiate a tmux session**

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

**ubuntu@ip-10-11-12-14:/home/ubuntu$ tmux**

This will ensure any active jobs will keep running even if your connection to the remote machine is somehow interrupted.

If your connection gets interrupted, 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-10-11-12-14:/home/ubuntu$ tmux attach**

**STEP 4: BASECALLING WITH GUPPY**

Now we will use Guppy, our standard basecaller package, to basecall a Lambda dataset. But first we will need to download some raw data from Dropbox:

1. Let’s make a new directory for our Lambda data and download it:

**ubuntu@ip-10-11-12-14:/home/ubuntu$ mkdir lambda**

**ubuntu@ip-10-11-12-14:/home/ubuntu$ cd lambda**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$**

**wget https://www.dropbox.com/s/eml7z2d82n3k8lq/lambda.tar.gz?dl=0 -O lambda.tar.gz**

1. Let’s extract this tar file:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ tar -xf lambda.tar.gz**

This should unzip and give us 4000 individual fast5 files. To double-check that the extraction completed successfully, output the number of files in a folder use with the following command (please note that the vertical bar after the **ls** command is a pipe character, **|**, and the flag succeeding the **wc** command is a lower-case letter L):

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ ls | wc -l**

4001

And then remove the .tar.gz file so we are left with only the .fast5 files:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ rm lambda.tar.gz**

1. We will next basecall the raw data in the **/home/ubuntu/lambda** directory. Guppy contains basecalling pipelines for 1D and 1D2 data. As we are using a 1D dataset, we will use the guppy\_basecaller package which handles 1D basecalling. Let’s first learn more about the various configuration options possible with guppy\_basecaller by initiating guppy\_basecaller followed by the “-h” flag as shown below:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ guppy\_basecaller -h**

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

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 **--print\_workflows** option flag:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ guppy\_basecaller --print\_workflows**

You will see a list like the one below:

ubuntu@ip-10-153-10-9:/home/ubuntu/guppy$ guppy\_basecaller --print\_workflows

Available flowcell + kit combinations are:

flowcell kit barcoding config\_name

FLO-MIN106 SQK-DCS108 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-DCS109 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-LRK001 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-LSK108 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-LSK109 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-LWP001 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-PCS108 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-PCS109 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-PSK004 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-RAD002 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-RAD003 dna\_r9.4.1\_450bps\_hac

FLO-MIN106 SQK-RAD004 dna\_r9.4.1\_450bps\_hac

We know the lambda dataset was generated using a FLO-MIN106 flowcell and an SQK-LSK108 library preparation kit. To basecall the data in the **/home/ubuntu/lambda** directory we can use the following options:

**-i** specifies the path to where the raw data is located

**-s**  specifies where the basecalling output will be saved

**--flowcell** specifies the flowcell model (FLO-MIN106/FLO-MIN107)

**--kit** specifies the sequencing kit (e.g. SQK-LSK109)

**--fast5\_out** specifies we want both fast5 and fastq output

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

**--num\_callers**

specifies the number of parallel basecallers to use. Note: will create one fastq file per caller.

**--cpu\_threads\_per\_caller**

specifies the number of compute threads to use for each basecaller instance. Since we have 8 threads available on this machine and are using 1 caller, we will use 8 threads per caller.

As the print workflows command from above shows, using this flow cell+kit combination will basecall using the high-accuracy model to obtain the highest raw read accuracy at the expense of speed. For the purposes of this tutorial, we will use the fast model as there is only a ~3% accuracy difference but it will basecall much faster. The command should look like the one below:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$**

**guppy\_basecaller -i /home/ubuntu/lambda -s /home/ubuntu/lambda --config dna\_r9.4.1\_450bps\_fast.cfg -r --num\_callers 1 --cpu\_threads\_per\_caller 8**

ONT Guppy basecalling software version 3.1.5+781ed57

config file: /usr/lib/ont-guppy-cpu/data/dna\_r9.4.1\_450bps\_fast.cfg

model file: /usr/lib/ont-guppy-cpu/data/template\_r9.4.1\_450bps\_fast.jsn

input path: /home/ubuntu/lambda

save path: /home/ubuntu/lambda

chunk size: 1000

chunks per runner: 20

records per file: 4000

num basecallers: 1

cpu mode: ON

threads per caller: 8

Found 4000 fast5 files to process.

Init time: 3220 ms

0% 10 20 30 40 50 60 70 80 90 100%

|----|----|----|----|----|----|----|----|----|----|

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Caller time: 100783 ms, Samples called: 310564469, samples/s: 3.08152e+06

Finishing up any open output files.

Basecalling completed successfully.

**STEP 5: EXPLORING READ LENGTH AND QUALITY SCORE DISTRIBUTION**

Let’s take a look at the basecalling output by plotting the quality score and read length distribution.

1. First let’s take a look at the guppy output. Guppy will deposit successfully basecalled reads as fastqs inside the output directory (/home/ubuntu/lambda in our case). Let’s list the fastq files therein:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ ls \*.fastq**

fastq\_runid\_25867780442cd34270bea76488b29fe01ba3be54\_0.fastq

1. We will be using the fastq files to explore the read length and quality score distribution. **Note**: if you used the command above, only one fastq file will be created and concatenation becomes unnecessary. The commands below are still performed so that you are familiar with the process when working with your own data.

First let’s concatenate all the individual fastqs into one file and name it to something easier to manage:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ cat \*.fastq > pass.fastq**

The cat command aggregates the fastq files and redirects the output into a new file, in this case “pass.fastq”

Use the list command to confirm that pass.fastq was created:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ ls pass\***

pass.fastq

1. Let’s plot the read and quality score distribution of the data in pass.fastq.

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ NanoPlot --fastq pass.fastq -t 8**

**Note**: When plotting all your reads, using the sequencing\_summary.txt file as input in conjunction with the --summary flag provides more extensive plots when running a command like:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ NanoPlot --summary sequencing\_summary.txt -t 8**

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

**Windows**

Use the Bitvise file browser to open the **/home/ubuntu/lambda** folder (click on the ‘refresh’ button first). Sort by date by clicking twice on the "Date Modified" header. Double-click on the **NanoPlot-report.html** file to open it in your web browser. Drag this file to your local machine to save a copy (your desktop will be fine).

**Mac/Linux**

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/$(whoami)/Downloads/ont\_mac\_key ubuntu@10.11.12.13**

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

**sftp> get lambda/\*.html**

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-10-11-12-14:/home/ubuntu/lambda$ mkdir /home/ubuntu/filtered**

1. We will now use filtlong to filter our data. It can be run in the following manner:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ filtlong -h**

usage: filtlong {OPTIONS} [input\_reads]

Filtlong: a quality filtering tool for Nanopore and PacBio reads

positional arguments:

input\_reads input long reads to be filtered

optional arguments:

output thresholds:

-t[int], --target\_bases [int] keep only the best reads up to this many total bases

-p[float], --keep\_percent [float] keep only this percentage of the best reads (measured by bases)

--min\_length [int] minimum length threshold

--min\_mean\_q [float] minimum mean quality threshold

--min\_window\_q [float] minimum window quality threshold

external references (if provided, read quality will be determined using these instead of from the Phred scores):

-a[file], --assembly [file] reference assembly in FASTA format

-1[file], --illumina\_1 [file] reference Illumina reads in FASTQ format

-2[file], --illumina\_2 [file] reference Illumina reads in FASTQ format

score weights (control the relative contribution of each score to the final read score):

--length\_weight [float] weight given to the length score (default: 1)

--mean\_q\_weight [float] weight given to the mean quality score (default: 1)

--window\_q\_weight [float] weight given to the window quality score (default: 1)

read manipulation:

--trim trim non-k-mer-matching bases from start/end of reads

--split [split] split reads at this many (or more) consecutive non-k-mer-matching bases

other:

--window\_size [int] size of sliding window used when measuring window quality (default: 250)

--verbose verbose output to stderr with info for each read

--version display the program version and quit

-h, --help display this help menu

For more information, go to: <https://github.com/rrwick/Filtlong>

1. The script will filter the data and save the output to a new file named filtered.fq. We can output the filtered fastq directly into the **/home/ubuntu/filtered** directory:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$** **filtlong --min\_mean\_q 90 --min\_length 3000 pass.fastq > /home/ubuntu/filtered/filtered.fq**

1. Let’s plot the filtered data to confirm the process was successful. We will use Nanoplot as before, but first switch to the **/home/ubuntu/filtered** directory:

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ cd /home/ubuntu/filtered**

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$ NanoPlot --fastq filtered.fq -t 8**

**Windows**

Use the Bitvise file browser window to take a look at the NanoPlot report in the **/home/ubuntu/filtered** folder. You will have to press the “Up” button, then the refresh button before the new filtered folder will show up. Drag the file onto your own machine to save a copy.

**Mac**

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

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

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

**STEP 6: 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: <https://github.com/nanoporetech/pomoxis>

Pomoxis chains together some very effective tools developed by the nanopore community such as **minimap2**, **miniasm**, **racon**, and **porechop**. We will use **mini\_align** module for mapping the reads we’ve basecalled so far to a reference.

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

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

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$ 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).

1. We will use the following command to map the filtered data to the lambda reference (available in **/home/ubuntu/refs/lambda\_ref.fasta**):

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$**

**mini\_align -i filtered.fq -r /home/ubuntu/refs/lambda\_ref.fasta -P -p lambda -t 8**

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 8 -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 (from /home/ubuntu/filtered) 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**

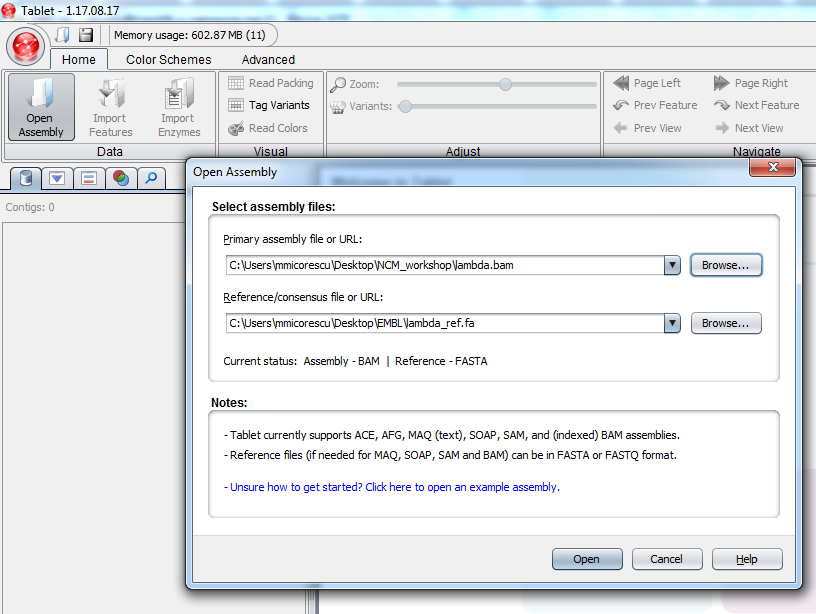
**STEP 7: VISUALISING THE ALIGNMENT USING TABLET**

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 respectively. 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.

**STEP 8: *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 change to the ubuntu user’s home directory.

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$ cd /home/ubuntu**

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

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$ mini\_assemble -h**

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

-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 assemble the fastq/fasta formatted reads and perform POA consensus.

**ubuntu@ip-10-11-12-14:/home/ubuntu$**

**mini\_assemble -i /home/ubuntu/filtered/filtered.fq -o denovo -p lambda\_denovo -m 1 -t 8 -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 the -o flag and files therein will be labeled using the –p flag with the “lambda\_denovo” tag. The -m flag will specify running one round of Racon polishing. 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 **/home/ubuntu/denovo** folder and download the **lambda\_denovo\_final.fa** file.

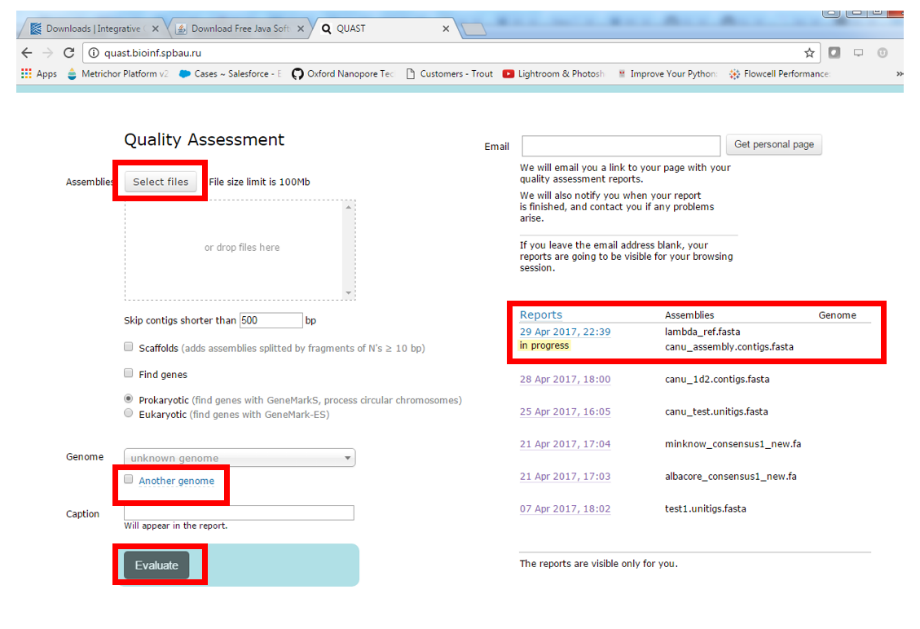
**Mac**

Use another get command in your sftp window:

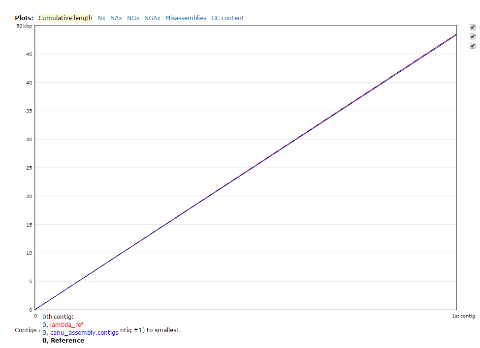
**sftp> get denovo/lambda\_denovo\_final.fa**

**STEP 9: ASSESS THE 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.



**STEP 10: ASSEMBLY POLISHING WITH MEDAKA**

We will now use Medaka to error correct our assembly using the nucleotide sequences contained in the fastq files. Medaka works by using neural networks applied from a pileup of individual sequencing reads against a draft assembly. It outperforms graph-based methods operating on basecalled data, and can be competitive with state-of-the-art signal-based methods, whilst being much faster.

1. Create a directory “polished” in the home directory for our alignment and switch to the new directory:

**ubuntu@ip-10-11-12-14:/home/ubuntu$ mkdir /home/ubuntu/polished**

**ubuntu@ip-10-11-12-14:/home/ubuntu$ cd /home/ubuntu/polished**

1. List parameters and options available in Medaka:

**ubuntu@ip-10-11-12-14:/home/ubuntu $ medaka\_consensus -h**

medaka 0.7.1

------------

Assembly polishing via neural networks. The input assembly should be

preprocessed with racon.

medaka\_consensus [-h] -i <fastx>

-h show this help text.

-i fastx input basecalls (required).

-d fasta input assembly (required).

-o output folder (default: medaka).

-m medaka model, (default: r941\_min\_high).

Available: r941\_trans, r941\_flip213, r941\_flip235, r941\_min\_fast, r941\_min\_high, r941\_prom\_fast, r941\_prom\_high.

Alternatively a .hdf file from 'medaka train'.

-t number of threads with which to create features (default: 1).

-b batchsize, controls memory use (default: 200).

It is crucially important to specify the correct model, -m in the above, according to the basecaller used. Allowed values are listed there and can also be found by running:

**ubuntu@ip-10-11-12-14:/home/ubuntu $ medaka tools list\\_models**

Available: r941\_trans, r941\_flip213, r941\_flip235, r941\_min\_fast, r941\_min\_high, r941\_prom\_fast, r941\_prom\_high

Default: r941\_min\_high

**Improving our draft assembly with Medaka**

To generate an error-corrected consensus, Medaka requires:

1. A fasta or fastq file containing the input data 🡪 we have our filtered.fq in /home/ubuntu/filtered
2. A draft assembly to be corrected 🡪 our draft assembly is in /home/ubuntu/denovo/lamda\_denovo\_final.fa
3. A specified model for medaka to use 🡪 we will use r941\_min\_fast due to our choice of basecalling model in Step 4 above
4. Run Medaka with the following parameters to improve consensus

**ubuntu@ip-10-11-12-14:/home/ubuntu $**

**medaka\_consensus -i /home/ubuntu/filtered/filtered.fq -d /home/ubuntu/denovo/lambda\_denovo\_final.fa -o /home/ubuntu/polished -m r941\_min\_fast -t 8**

**Comparing our polished and draft assemblies**

1. Next, use the “dnadiff” package within “mummer” to compare the percent identity of our polished consensus with the lambda reference:

**ubuntu@ip-10-11-12-14:/home/ubuntu/polished$**

**dnadiff /home/ubuntu/refs/lambda\_ref.fasta -p comp\_polished /home/ubuntu/polished/consensus.fasta**

1. We will use “less” to open our comparison report:

**ubuntu@ip-10-11-12-14:/home/ubuntu/polished$**

**less comp\_polished.report**

Take a look at the AvgIdentity value under [Alignments] 1-to-1 – this is a measure of how similar our polished assembly is to the reference.

To exit the **less** terminal pager, press **q**

**STEP 11: METHYLATION DETECTION WITH NANOPOLISH**

We will now use Nanopolish to detect methylated bases.

This part of the tutorial is based on the Nanopolish methylation quickstart guide (http://nanopolish.readthedocs.io/en/latest/quickstart\_call\_methylation.html), although we will use pomoxis to map our data.

**Download a human dataset**

1. Create a “mods” directory inside your home directory and change into it

**ubuntu@ip-10-11-12-14:/home/ubuntu/polished$** **mkdir /home/ubuntu/mods**

**ubuntu@ip-10-11-12-14:/home/ubuntu/polished$** **cd /home/ubuntu/mods**

1. Use wget to download the human dataset into ‘mods’, and extract it with tar

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods$ wget https://www.dropbox.com/s/1n31to1ac108apa/methylation\_example.tar.gz?dl=0 -O methylation\_example.tar.gz**

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods$** **tar -xf methylation\_example.tar.gz**

**Map this dataset to the reference using Pomoxis**

1. Change into the ‘methylation\_example’ directory

**ubuntu@ip-10-11-12-14:/home/ubuntu/** **mods$** **cd methylation\_example**

1. Next, perform the alignment using the mini\_align pipeline in Pomoxis:

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$**

**mini\_align -i /home/ubuntu/mods/methylation\_example/albacore\_output.fastq -r /home/ubuntu/mods/methylation\_example/reference.fasta -P -p methyl -t** **8**

1. Index the fastq and fast5 files as below:

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$** **nanopolish index -d fast5\_files/ albacore\_output.fastq**

**Call methylation**

1. Call methylation with nanopolish

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$**

**nanopolish call-methylation -t 8 -r albacore\_output.fastq -b methyl.bam -g reference.fasta -w "chr20:5,000,000-10,000,000" > methylation\_calls.tsv**

1. Take a look at methylation\_call.tsv using less:

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$** **less methylation\_calls.tsv**

1. The nanopolish from bioconda does not come packaged with scripts from github repository. So, we will clone the Nanopolish repository to the /home/ubuntu/git directory:

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$**

**git clone --recursive https://github.com/jts/nanopolish.git /home/ubuntu/git/nanopolish**

1. Now calculate methylation frequency:

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$**

**/home/ubuntu/git/nanopolish/scripts/calculate\_methylation\_frequency.py methylation\_calls.tsv > methylation\_frequency.tsv**

1. Examine methylation\_frequency.tsv using nano.

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$** **less methylation\_frequency.tsv**

To exit the less pager, press q

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

**APPENDIX A: TUTORIAL SUMMARY AND LIST OF COMMANDS**

1. First we made a new directory for our Lambda data, to which we downloaded and extracted the data set. A total of 4000 fast5 files were extracted, so the original archive was removed:

**ubuntu@ip-10-11-12-14:/home/ubuntu$ mkdir lambda**

**ubuntu@ip-10-11-12-14:/home/ubuntu$ cd lambda**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$**

**wget https://www.dropbox.com/s/eml7z2d82n3k8lq/lambda.tar.gz?dl=0 -O lambda.tar.gz**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ tar -xf lambda.tar.gz**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ ls | wc -l**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ rm lambda.tar.gz**

1. Next we basecalled those Lambda data using Guppy. The first two commands showed the help text and listed what config files are called for when using specific combinations of --flowcell and --kit flags.

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ guppy\_basecaller -h**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ guppy\_basecaller --print\_workflows**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$**

**guppy\_basecaller -i /home/ubuntu/lambda -s /home/ubuntu/lambda --config dna\_r9.4.1\_450bps\_fast.cfg -r --num\_callers 1 --cpu\_threads\_per\_caller 8**

1. View the output files of basecalling, concatenate output fastq files into one large pass.fastq file, then plot these data using NanoPlot.

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ ls \*.fastq**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ cat \*.fastq > pass.fastq**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ ls pass\***

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ NanoPlot --fastq pass.fastq -t 8**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ NanoPlot --summary sequencing\_summary.txt -t 8**

1. Next we filtered our data using Filtlong after creating a filtered directory for the output. After filtering, we changed into the filtered directory and plotted our filtered data with NanoPlot.

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ mkdir /home/ubuntu/filtered**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ filtlong -h**

**filtlong --min\_mean\_q 90 --min\_length 3000 pass.fastq > /home/ubuntu/filtered/filtered.fq**

**ubuntu@ip-10-11-12-14:/home/ubuntu/lambda$ cd /home/ubuntu/filtered**

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$ NanoPlot --fastq filtered.fq -t 8**

1. Reads were then aligned to the supplied Lambda reference using the mini\_align pipeline in Pomoxis. This tool uses minimap2 to map the reads, followed up samtools to create, sort, and index a **.bam** alignment file.

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$ mini\_align -h**

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$ mini\_align -i filtered.fq -r /home/ubuntu/refs/lambda\_ref.fasta -P -p lambda -t 8**

1. A denovo assembly was created using the mini\_assemble overlap-layout-consensus (OLC) pipeline in Pomoxis. This uses minimap2 to find overlaps between reads, miniasm to create a layout graph, and four rounds of racon to build a consensus.

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$ cd /home/ubuntu**

**ubuntu@ip-10-11-12-14:/home/ubuntu/filtered$ mini\_assemble -h**

**ubuntu@ip-10-11-12-14:/home/ubuntu$**

**mini\_assemble -i /home/ubuntu/filtered/filtered.fq -o denovo -p lambda\_denovo -m 1 -t 8 -c**

1. The denovo assembly was further error-corrected using Medaka. A new polished directory is made and available options/models can be found with commands 3 and 4.

**ubuntu@ip-10-11-12-14:/home/ubuntu$ mkdir /home/ubuntu/polished**

**ubuntu@ip-10-11-12-14:/home/ubuntu$ cd /home/ubuntu/polished**

**ubuntu@ip-10-11-12-14:/home/ubuntu $ medaka\_consensus -h**

**ubuntu@ip-10-11-12-14:/home/ubuntu $ medaka tools list\\_models**

**ubuntu@ip-10-11-12-14:/home/ubuntu $**

**medaka\_consensus -i /home/ubuntu/filtered/filtered.fq -d /home/ubuntu/denovo/lambda\_denovo\_final.fa -o /home/ubuntu/polished -m r941\_min\_fast -t 8**

1. The dnadiff tool packaged in MUMmer was used to determine percent identity of our polished consensus to the supplied Lambda reference.

**ubuntu@ip-10-11-12-14:/home/ubuntu/polished$**

**dnadiff /home/ubuntu/refs/lambda\_ref.fasta -p comp\_polished /home/ubuntu/polished/consensus.fasta**

**ubuntu@ip-10-11-12-14:/home/ubuntu/polished$**

**less comp\_polished.report**

1. Methylation tutorial: a new mods directory was created, to which we downloaded and extracted an example dataset for methylation analysis.

**ubuntu@ip-10-11-12-14:/home/ubuntu/polished$** **mkdir /home/ubuntu/mods**

**ubuntu@ip-10-11-12-14:/home/ubuntu/polished$** **cd /home/ubuntu/mods**

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods$ wget https://www.dropbox.com/s/1n31to1ac108apa/methylation\_example.tar.gz?dl=0 -O methylation\_example.tar.gz**

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods$** **tar -xf methylation\_example.tar.gz**

**ubuntu@ip-10-11-12-14:/home/ubuntu/** **mods$** **cd methylation\_example**

1. The mini\_align pipeline in Pomoxis was used to align the reads to the supplied reference. Nanopolish first created an index such that each read in the fastq file was tied to its corresponding raw signal in the fast5 files. Nanopolish was then used to call methylation within a 5 Mb region of chromosome 20. Methylation calls were visualized with a terminal pager: less.

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$**

**mini\_align -i /home/ubuntu/mods/methylation\_example/albacore\_output.fastq -r /home/ubuntu/mods/methylation\_example/reference.fasta -P -p methyl -t 8**

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$** **nanopolish index -d fast5\_files/ albacore\_output.fastq**

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$**

**nanopolish call-methylation -t 8 -r albacore\_output.fastq -b methyl.bam -g reference.fasta -w "chr20:5,000,000-10,000,000" > methylation\_calls.tsv**

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$** **less methylation\_calls.tsv**

1. A script to convert methylation calls to a methylation frequency was downloaded from the Nanopolish github repository, then run to output frequencies and visualized with less.

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$**

**git clone --recursive https://github.com/jts/nanopolish.git /home/ubuntu/git/nanopolish**

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$**

**/home/ubuntu/git/nanopolish/scripts/calculate\_methylation\_frequency.py methylation\_calls.tsv > methylation\_frequency.tsv**

**ubuntu@ip-10-11-12-14:/home/ubuntu/mods/methylation\_example$** **less methylation\_frequency.tsv**