1 Pangenome Construction using Roary

1.1 Introduction

When analysing prokaryotic genomes it is important to take into account the differences in gene content resulting from horizontal gene transfer, gene duplication and gene loss. Such differences can play a role in increased virulence or antimicrobial drug resistance. Aligning sequence data for an isolate to a single reference genome can fail to incorporate non-reference sequences XXX.

One approach to ovecome this limitation is to perform a pangenome analysis of your isolates. The pangenome is the set of all genes that have been found in a set of isolates. Within the pangenome, genes are often then described as being part of the 'core' genome, the set of genes present in all isolates, or the non-core ('accessory') genome. Gaining a better picture of the conserved genes of a species, and the accessory genome, can lead to a better understanding of key processes such as selection and evolution.

There are several tools avaiable for pangenome analysis, including Panaroo and Roary. In this tutorial we will demonstarte how Roary can be used to perform a pangenome analysis. For more in depth information about Roary, please feel free to have a look the Roary paper:

> **Roary: Rapid large-scale prokaryote pan genome analysis**
> Andrew J. Page, Carla A. Cummins, Martin Hunt, Vanessa K. Wong, Sandra Reuter, Matthew T. G. Holden, Maria Fookes, Daniel Falush, Jacqueline A. Keane, Julian Parkhill
> *Bioinformatics, 2015;31(22):3691-3693 doi:10.1093/bioinformatics/btv421*

A copy of the paper can be found at

```
~/course_data/pangenome/roary_paper.pdf
```

Or visit the Roary manual at http://sanger-pathogens.github.io/Roary/.

1.2 Learning outcomes

By the end of this tutorial you can expect to be able to:

- Describe what a pangenome is
- Use Roary to construct a pangenome
- Understand the different output files produced by Roary
- Draw a basic tree from the core gene alignment produced by Roary
- Query the pangenome results produced by Roary
- Use Phandango to visualise the results produced by Roary
- Generate an annotated genome assembly

1.3 Tutorial sections

This tutorial comprises the following sections:
1. What is a pan genome
2. Preparing the input data for Roary
3. Performing QC on your data
4. Constructing a pangenome with Roary
5. Exploring the results
6. Visualising the results with Phandango 7. Creating genome assemblies

1.4 Authors and License

This tutorial was created by Jacqui Keane and Sara Sjunnebo.

The content is licensed under a Creative Commons Attribution 4.0 International License (CC-By 4.0).

1.5 Running the commands from this tutorial

You can follow this tutorial by running all the commands you see in a terminal window on your computer. Remember, the terminal window is similar to the "Command Prompt" window on MS Windows systems, which allows the user to type DOS commands to manage files.

To get started, open a terminal window and type the command below folowed by the `return` key:

[ ]:
```
cd ~/course_data/roary/data
```

1.6 Prerequisites

This tutorial assumes that you have the following software and their dependencies installed on your computer. The software used in this tutorial may be updated from time to time so, we have also given you the version which was used when writing this tutorial.

Package name	Link for download/installation instructions	Version
Prokka	https://github.com/tseemann/prokka	1.14.6
Roary	https://github.com/sanger-pathogens/roary	3.13.0
Fastree	http://www.microbesonline.org/fasttree/	2.1.11
Spades	https://github.com/ablab/spades	3.15.5
assembly-stats	https://github.com/sanger-pathogens/assembly-stats	1.0.1

The easiest way to install the required software is using `conda`, a software package manager. These software have already been installed on the computer for you. To activate them run:

[ ]:
```
conda activate pangenome
```

After the software is activated run the following commands:

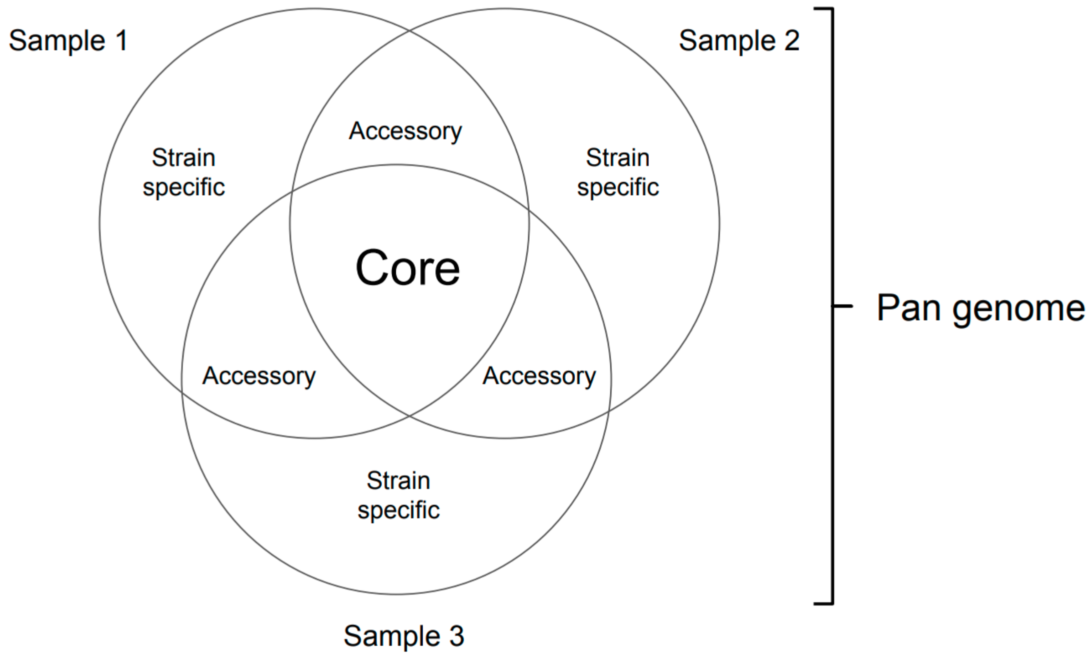
[ ]:
```
prokka --help
roary --help
fasttree -h
spades.py -h
assembly-stats
```

This should return the help messages for all the software tools you will use in this tutorial.

To get started with the tutorial, go to the first section: What is a pangenome?

2 The Pangenome explained

The pangenome for a proakryote population is the complete set of genes that it contains. This includes genes present in all isolates, and genes that are present in only a subset of isolates (even just one isolate). The subset of genes present in all of the isolates is called the **core genome**. These are often highly conserved genes with important functions, for instance housekeeping genes. The subset of genes that are not present in all, but in two or more of the isolates, is called the **accessory genome**. The accessory genome often contains genes that have been transferred between baterial strains, for example genes linked to virulence or drug resistance. Genes present in only one of the isolates can be referred to as **strain specific**.



The pan, core and acessory genomes can provide important insight into the genetic structure of prokaryotic genomes. By analysing the pangenome we can gain a better understanding of key processes like evolution and selection. Roay is a software tool that allows you to calculate the pangenome for a set of isolates from a set of annotated genome assemblies. It is fast and accurate and can conveniently be run on most modern PCs. In this tutorial we are going to guide you through a complete pangenome analysis, starting with annotation of the genome assemblies, construction of the pangenome, and finally visualisation of the results.

2.1 Check your understanding

Q1: The pangenome contains:
a) Only genes present in one isolate in a population
b) All genes from all isolates in a population
c) Only genes present in all isolates in a population

Q2: Core genes are:

a) Often important for basic cell functions
b) Present in only a subset of the isolates of a population
c) Often related to drug resistance

Now that you have an understanding of pangenomes, go to the next session: Preparing the input data for Roary

3 Preparing the input data for Roary

`Roary` requires annotated genome assemblies in GFF3 format as input.

A **genome assembly** is an attempted reconstruction of the complete DNA sequence of an organism's genome from the short DNA fragments generated by high-throughput sequencing technologies.

An **annotated genome assembly** is a genome assembly that has been further analyzed to identify the various functional elements present in the DNA sequence. It contains information about genes, regulatory elements, non-coding regions, repetitive elements, and other biologically relevant features.

For this tutorial, we have pre-prepared the genome assemblies of three *Streptococcus pneumoniae* isolates using sequence data obtained from one of the public sequence archives, the ENA (European Nucleotide Archives). The three assembled *S. pneumoniae* genomes are located in a directory called "assemblies".

[ ]:
```
ls assemblies
```

These data and assemblies are also available for download from the ENA and the accession numbers for the assemblies and the sequence data are included below for reference.

Name	Genome Accession	Data Accession
sample1	GCA_900194945.1	ERR657006
sample2	GCA_900194155.1	ERR657305
sample3	GCA_900194195.1	ERR657310

3.1 Annotating genome assemblies

We must now annotate our genome assemblies to produce GFF3 files for `Roary`. The GFF3 files must include the nucleotide sequence at the end of the file, and to make it easier to identify which isolate each gene came from, each GFF3 file should have a unique locus tag (identifier) for the genes.`Prokka` is a tool that performs genome annotation. All GFF3 files created by `Prokka` are valid with `Roary` and therefore this is the recommended way of generating the input files.

To run Prokka on a single file using the default settings, you would use something like:

```
prokka sample1.fasta
```

If you have many assemblies, running this for each sample will soon become tedious. Instead, we will use a for-loop to run `Prokka` on all the fasta files in the assemblies directory.

[ ]:
```
for F in assemblies/*.fasta; do FILE=${F##*/}; PREFIX=${FILE/.fasta/}; \
    prokka --locustag $PREFIX --outdir annotated_$PREFIX \
    --prefix $PREFIX $F; done
```

For each fasta file in the assemblies directory, this will set FILE to be the filename without the text 'assemblies/' (e.g. sample1.fasta) and set the value of PREFIX to be the text found before '.fasta' (e.g. sample1) and the value of $F will be the path to the fasta file (assemblies/sample1.fasta). We have also used the following options for Prokka:

Option	Description
–locustag	The locus tag prefix
–outdir	The directory to put the output
–prefix	The prefix for the output files

By providing a unique value for the **–locustag** option we make it easier to identify which sample different genes came from when we look at the results from Roary. The **–outdir** and **–prefix** options will make it easier for us to keep track of our files.

This is going to take around 5 minutes or longer to run, so be patient. Perhaps read the next section Performing QC on your data come back here when Prokka is finished running.

Once this has finished, you should have three new directories called annotated_sample1, annotated_sample2 and annotated_sample3. Have a look to see that it worked:

[ ]:
```
ls -l
```

[ ]:
```
ls -l annotated_sample1
```

As you can see, for sample1 we now have a number of annotation files. There is more information about the different output files, along with information about other usage options, on the Prokka GitHub page (https://github.com/tseemann/prokka). For now, we are only interrested in the GFF files that were generated as this is what we are going to use as input for Roary.

3.2 Check your understanding

Q3: Why do we need to run Prokka?
a) It will perform QC on our data
b) It will annotate our data
c) We don't, Roary can handle fasta files as input

Q4: Why do we use the –locustag option when we run Prokka?
a) To make it easier to keep track of the output files
b) Because Roary won't work without it
c) To make the Roary results easier to interpret

Now continue to the next section of the tutorial: Performing QC on your data.

4 Performing QC on your data

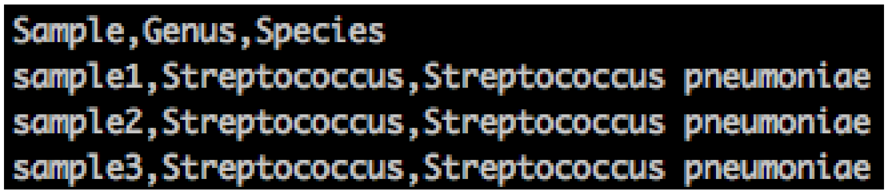
The results you can get from any analysis will only ever be as good as the data you put into it. To avoid spending countless hours performing analysis without receiving any satisfactory results, or worse yet erroneous or misleading results, it is important to QC your data before starting. There are a number of checks you can make to ensure you are dealing with good quality data, and we will walk you through some of them here.

4.1 Contamination

In order to get meaningful results from `Roary`, the samples should be closely related. If you have lots of contamination in your data, for instance if one of your samples is from a different species, you will get very few genes in your core genome, if any at all.

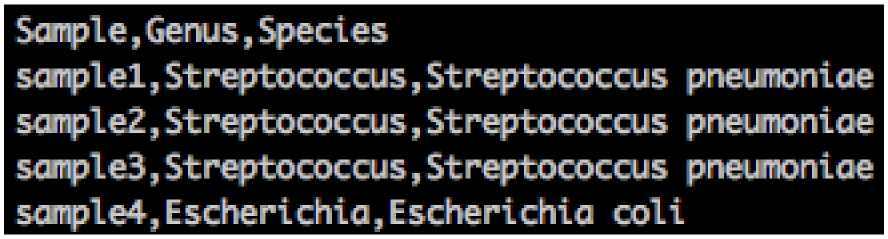
It is always a good idea to check that your samples are the species you expect them to be. You can use tools such as `Bactinspector` or `Kraken` (https://www.ebi.ac.uk/research/enright/software/kraken) for this. `Roary` has a qc option that will run `Kraken` for you and generate a report listing the top species of all the samples.

We will not run this here but an eample report may look something like this:



```
Sample,Genus,Species
sample1,Streptococcus,Streptococcus pneumoniae
sample2,Streptococcus,Streptococcus pneumoniae
sample3,Streptococcus,Streptococcus pneumoniae
```

As we expected, these three samples are all of the same species. Let's assume that we had an additional forth sample and we run `Roary` with the qc option, we get the following output:



```
Sample,Genus,Species
sample1,Streptococcus,Streptococcus pneumoniae
sample2,Streptococcus,Streptococcus pneumoniae
sample3,Streptococcus,Streptococcus pneumoniae
sample4,Escherichia,Escherichia coli
```

This tells us that the most prevalent species in sample 4 is in fact *Escherichia coli* so we will exclude this sample from our analysis before we carry on.

4.2 Coverage

To get reasonable quality assemblies, you need a genome coverage of at least 30x. Remember to get a quick estimate of your coverage, you can divide the number of bases in your sequence data with the number of bases in the reference genome of the species. For the samples used in this tutorial, the coverage is listed below. The genome of *S. pneumoniae* is approximately 2,200,000 bases.

Sample	No. of Bases	Coverage
sample1	262705400	120x
sample2	218026200	99x
sample3	173524000	79x

4.3 Assembly size

The size of the assemblies can also provide a useful hint. If one of the assemblies is much smaller or bigger than the others there is a chance that this is not of the same species as the rest.

4.4 Fragmented assemblies

If the assemblies are very fragmented (thousands of contigs), the genes may be too fragmented for inferring the pangenome.

These are just some of the most basic things that you can do to make sure your data looks ok. There is much more that can be done but we won't go into any further detail in this tutorial.

To generate some basic metrics about genome assemblies you can use the `assembly-stats` tools.

[ ]:
```
assembly-stats assemblies/sample1.fasta
```

4.5 Check your understanding

Q5: Why is it important to QC your data?

Q6: You're not getting any core genes when you run Roary. What could be the reason?

Q7: What is the size of the assembly for sample1?

Q8: How many contigs are in the assembly of sample1?

Now you should be ready to run Roary to construct a pangenome, so go to the next section, Constructing a Pangenome using Roary.

5 Constructing a Pangenome with Roary

At this stage you should have three GFF files generated by `Prokka`, each in its own directory. Provided your QC looked ok, you are now ready to run `Roary` to generate the pangenome.

We are going to run `Roary` twice, first with the default settings, and then using `MAFFT` to generate a core gene alignment. For both of these runs we will want all the annotation files in the same directory, so lets take a copy of them to our current directory:

[ ]:
```
cp annotated_sample*/*.gff .
```

5.1 Run Roary with default settings

To run `Roary` with the default settings all you need to do is run `roary *.gff` and it will create a pangenome using all GFF files in the current directory.Try the following command:

[ ]:
```
roary -f output_no_alignment *.gff
```

We want to run Roary twice with different settings, so in order to keep track of our output files from each run The **-f** option is used to specify the name of the ouptut directory where `Roary` should put the results. This will run for a few minutes.

We will have a closer look at the results in the next section, so for now let us just see that there are some output files in the directroy we asked `Roary` to create:

[ ]:
```
ls -l output_no_alignment
```

5.2 Run Roary with MAFFT

But we want to generate a multi-FASTA alignment of the core genes so that we can draw a phylogenetic tree. So try:

[ ]:
```
roary -f output_with_alignment -e --mafft -p 2 *.gff
```

Here we have run `Roary` again, but this time with some more options.

Option	Description
-e	Create a multi-FASTA alignment of the core genes
–mafft	Use with -e to use MAFFT instead of PRANK
-p	Number of threads to use

By default, 'Roary' will use `PRANK` when the **-e** option is speified. It is accurate but slow. `MAFFT` is less accurate but very fast so we are going to use this instead by specifying the **–mafft** option. To further speed things up, we are going to use 2 threads (the **-p** option). For all usage options, you can have a look at the Roary website (https://sanger-pathogens.github.io/Roary/).

This will take a bit longer to run than the previous command, maybe 5 or 10 minutes, perhaps answer the questions at the end of this section while waiting for this to complete.

Once finished you should have a directory called `output_with_alignment` containing the output files, this time including a core_gene_alignment.aln file. Just quickly check that this is the case.

[ ]:
```
ls -l output_with_alignment
```

5.3 Check your understanding

Q9: Why do we want to run Roary with MAFFT?
a) Because it's quicker than to run Roary without the -e option
b) To get more accurate results
c) To generate a core gene alignment

Q10: Why do we use the -p otion?
a) We have to when we use MAFFT
b) To speed up the run
c) To get a nice tree

Now go to the next section: Exploring the results.

6 Exploring the results

6.1 Output files

Let's have a look at the results. We will focus on the output from the second run as this will be the same as the first run but will also include the core gene alignment produced by `MAFFT`. We will start by looking at the most important output files and after this we will look at how you can query your pangenome and draw a basic tree form the core gene alignment.

[ ]:
```
ls output_with_alignment
```

6.1.1 summary_statistics.txt

The summary_statistics.txt file contains a summary of the number of genes in the pan, core and accessory genomes. It provides an overview of the genes and how frequently they occur in the input isolates. Usually, you can expect the total number of genes in this file to be about 1,000 genes per million bases of your species reference genome. In this case, the genomes are around 2 million bases, so we would expect a total number of genes to be in the order of 2,000. Let's have a look and see if this is the case.

[ ]:
```
cat output_with_alignment/summary_statistics.txt
```

As you can see, we have around 2,500 genes which seems about right. If you get a lot fewer or many more genes than expected this could be an indication of an issue with your input data, for example contamination.

6.1.2 gene_presence_absence

The gene_presence_absence files lists each gene and which samples it is present in. The .csv file can be opened in Excel.

Let's have a look at the first ten lines of the file:

[ ]:
```
head output_with_alignment/gene_presence_absence.csv
```

The columns are tab separated and contains the following information:

1. The gene name (the most frequently occurring gene name from the sequences in the cluster)
2. A non unique gene name
3. Functional annotation (the most frequently occurring functional annotation from the cluster)
4. Number of isolates in the cluster
5. Number of sequences in the cluster
6. Average number of sequences per isolate (normally 1)
7. Genome fragment
8. Order within fragment
9. Accessory fragment
10. Accessory order within fragment
11. Comments on the quality of the cluster
12. Minimum sequence length in nucleotides of the cluster
13. Maximum sequence length in nucleotides of the cluster

14. Average sequence length in nucleotides of the cluster
15. Presence and absence of genes in each sample, with the corresponding source Gene ID

The .Rtab file contains a tab delimited binary matrix with the precence and abscence of each gene in each sample. This makes it easy to load into R using the read.table function, giving you access to a number of useful tools. The first row is the header containing the name of each sample, and the first column contains the gene name. In the matrix, 1 indicates the gene is present in the sample and 0 indicates it is absent.

6.1.3 pan_genome_reference.fa

This fasta file contains a single nucleotide sequence from each of the clusters in the pan genome. The name of each sequence is the source sequence ID followed by the cluster it came from.

6.1.4 .Rtab

Roary comes packaged with a script called create_pan_genome_plots.R. It requires R and the R-package ggplot2, and can be used to generate graphs from the .Rtab files, showing how the pan genome varies as genomes are added.

6.1.5 accessory_binary_genes.fa.newick

This is a tree in newick format, created using the binary presence and absence of accessory genes. It can for example be viewed in FigTree. The tree is only a tough estmate, generated to provide a basic overview of the data. To generate a more accurate tree, we will use the core gene alignment a bit further on.

6.1.6 core_gene_alignment.aln

This is the multi-FASTA alignment of core genes that we created in the second run, using MAFFT. We will soon use this as input to build a phylogenetic tree.

6.1.7 clustered_proteins

In this file each line lists the sequences in a cluster. We will use this later on in the tutorial to query the pan genome.

For more information about the different output files, including some that we haven't mentioned here, please feel free to have a look at the Roary web page.

6.2 Query the pan genome

Roary comes with a script called query_pan_genome that can be used to examine the gene differences between groups of isolates. To have a look at the usage options for this script, you can do:

```
query_pan_genome -h
```

This will show you the following usage options:

```
Usage: query_pan_genome [options] *.gff
Perform set operations on the pan genome to see the gene differences
```

```
between groups of isolates.

Options: -g STR    groups filename [clustered_proteins]
         -a STR    action (union/intersection/complement/gene_multifasta/
                     difference) [union]
         -c FLOAT  percentage of isolates a gene must be in to be core [99]
         -o STR    output filename [pan_genome_results]
         -n STR    comma separated list of gene names for use with
                     gene_multifasta action
         -i STR    comma separated list of filenames, comparison set one
         -t STR    comma separated list of filenames, comparison set two
         -v        verbose output to STDOUT
         -h        this help message

Examples:
Union of genes found in isolates
         query_pan_genome -a union *.gff

Intersection of genes found in isolates (core genes)
         query_pan_genome -a intersection *.gff

Complement of genes found in isolates (accessory genes)
         query_pan_genome -a complement *.gff

Extract the sequence of each gene listed and create multi-FASTA files
         query_pan_genome -a gene_multifasta -n gryA,mecA,abc *.gff

Gene differences between sets of isolates
         query_pan_genome -a difference --input_set_one 1.gff,2.gff --
input_set_two 3.gff,4.gff,5.gff

For further info see: http://sanger-pathogens.github.io/Roary/
```

As you can see, this also shows us some example uses. Try the first example:

[ ]:
```
query_pan_genome -a union \
    -g output_with_alignment/clustered_proteins *.gff
```

This will give us a file called pan_genome_results that contains a list of all genes in all samples, i.e. the pangenome. Have a look at the first ten lines of the newly generated file:

[ ]:
```
head pan_genome_results
```

The list contains the names of the gene clusters (this is usually the most frequently occurring gene name from the sequences in the cluster or, if there is no gene name, a generic unique name group_XXX). For each cluster, there is a tab separated list of each sample specific gene belonging in that cluster.

In a similar way, you can use query_pan_genome to get a list of the core genes:

```
query_pan_genome -a intersection \
    -g output_with_alignment/clustered_proteins *.gff
```

and a list of the accessory genes:

```
query_pan_genome -a complement \
    -g output_with_alignment/clustered_proteins *.gff
```

You can also use query_pan_genome to extract the protein sequences for genes you are particularly interested in.

[ ]:
```
query_pan_genome -a gene_multifasta \
    -g output_with_alignment/clustered_proteins \
    -n patB,mnmG,hsdM *.gff
```

The **-a** option species the action (create a multi-FASTA file of protein sequences) and the **-n** option specifies a comma separated list of the cluster names (patB,mnmG,hsdM).

You should have three new files, one for each gene you specified.

[ ]:
```
ls *.fa
```

[ ]:
```
Have a look at the `pan_genome_results_patB.fa` file:
```

[ ]:
```
cat pan_genome_results_patB.fa
```

This multi-FASTA file contains the three protein sequences in the specified cluster (patB).

There is yet more functionality of query_pan_genome, but we won't go into that here. For further information, please feel free to visit the Roary web page (https://sanger-pathogens.github.io/Roary/).

6.3 Draw a tree form the core gene alignment

The tree created by `Roary` (accessory_binary_genes.fa.newick) is just a quick tree to provide a rough insight into the data. To create a more accurate tree you can use the core gene alignment as input to a tree building software of your choice. RAxML (https://sco.h-its.org/exelixis/web/software/raxml/index.html) is very accurate, however it is also fairly slow so in this tutorial we are going to use FastTree (http://www.microbesonline.org/fasttree/).

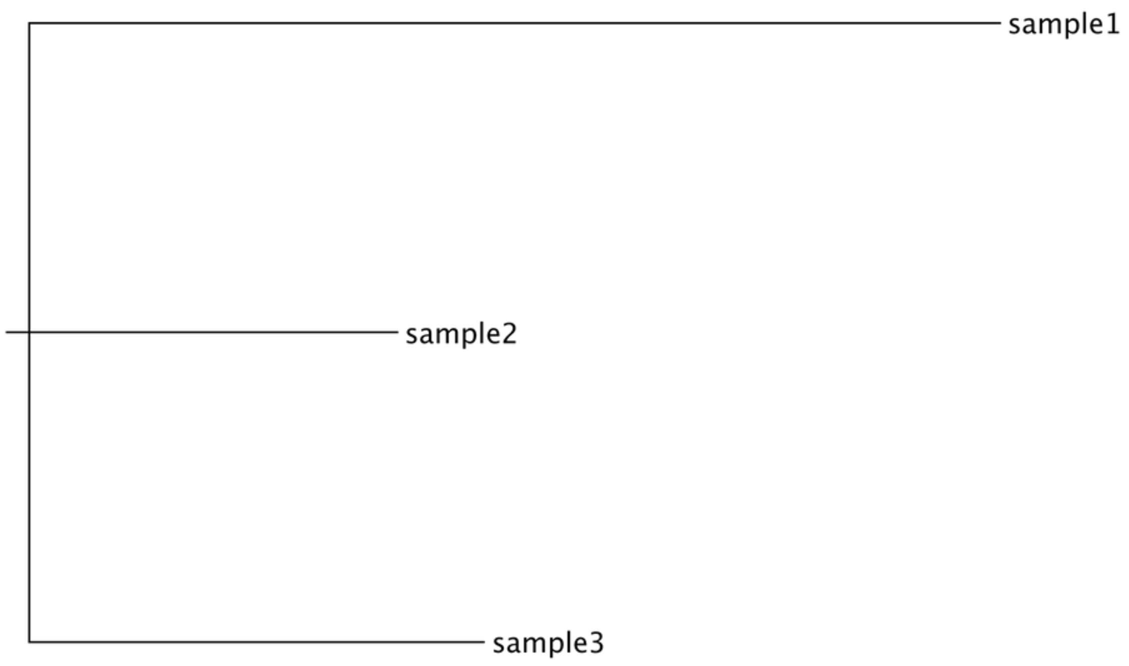
To create a tree in Newick format ([https://evolution.genetics.washington.edu/phylip/newicktree.html])(https://evolution.genetics.washington.edu/phylip/newicktree.html) from a nucleotide alignment using a generalized time-reversible model (the -gtr option), do:

[ ]:
```
fasttree -nt -gtr output_with_alignment/core_gene_alignment.aln \
    > tree.newick
```

The tree in this case will look like:

```
(sample1:0.006228253,sample2:0.002364375,sample3:0.002920483);
```

We can view this in iTOl or FigTree, which will look something like:



In the event that you did not run this step, a copy of tree.newick has been placed in the `tree` directory for the next section of this tutorial.

6.4 Check your understanding

Q11: Approximately how many genes would you expect to see in the summary_statistics.txt file if you are working with a species with a genome size of 5,000,000 bases?
a) 500
b) 5000
c) 50,000

Q12: What does the accessory_binary_genes.fa.newick file provide?
a) A pylogenetic tree ready for publishing
b) Nothing, it is useless
c) A quick insight to the data

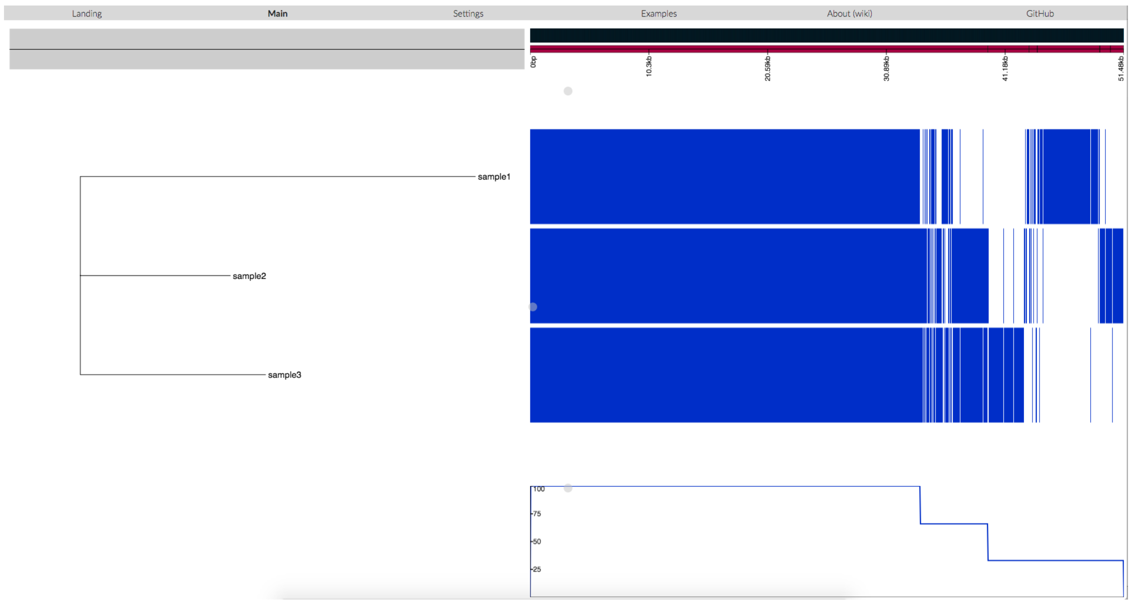
Q13: For query_pan_genome, what option should you use to get the accessory genome?
a) union
b) intersection
c) complement

Now that you are familiar with the output files produced by Roary, let's make use of them by visualising the results using Phandango.

7 Visualising the results with phandango

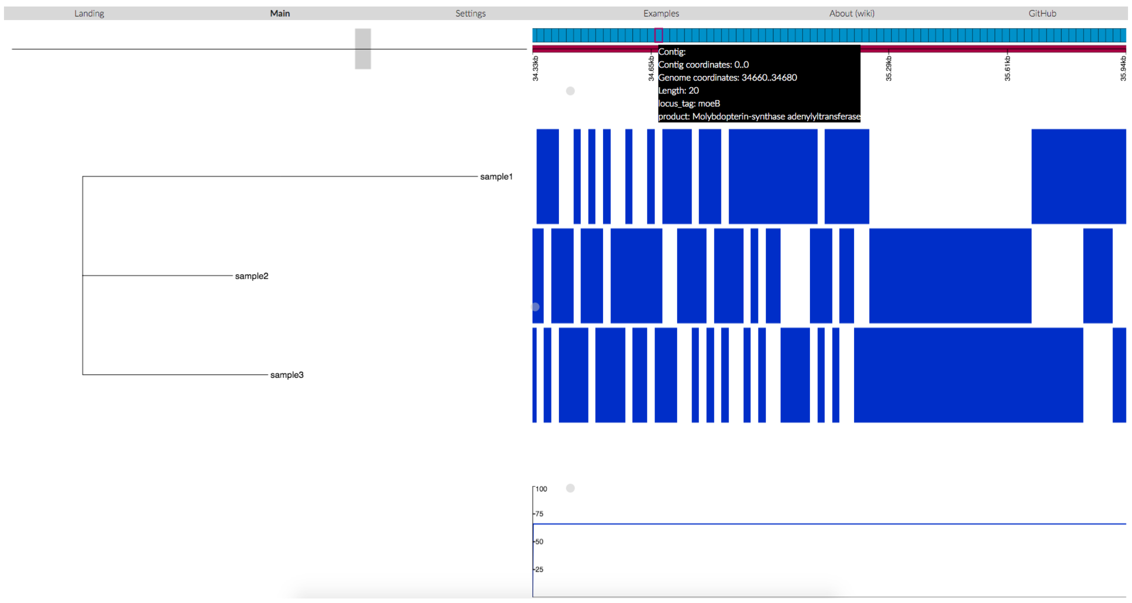
Phandango (http://phandango.net/) is a web based tool for visualising genomic analysis results such as phylogenetic trees and the output of `Roary` and other tools. Using phandango is straightforward, you just drag your files onto the browser and drop them and `Phandango` will display them for you automatically. For more in depth information please feel free to have a look around the phandango wiki page (https://github.com/jameshadfield/phandango/wiki).

In this section of the tutorial we are going to use `Phandango` to look at the pylogenetic tree we generated using `FastTree`, and the gene_presence_absence.csv file we obtained from `Roary`. Simply dragging the two files and droping them on the phandango front page will give you the following view:



The image shows our tree compared to a matrix with the presence and absence of genes in the pan genome. The graph on the bottom right provides a summary of the matrix above, indicating the percentage of isolates carrying a gene at each position.

You can zoom in on a particular area you are interested in simply by scrolling, and to see the annotation for a particular gene you can place your pointer over the corresponding bar at the top of the page, like in the figure below.



Contig:
Contig coordinates: 0..0
Genome coordinates: 34660..34680
Length: 20
locus_tag: moeB
product: Molybdopterin-synthase adenylyltransferase

In this case we are looking at a gene called *moeB*. We can see that the gene is present in sample2 and sample3, but not in sample1, and that the product is an Molybdopterin-synthase adenylyltransferase.

Go ahead and experiment with `Phandango`. You can alter the layout in *settings* in the navigation bar at the top of the page, or by clicking and dragging the grey circles on the page. To download the current view in svg format, simply press *p*.

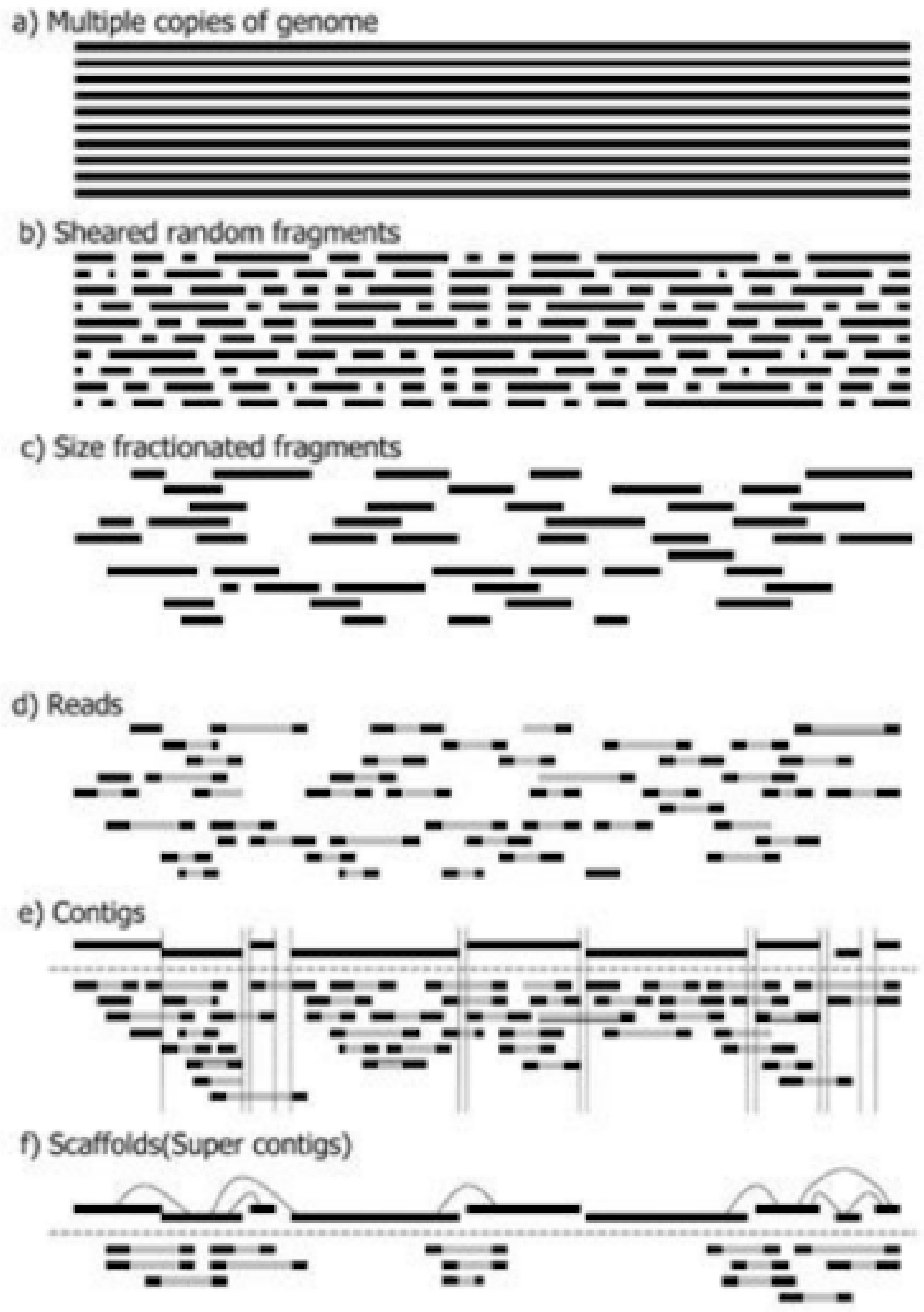
7.1 Check your understanding

In Phandango, zoom in on the gene cluster at position 25080.
Q14: What is the name of this gene cluster?
Q15: Is this a core gene?

This tutorial has shown you how to construct a pangenome for a set of genome assemblies. The final section of this turorial will discuss how to generate genome assemblies from short read sequence data. OK, let's generate some genome assemblies!

8 Creating genome assemblies

Genome assembly is the process of taking a large number of fragments of DNA and putting them back together to create a representation of the original DNA sequence from which they originated.


a) Multiple copies of genome
b) Sheared random fragments
c) Size fractionated fragments
d) Reads
e) Contigs
f) Scaffolds(Super contigs)

Many genomes contain large numbers of repeat sequences. Often these repeats are thousands of nucleotides long, and some occur in many different locations in the genome. This makes genome assembly a very difficult computational problem to solve. However, there are many genome assembly tools that exist that can produce long contiguous sequences (contigs) from sequencing reads. The assembly tool that you use will be determined by different factors, largely this will be the length of the sequencing reads and the sequencing technology used to produce the reads.

First, check you are in the correct directory.

```
[ ]: pwd
```

It should display something like:

```
/home/manager/course_data/pangenome/data
```

8.1 Assemble a genome with SPAdes

Now we are going to use the `SPAdes` assembly tool to generate an assembly. As always it is a good idea to get a look at the options that a program accepts using the -h option. SPAdes is actually written in python and the base script name is "spades.py".

```
[ ]: spades.py -h
```

To assemble the data for sample ERR657310 using `SPAdes` and default parameters use:

```
[ ]: spades.py --pe1-1 fastq/ERR657310_1.fastq.gz --pe1-2 fastq/ERR657310_2.fastq.gz␣
     ↪--only-assembler --careful -o ERR657310 -k 33,43,53,63 --threads 1
```

This may take some time to run so please be patient.

`SPAdes` (St. Petersburg genome assembler) is a genome assembly algorithm which was designed for single cell and multi-cell bacterial data sets. Therefore, it might not be suitable for large genomes projects. `SPAdes` works with Ion Torrent, PacBio, Oxford Nanopore, and Illumina paired-end, mate-pairs and single reads.

Here we are using it to assemble Illumina paired end data. Take a look at the parameters passed to SPAdes, what do the different parameters mean?

When the assembly is complete look at the files that were produced by SPAdes:

```
[ ]: ls ERR657310
```

As you can see from listing the contents of the output `sample3` directory, several new files have been generated. There are two files that I consider to be the most important. * `contigs.fasta` - this is the actual result of all the different contigs that were created. For circular chromosomes (such as plasmids) the goal would be that there is a single contig meaning that all of the reads were able to close the circle. * `spades.log` - this has the information about the completed run that you can use to compare different samples or conditions in the event that you are interested trying to optimize the command options, as would likely be the case if you were trying to assemble the best reference possible.

8.2 Assembly metrics

To generate metrics to assess the quality of the assembly you can use program called `assembly-stats`. It displays the number of contigs, the mean contig size and a lot of other useful metrics about the assembly. These numbers can be used to assess the quality of your assembly.

[ ]:
```
assembly-stats ERR657310/contigs.fasta
```

Now look at the output of `assembly-stats` and answer the questions below

8.3 Check your understanding

Q16: What is the size of the assembly?

Q17: How many contigs did it assemble into?

Q18: What is the largest contig?

Q19: What is the N50?

Q20: Is this a good assembly?

Another good software tool to assess the quality of your assemblies is `Quast` (http://bioinf.spbau.ru/quast). However, we do not have time here to cover the use of Quast for assembly QC.

8.4 Genome assembly pipelines

Unfortunately, to generate high quality genome assemblies it is usually not as straightforward as just running an assembly tool like `SPAdes`. There are several pre-processing and post-processing steps than need to be carried out in order to improve the chances of creating a good quality assembly. These steps can include:

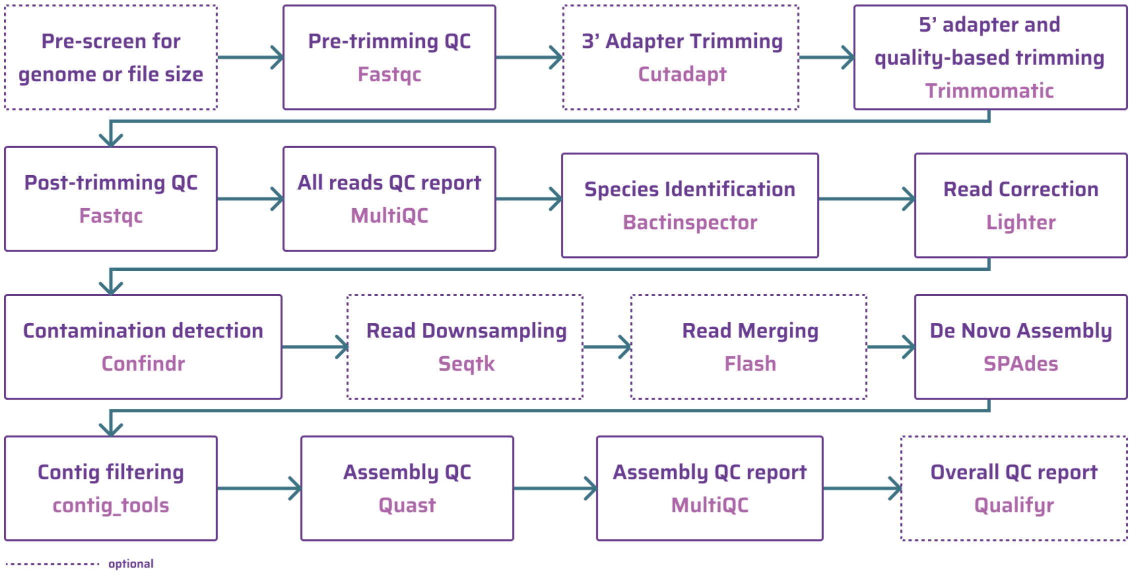
- Trimming reads to remove low quality bases
- Trimming reads to remove sequence adapter
- Correcting sequencing errors in the reads
- Downsampling the data to allow the assembly to run within reasonable time and memory
- Mapping reads back to the assembly to correct errors in the assembly
- Filtering small/low quality contigs from the assembly

Fortunately, there are some pipelines available that incorporate all these steps and can be used to assemble genomes in batch for multiple samples. Two pipelines are:

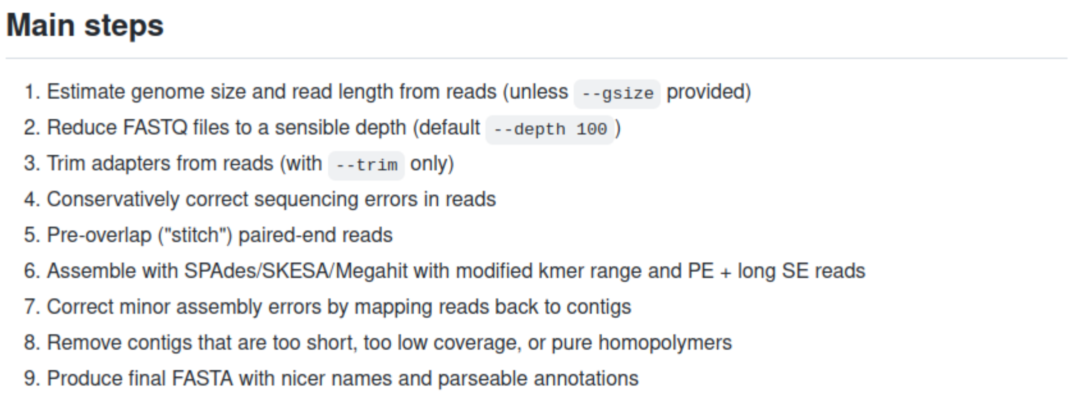
- Pathogenwatch/GHRU assembly (https://gitlab.com/cgps/ghru/pipelines/dsl2/pipelines/assembly)
- Shovill (https://github.com/tseemann/shovill)

The steps involved in each of these pipelines are outlined below.

8.4.1 Pathogenwatch/GHRU



8.4.2 Shovill



Main steps

1. Estimate genome size and read length from reads (unless `--gsize` provided)
2. Reduce FASTQ files to a sensible depth (default `--depth 100`)
3. Trim adapters from reads (with `--trim` only)
4. Conservatively correct sequencing errors in reads
5. Pre-overlap ("stitch") paired-end reads
6. Assemble with SPAdes/SKESA/Megahit with modified kmer range and PE + long SE reads
7. Correct minor assembly errors by mapping reads back to contigs
8. Remove contigs that are too short, too low coverage, or pure homopolymers
9. Produce final FASTA with nicer names and parseable annotations

We do not have time to cover the use of these pipelines in this tutorial. But we highly recommend using them if you intend to assembly your own data.

Congratulations! You have reached the end of the tutorial.