Practical 4: Genome assembly and annotation

4.1 Learning outcomes

1. Perform *de novo* genome assembly using both short-read and long-read sequencing data
2. Evaluate the quality of genome assemblies using QUAST
3. Annotate genome assemblies using Prokka

4.2 Introduction

Genome assembly is a key step in genomic analysis, enabling researchers to reconstruct the complete genomic sequence of an organism from raw sequencing data. This process is particularly significant for bacterial genomes, which are typically compact, circular, and rich in essential genetic information. Understanding bacterial genome assembly is essential for applications ranging from pathogen identification and antibiotic resistance profiling to studying microbial evolution and ecology. Genome assembly can broadly be categorised into two main approaches: *de novo* assembly, relies solely on overlaps or *k*-mers, and reference-based assembly, aligning reads to a known reference genome (**Figure 4.1**).

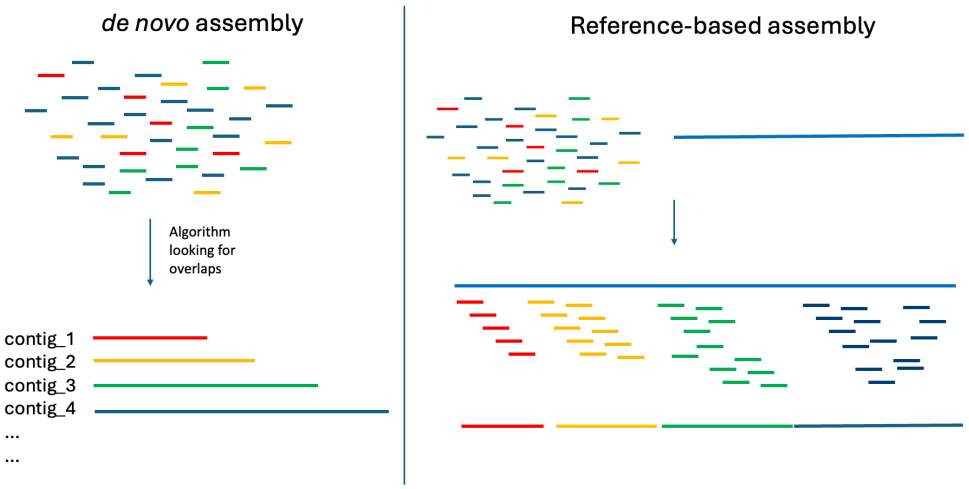

Figure 4.1 Two main methods for genome assembly from raw sequencing data.

The desired outcome is to achieve long contiguous DNA sequences (contigs). We will have to try to put fragmented DNA sequences back to its original, ordered continuous state (whether it is linear or circularized DNA), and this process is called genome assembly. The fundamental principal of genome assembly is to look for overlaps between fragments, and you will appreciate why longer reads are advantageous here (**Figure 4.2**).

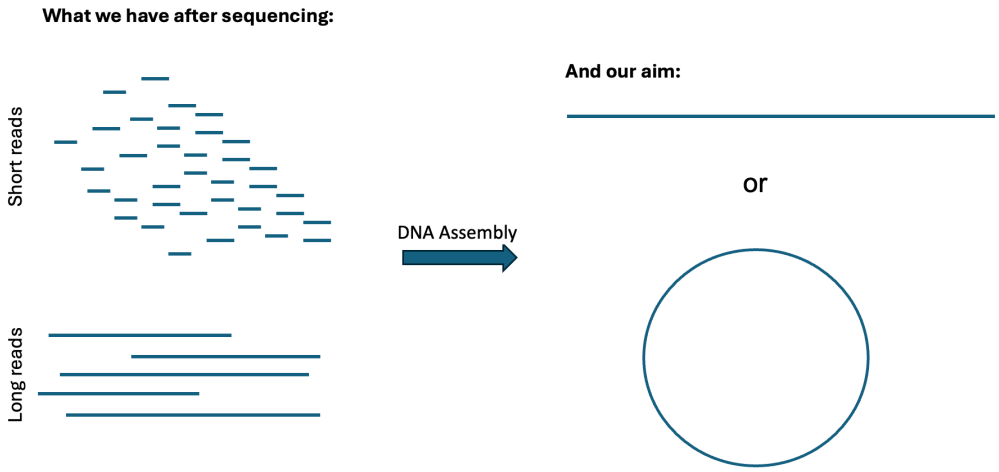

Figure 4.2 Desired outcome of genome assembly of raw sequencing data

De Bruijn graphs

Short-read assemblers predominantly rely on de Bruijn graphs, which break the sequencing reads into fixed-lengths called *k*-mers. Overlaps between *k*-mers are identified and represented as edges in a graph, with *k*-mers as nodes. However, challenges arise with repetitive regions, which can create ambiguities in the graph structure.

String graphs

Long-read assemblers use string graphs, which directly represent overlaps between entire reads instead of breaking them into *k*-mers. This method is well suited for long-read data, as the greater read lengths can span across repetitive regions and reduce fragmentation. String graphs allow for a more straightforward assembly process but requires accurate overlap detection and can also require error correction, particularly if using older long-read sequencing datasets. The resulting assemblies if often more contiguous, capturing complex genomic regions that are challenging for short-read assemblers.

Assembly graph vs final assembly

The two main files produced by genome assemblers are an assembly graph (.gfa) and the final genome assembly (.fasta). The assembly graph represents an intermediate output of the assembly process, detailing the connections between reads or contigs as nodes and edges in a graphical format. It includes unresolved regions, repetitive sequences, and possible alternative paths that are not yet fully resolved into a linear sequence. In contrast, the final assembly file provides a polished and linearised sequence of contigs, representing the best approximation of the bacterial genome after resolving ambiguities and filtering out alternative paths.

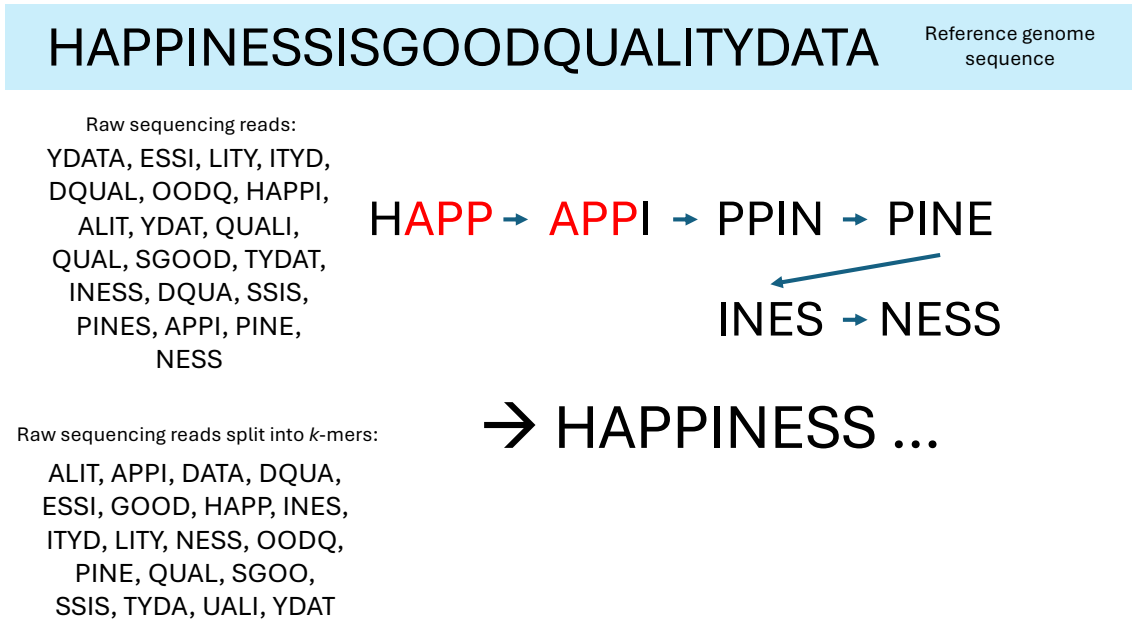

Figure 4.3

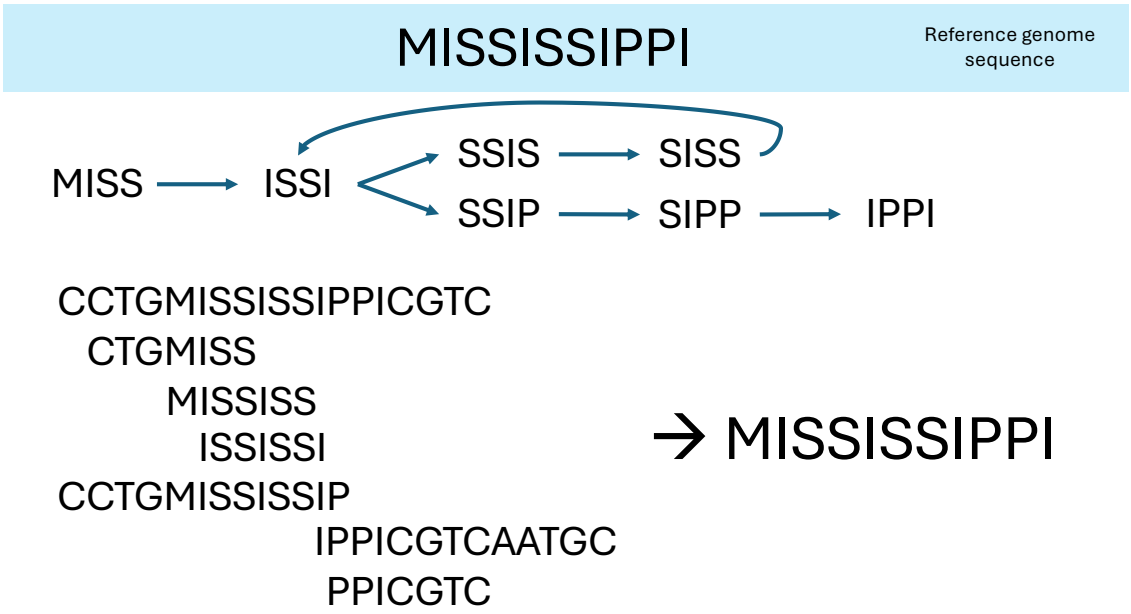

Figure 4.4

4.3 Software check

To check the tools required for this practical are correctly installed, open a new terminal window and run the following commands in turn.

```
shovill -h
flye -h
unicycler -h
quast -h
prokka -h
```

If all are installed correctly, you should see the help page of each tool, and no error.

Navigate to the home directory:

```
cd ~
```

Make a new working directory – this is where you will perform all commands

```
mkdir cp4_work
cd cp4_work
```

4.4 Dataset

In this practical, we will be using fastq files generated by both short-read paired-end Illumina sequencing, as well as long-read Nanopore sequencing, for a single sample (**cpe004**). The Illumina data was retrieved from accession ERR4095909, and the long-read data from accession ERR8282741.

The fastq read files have already been processed with QC and filtering steps using the tools and commands shown in **Table 4.1** below. Specifically, the Illumina paired-end files were processed using the same *fastp* command you used in Practical 2: Accessing Data and QC. The Nanopore long-reads have had adapters removed with *Porechop*. then filtered both on quality and read length with *Filtlong*, retaining only the best quality reads of minimum 1000 bp length. The commands used are in **Table 4.1**:

Table 4.1 Commands used to pre-process the data

Fastp	`fastp --in1 ERR4095909_1.fastq.gz --in2 ERR4095909_2.fastq.gz --out1 cpe004_R1.fastq.gz --out2 cpe004_R2.fastq.gz --length_required 40 --cut_front --cut_tail --cut_mean_quality 25`
Porechop	`porechop -i cpe004_long.fastq.gz -o cpe004_porchop.fastq.gz`
Filtlong	`filtlong --min_length 1000 --keep_percent 95 cpe004_porechop.fastq.gz | gzip > cpe004_filtered.fastq.gz`

The files we will be using for this practical are listed below and are in the `practical4/reads/` folder:

1. **cpe004_R1.fastq.gz** = Read 1 (trimmed) Illumina
2. **cpe004_R2. fastq.gz** = Read 2 (trimmed) Illumina
3. **cpe004_filtered.fastq.gz** = ONT long-reads

4.5 Short-read assembly (Shovill)

Assemble bacterial isolate genomes from Illumina paired-end reads
Github: https://github.com/tseemann/shovill
SPAdes assembler: https://doi.org/10.1002/cpbi.102

First, we will perform the genome assembly using only the short-read data. This will be performed using the Shovill tool which uses the SPAdes assembler.

1. Create a new screen session for Shovill:

```
screen -S shovill
```

2. Then run the command (this could take up to 15 minutes to complete):

```
shovill --cpus 4 --outdir shovill/ --R1
../practical4/reads/cpe004_R1.fastq.gz --R2
../practical4/reads/cpe004_R2.fastq.gz && cp
shovill/contigs.fa cpe004_shovill.fasta && cp
shovill/contigs.gfa cpe004_shovill.gfa
```

Breakdown of the command:

`--cpus`	Number of CPUs to allocate to this task
`--outdir`	Name of directory where results will be output
`--R1`	Read 1 sequencing file
`--R2`	Read 2 sequencing file
`../`	Move up one directory
`&&`	Join the next command
`cp`	Copy and rename output files

Shovill produces several files in the output folder. The only two files we will use are:

`contigs.fa`	The final assembly file to be used

`contigs.gfa` Assembly graph used to visualise the assembly

3. Count how many contigs the Shovill assembly has produced with this command:

```
grep -c ">" cpe004_shovill.fasta
```

4. The assembly file should be ordered by contig size, so the largest contig will be reported at the top. Check the output of the first few lines of the assembly file with this command:

```
head -n 4 cpe004_shovill.fasta
```

5. Use the Bandage tool to visualise the assembly graph produced by Shovill.

Open the Bandage software and select File > Load graph > navigate to and select **cpe004_shovill.gfa** > Click on Draw graph

The result should look like the image in **Figure 4.3** below.

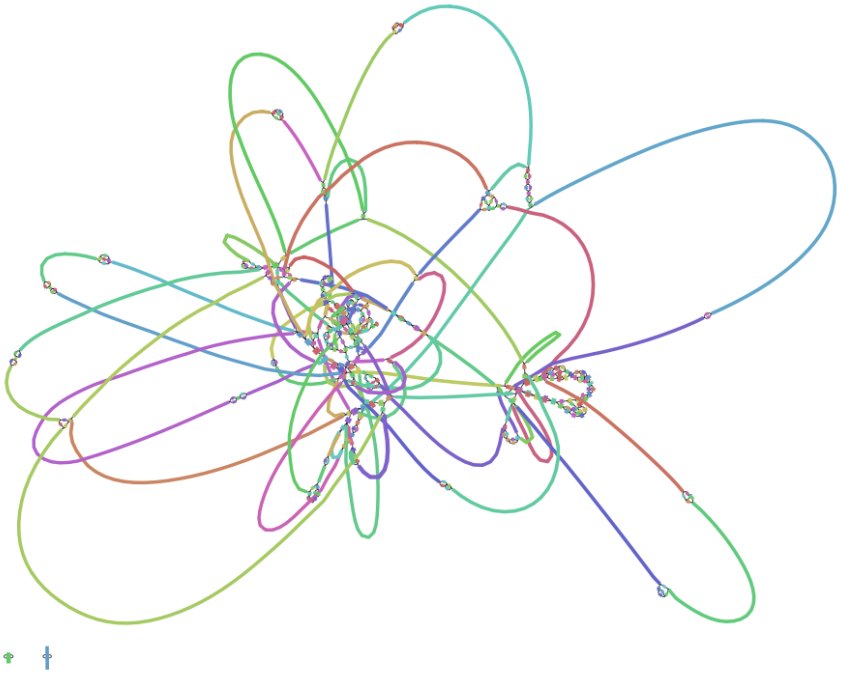

Figure 4.5 Visualising the cpe004_shovill assembly graph file in Bandage.

4.6 Long-read assembly (Flye)

Manuscript: Assembly of long, error-prone reads using repeat graphs.
DOI: https://doi.org/10.1038/s41587-019-0072-8
Github: https://github.com/fenderglass/Flye

Long-read sequencing has several advantages over short-read sequencing. Firstly, long reads can span over longer genomic regions and thus provide more contiguous sequences, leading to less fragmentation of the genome. This is particularly useful for resolving complex genomic regions such as repetitive sequences, structural variations, and regions with high GC-content. Additionally, long-read sequencing can detect larger structural variations and can be used to phase haplotypes, allowing for the study of the inheritance of genomic regions. Moreover, long-read sequencing can also facilitate the detection and characterization of previously unknown or unannotated elements such as large transposable elements, viral sequences and mitochondrial DNA. Finally, long-reads also enable to sequence and assemble genomes from single cells, providing insights into the diversity and structure of genomes in mixed populations.

Perform genome assembly using only the long reads for the same sample.

1. Detach from the shovill screen by pressing: `CTRL + A + D`

2. Create new screen for running Flye: `screen -S flye`

3. Run the following command:

```
flye -t 4 -o flye/ --nano-raw
../practical4/reads/cpe004_filtered.fastq.gz && cp
flye/assembly.fasta cpe004_flye.fasta && cp
flye/assembly_graph.gfa cpe004_flye.gfa
```

This will now proceed through the stages of genome assembly with Flye, which should take around 15 minutes for this sample. The resulting files of interest are again the .fasta assembly file and the .gfa assembly graph

Let this run and proceed onto the next section…

4.7 Hybrid-assembly (Unicycler)

Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads

DOI: https://doi.org/10.1371/journal.pcbi.1005595

https://github.com/rrwick/Unicycler

Unicycler can be used for both short-read only assembly, as well as with long-read data to produce a "hybrid" genome assembly. We can use the same long-read sequencing file from the previous section with Flye, in combination with the corresponding short-read files that were used with Shovill. Unicycler will first create a short-read only assembly using SPAdes, then use the long reads to overlay on top of the assembly to hopefully resolve some of the regions.

1. Detach from Flye screen and create new screen for Unicycler:

```
CTRL+A+D
screen -S unicycler
```

2. Perform hybrid genome assembly using the following command:

```
unicycler -t 4 -o unicycler/ --mode bold -1
../practical4/reads/cpe004_R1.fastq.gz -2
../practical4/reads/cpe004_R2.fastq.gz -l
../practical4/reads/cpe004_filtered.fastq.gz && cp
unicycler/assembly.fasta cpe004_hybrid.fasta && cp
unicycler/assembly.gfa cpe004_hybrid.gfa
```

Command breakdown:

`-t`	Number of CPUs to allocate to this task
`-1`	Read 1 sequencing file
`-2`	Read 2 sequencing file
`-l`	Long-read sequencing file
`../`	Move up one directory
`-o`	Output directory
`&&`	Join the next command
`cp`	Copy and rename output files

Once the assembly is complete:

3. Use *grep* to display all contig header names and determine the number of contigs

4. Use Bandage to visualise the assembly graph

How does this compare to the short-read assembly?

4.8 Autocycler (long-read only assembly)

Autocycler: generating consensus long-read assemblies for bacterial genomes.

https://github.com/rrwick/Autocycler/wiki

Autocycler is a recently developed tool for long-read only bacterial genome assembly, built on the principles of Trycycler. It works by leveraging multiple long-read assemblies (e.g. Flye, Miniasm, NextDenovo, and Raven) to generate individual assemblies and then combining them to produce a high-quality consensus assembly.

This method is particularly effective given the improved accuracy of long-read sequencing technologies such as Oxford Nanopore (ONT). By using only long-read data, Autocycler ensures a more complete and accurate bacterial assembly, making it an excellent choice for researchers relying on long-read sequencing platforms.

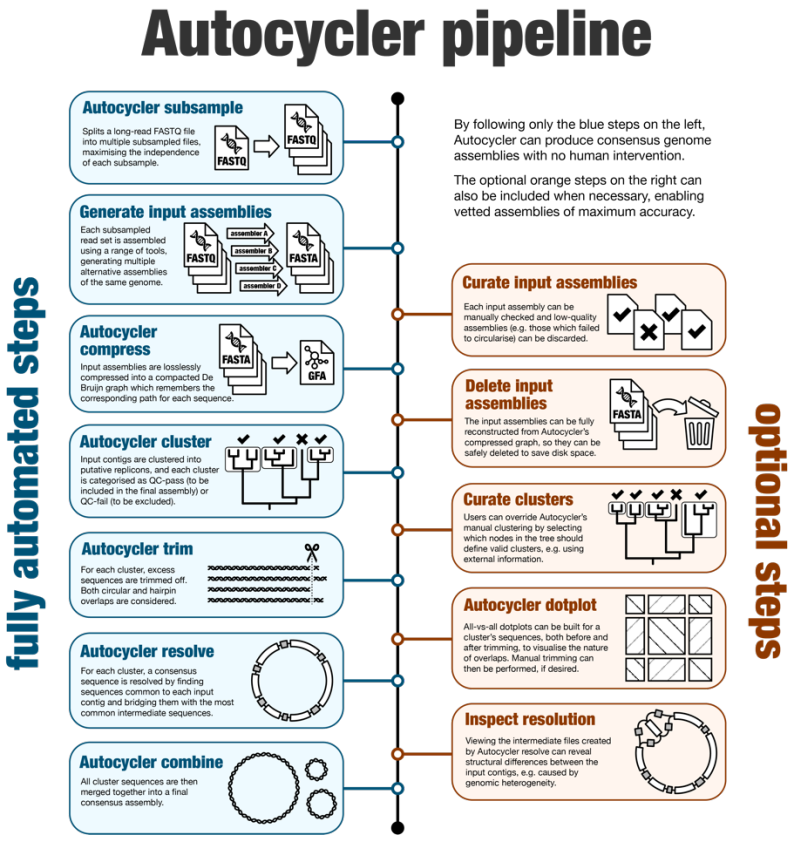

Figure 4.6 Autocycler illustrated pipeline overview

4.9 Assembly quality check

Genome assembly evaluation tool
DOI: 10.1093/bioinformatics/bty266
DOI: 10.1093/nar/gkad406
Github: https://github.com/ablab/quast

Quality Assessment Tool for Genome Assemblies (QUAST) provides a comprehensive evaluation of genome assemblies to assess contiguity and completeness. It can compare assemblies against a reference when available, highlighting misassembles, mismatches, and gaps. By integrating these quality metrics, researchers can identify areas for improvement and compare the performance of different assembly tools.

Table 4.2 Definitions of genome assembly quality checks

Metric	Description
Contigs	Number of contiguous DNA sequences assembled
Total length	Combined length of all contigs in an assembly
N50	Length of the shortest contig such that 50% of the assembly is covered by contigs of this length or longer
N90	As above but 90% of assembly
L50	Minimum number of contigs required to cover 50% of the total assembly length
L90	As above but 90% of assembly

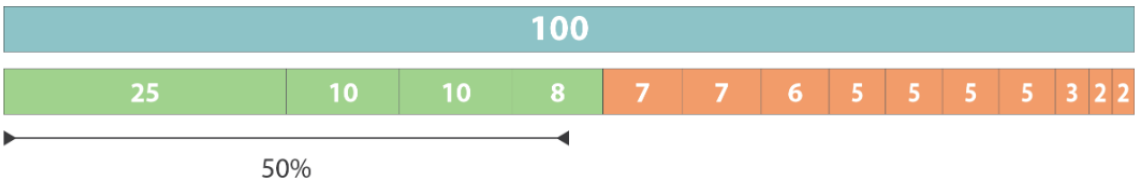

Figure 4.7 Example genome of 100 bp length with the contigs assembled.

Run QUAST on all three assemblies generated in this practical:

```
quast cpe004_shovill.fasta cpe004_flye.fasta
cpe004_hybrid.fasta -o quast/
```

View the results by opening the file `quast/report.html`

4.10 Genome annotation

Prokka: rapid prokaryotic genome annotation
DOI: 10.1093/bioinformatics/btu153
Github: https://github.com/tseemann/prokka

Bacterial genome annotation is an important process for interpreting raw genomic data, transforming it into biologically meaningful insights. The process involves structural annotation, which identifies genomic features such as coding sequences (CDS), regulatory regions, and non-coding RNA (e.g. rRNA/tRNA), and functional annotation, which assigns putative roles to the elements based on comparative genomics and database analyses.

Tools such as Prokka have revolutionised this workflow by offering an automated, high-throughput solution for generating annotated genome files. Utilising genome assembly fasta files, Prokka integrates curated databases and advanced algorithms to deliver precise predictions and functional assignments, facilitating robust and reproducible annotation.

The annotated genome files, commonly formatted as GFF3 (.gff), serve as foundational resources for a wide range of downstream applications. These include comparative genomics, phylogenetic analyses, as well as aiding in the development of diagnostic tools.

1. Use the following Prokka command to annotate the **cpe004_hybrid.fasta** assembly:

```
prokka --cpus 4 --prefix cpe004_hybrid --outdir prokka/ cpe004_hybrid.fasta
```

Prokka produces several files:

Table 4.3 Output files resulting from Prokka genome annotation.

Extension	Description
.gff	This is the master annotation in GFF3 format, containing both sequences and annotations. It can be viewed directly in Artemis or IGV.
.gbk	This is a standard Genbank file derived from the master .gff. If the input to prokka was a multi-FASTA, then this will be a multi-Genbank, with one record for each sequence.
.fna	Nucleotide FASTA file of the input contig sequences.
.faa	Protein FASTA file of the translated CDS sequences.
.ffn	Nucleotide FASTA file of all the prediction transcripts (CDS, rRNA, tRNA, tmRNA, misc_RNA)
.sqn	An ASN1 format "Sequin" file for submission to Genbank. It needs to be edited to set the correct taxonomy, authors, related publication etc.
.fsa	Nucleotide FASTA file of the input contig sequences, used by "tbl2asn" to create the .sqn file. It is mostly the same as the .fna file, but with extra Sequin tags in the sequence description lines.
.tbl	Feature Table file, used by "tbl2asn" to create the .sqn file.
.err	Unacceptable annotations - the NCBI discrepancy report.
.log	Contains all the output that Prokka produced during its run.
.txt	Statistics relating to the annotated features found.
.tsv	Tab-separated file of all features

2. Display the top 20 lines of the **cpe004_hybrid.gff** file

3. Search if any beta-lactamase resistance genes "*bla*" are present in this genome

4.11 Extra sample for genome assembly

Sequencing data for an extra bacterial isolate is available in **practical_4/extra_isolate/** folder:

1. **bc01_R1.fastq.gz** = Read 1 (trimmed) Illumina
2. **bc01_R2. fastq.gz** = Read 2 (trimmed) Illumina
3. **bc01_long.fastq.gz** = ONT long-reads

*(This folder also contains the output of the hybrid assembly using Unicycler, **bc01_hybrid.fasta** and **bc01_hybrid.gfa**, as this step might take a long time. But first try running Unicycler yourself!)*

Task: perform short-read and hybrid genome assembly for the isolate "bc01"

(Make sure to perform all analyses in your own separate working directory as before)

1. Display the header for all contigs produced in the resulting fasta assembly files
2. Determine the largest contig size in the assembly files
3. Visualise the output assembly graph in Bandage
4. Use QUAST to compare the two assemblies

How does this compare to the other genomes?