Unlocking the Potential of MinION: A Hands-On Genomic Sequencing Workshop

Hands-on: Assembly of bacterial genome based on MinION sequence data

This section provides a step-by-step guide for assembling a bacterial genome using MinION sequence data. The workflow covers the entire process, from the initial quality control of raw sequencing reads to the final stages of genome annotation and phylogenetic analysis.

This Hands-on guide covers:

- Process long-read sequencing data
- Assemble a genome
- Polish the assembly for error correction
- Evaluate the assembly quality
- Annotate functional genomic features
- Place the genome in a taxonomic and phylogenetic context

1. Dataset Overview

The data set used in this tutorial is retrieved from the complete genome sequence of *Bartonella schoenbuchensis* Strain L2[1], a bacterium that causes bacteremia in ruminants and is primarily transmitted by deer keds (*Lipoptena cervi*). *Bartonella schoenbuchensis* belongs to the genus *Bartonella*, known for infecting a variety of mammalian hosts, including humans, through arthropod vectors such as fleas, lice, and ticks.

2. Open the terminal session and change the working directory

To begin, launch a terminal session and navigate to your working directory

```
cd Desktop/BacterialGenome
```

3. Activate the CONDA working environment

Begin by activating the “GenomeTools” CONDA environment that contains most of the necessary tools for the analysis. This ensures that the correct versions of software are used throughout the pipeline.

[1] Solis Cayo L, Hammerbauerová I, Sommer J, Nemati Z, Ballhorn W, Tsukayama P, Dichter A, Votýpka J, Kempf VAJ. 2024. Genome sequences of three Bartonella schoenbuchensis strains from Czechia. Microbiol Resour Announc 13:e00397-24.

```
# Check available environments
conda info --envs

# Activate GeomeTools environment
conda activate GenomeTools
```

4. Explore the sequence data

Before proceeding with the assembly, it is crucial to assess the quality of the raw sequence data.

- **NanoPlot** allows us to quickly assess the quality of the raw Fastq file. It generates reports on the number of reads, their quality, and the mean read length.

```
#unzip Fastq file
gunzip SRR28636163_30pct.fastq.gz

#List NanoPlot parameters
NanoPlot -h

#Run Nanoplot
NanoPlot --fastq SRR28636163_30pct.fastq -o Nanoplot_raw --no_static
```

Output: The output is a folder named Nanoplot_raw containing general reports in both TXT and HTML formats, along with various plots that visualize the data quality.

- **FastQC** allows us to check the average of bases quality along the length of the reads

```
# Create ouput directory
mkdir fastqc_out

#run FastQC
fastqc -o fastqc_out --memory 1000 SRR28636163_30pct.fastq
```

Output: The result is an HTML file that provides an overview of the average quality scores across all bases in your reads.

5. Remove sequencing adaptors

To remove bias that could be related to the presence of adaptors on the reads, it’s important to check their presence and remove them. **Porechop** is a tool for finding and removing adapters from the ends of Oxford Nanopore reads even at low sequence identity.

```
#Run Porechop (≈ 11min)
porechop -i SRR28636163_30pct.fastq -o porechop_out.fastq
```

Output: A new FASTQ file (porechop_out.fastq) containing trimmed sequencing reads.

6. Filter raw sequencing data

To improve the quality of the assembly, it’s important to remove reads with low average quality and short sequences. **NanoFilt** filters the raw sequencing data based on quality scores and read length.

```
#List NanoFilt parameters
NanoFilt -h

#Run NanoFilt
NanoFilt -q 10 porechop_out.fastq > filtered.fastq
```

Output: A new FASTQ file (filtered.fastq) containing only high-quality reads.

7. Check quality after adaptor trimming and filtering

After filtering, it's important to re-assess the quality of your data to ensure the filtering process was successfully performed.

```
#Run NanoPlot on filtered data
NanoPlot --fastq filtered.fastq -o Nanoplot_filt --no_static

#Run FastQC on filtered data
fastqc -o fastqc_out --memory 1000 filtered.fastq
```

8. Assembly with Flye

With the high-quality reads prepared, you can now proceed to assemble the bacterial genome. **Flye** assembler is designed for long-read data and well-suited for assembling genomes from MinION data.

```
#Run Flye (> 2h using 4CPUs and 6GB of RAM)
flye --nano-raw filtered.fastq --out-dir Assembly/

## outputs/flye_out/ folder can be used for the rest of the analysis
# Copy pre-assembled genome into the working directory
cp -r Outputs/Assembly .
```

Output: The output folder **Assembly** contains the assembled contigs in FASTA format (**assembly.fasta**) and various intermediate and log files.

9. Assembly polishing

Polishing is necessary to correct errors in the assembly, particularly those common in long-read data. **Racon** (or a similar tool like Medaka) can be used to polish the assembly by aligning the raw reads back to the assembly and correcting any errors.

9.1. Map reads to the assembled genome

Align the raw sequencing reads back to the assembled genome to identify and correct any errors. This can be done using **minimap2**. (-a: Output in SAM format (Sequence Alignment/Map).

```
minimap2 -a -t 4 Assembly/assembly.fasta filtered.fastq > Assembly/Map.sam
```

Output: A SAM file (Map.sam) that stores the mapping of reads to the assembled genome.

9.2. Run Racon for polishing

Next, the mapping is used by **Racon** to polish the genome by correcting errors based on the mapped reads.

```
racon -t 4 -m 8 -x -6 -g -8 -w 500 filtered.fastq Map.sam
Assembly/assembly.fasta > Assembly/assembly_polished.fasta
```

Output: A polished genome assembly (assembly_polished.fasta), with corrections made based on the raw reads.

10. Assembly metrics and completeness

How well has the genome been assembled? To evaluate the assembly's quality, we can examine several key metrics that provide insight into whether the assembly meets our expectations for contiguity and completeness.

Quast: offers a detailed assessment of the assembly's quality by reporting on metrics such as the number of contigs, N50 value, total assembly length, and genome coverage.

```
quast.py --output-dir Quast_out/ Assembly/*.fasta
```

CheckM: To evaluate the completeness and contamination of the genome. **CheckM** estimates these metrics based on the presence and redundancy of taxon-specific marker genes (*Bartonella* genus). The list of available taxa in different levels can be found using *checkm taxon_list* command

```
#List available taxon list
checkm taxon_list

#Run checkm (≈ 1min)
checkm taxonomy_wf -t 4 -x fasta genus Bartonella Assembly/ CheckM_out/
```

Output: CheckM returns the completeness and the contamination metrics as percentages directly on the terminal. However, CheckM_out directory contains the various intermediate files and logs created during the process.

11. Genome Annotation with Prokka

After assembling and polishing the genome, the next crucial step is to annotate it. Genome annotation involves identifying and labeling various genomic features such as genes, coding sequences (CDS), rRNAs, tRNAs, and other functional elements. **Prokka** is a widely-used tool for rapid annotation of prokaryotic genomes. It automatically identifies genes and other features in the genome and assigns functional annotations.

```
prokka --outdir Annotation --prefix L2 Assembly/assembly_polished.fasta
```

Output: The **Annotation** directory contains a set of files that describe the annotated genome:

- L2.gff: A General Feature Format file, which provides a detailed map of annotated features in the genome.
- L2.gbk: A GenBank file format that includes annotations and can be visualized in genome browsers.

- L2.faa: A FASTA file containing the predicted protein sequences.
- L2.ffn: A FASTA file containing nucleotide sequences of the coding regions.
- L2.sqn: A file formatted for submission to NCBI.
- L2.txt: A summary report of the annotation process.

12. Taxonomy Classification

After annotating the genome, the next step is to classify the organism taxonomically. This involves comparing the genome to a database of reference genomes to identify its closest relatives. One common method is calculating the Average Nucleotide Identity (ANI) between the assembled genome and known reference genomes.

11.1. Unzip reference genomes assemblies

Before starting the classification, unzip the reference genomes to make them accessible for comparison.

```
gunzip Bartonella_References/*.gz
```

11.2. Prepare a list of reference genomes

Create a PATH list of the unzipped reference genomes that will be used in the ANI analysis.

```
ls -d -1 $PWD/ Bartonella_References/*.fna > GenomeList.txt
```

11.3. Run ANI Calculation

Use **fastANI** to calculate the ANI between the assembled genome and the reference genomes.

```
fastANI -q Assembly/assembly_polished.fasta --rl GenomeList.txt -o
fastani_out.tsv
```

Output: A file (**fastani_out.tsv**) with ANI values, which shows the genomic similarity between the query genome and reference genomes. Higher ANI values indicate closer taxonomic relatedness.

13. Phylogenetic Tree Construction

12.1. Build Phylogenetic Tree Using Mashtree

Use **Mashtree** to generate a phylogenetic tree by comparing the genome against reference genomes.

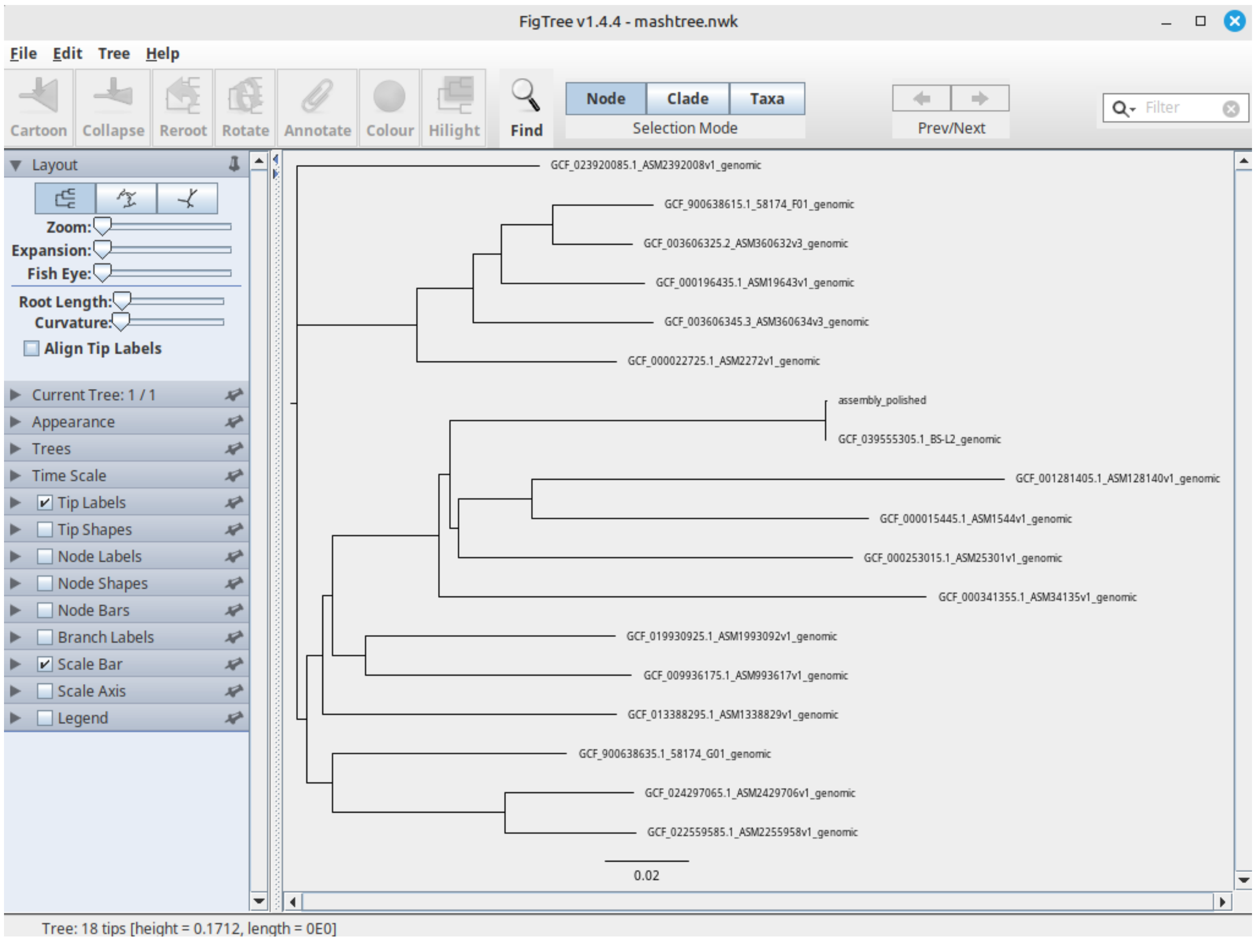
```
mashtree --numcpus 4 assembly_polished.fasta Bartonella_References/*.fna > mashtree.nwk
```

Output: Newick file (**mashtree.nwk**), which is a standard format for representing phylogenetic trees.

12.2. Visualize the Tree

Once the tree is generated, it can be visualized using a tool like **Figtree** (Figure 1).

```
figtree mashtree.nwk
```


FigTree v1.4.4 - mashtree.nwk
File Edit Tree Help
Cartoon Collapse Reroot Rotate Annotate Colour Hilight Find
Node Clade Taxa
Selection Mode
Prev/Next
Filter
Layout
Zoom:
Expansion:
Fish Eye:
Root Length:
Curvature:
Align Tip Labels
Current Tree: 1 / 1
Appearance
Trees
Time Scale
Tip Labels
Tip Shapes
Node Labels
Node Shapes
Node Bars
Branch Labels
Scale Bar
Scale Axis
Legend
GCF_023920085.1_ASM2392008v1_genomic
GCF_900638615.1_58174_F01_genomic
GCF_003606325.2_ASM360632v3_genomic
GCF_000196435.1_ASM19643v1_genomic
GCF_003606345.3_ASM360634v3_genomic
GCF_000022725.1_ASM2272v1_genomic
assembly_polished
GCF_039555305.1_BS-L2_genomic
GCF_001281405.1_ASM128140v1_genomic
GCF_000015445.1_ASM1544v1_genomic
GCF_000253015.1_ASM25301v1_genomic
GCF_000341355.1_ASM34135v1_genomic
GCF_019930925.1_ASM1993092v1_genomic
GCF_009936175.1_ASM993617v1_genomic
GCF_013388295.1_ASM1338829v1_genomic
GCF_900638635.1_58174_G01_genomic
GCF_024297065.1_ASM2429706v1_genomic
GCF_022559585.1_ASM2255958v1_genomic
0.02

```
Tree: 18 tips [height = 0.1712, length = 0E0]
```

Figure 1: Figtree user interface