1 PathFind scripts

1.1 Introduction

A series of scripts were developed so that users can access imported sequence data and the results of the analysis pipelines. These are referred to as the **PathFind** or **pf** scripts. The source code for the Perl module which is used to run the pf scripts can be found on the sanger-pathogens Git repository as Bio-Path-Find.

1.2 Learning outcomes

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

- Find the pipeline status for your lane(s) using the pf scripts
- Find the data for your lane(s) using the pf scripts
- Find the quality control (QC) results for your lane(s) using the pf scripts
- Find the analysis pipeline results for your lane(s) using the pf scripts
- Find a reference using the pf scripts

1.3 Tutorial sections

1.4 Authors

This tutorial was created by Victoria Offord.

1.5 Running the commands from this tutorial

You can run the commands in this tutorial either directly from the Jupyter notebook (if using Jupyter), or by typing the commands in your terminal window.

1.5.1 Running commands on Jupyter

If you are using Jupyter, command cells (like the one below) can be run by selecting the cell and clicking *Cell -> Run* from the menu above or using *Ctrl Enter* to run the command. Let's give this a try by printing our working directory using the `pwd` command and listing the files within it. Run the commands in the two cells below.

```
pwd
```

```
ls -l
```

1.5.2 Running commands in the terminal

You can also follow this tutorial by typing all the commands you see into a terminal window. This 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, select the cell below with the mouse and then either press control and enter or choose *Cell -> Run* in the menu at the top of the page.

```
echo cd $PWD
```

Open a new terminal on your computer and type the command that was output by the previous cell followed by the enter key. The command will look similar to this:

```
cd /home/manager/pathogen-informatics-training/Notebooks/PathFind/
```

Now you can follow the instructions in the tutorial from here.

1.6 Prerequisites

This tutorial assumes that you have Bio-Path-Find installed on your computer.

To be able to access the tutorial dataset, we need to add a few lines to our configuration file which tell it where to write the log file and the location of the tutorial data. We also need to add the paths for our tutorial reference sequences to `data/refs.index`.

Please run the following commands.

```
./setup_config.sh
```

```
./setup_refs.sh
```

```
(no output)
```

You will be asked to run the following command in each section of the tutorial. This tells the system where to find our configuration file. Please run the command now.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

To check that you have installed Bio-Path-Find correctly and that you can access the tutorial, please run the following command:

```
pf -h
```

This should return the help message for the pf scripts.

1.7 Let's get started!

To get started with the tutorial, head to the first section: Introduction.

2 Introduction

2.1 Automated analysis pipelines

Pathogen Informatics maintain a series of databases which track the progress of pathogen studies and samples. These samples may have been through the Sanger sequencing pipeline (internal) or imported from other sources (external).

Once the sample data is in the Pathogen Informatics databases, it is then available to the automated analysis pipelines. Pathogen Informatics maintain the following automated analysis pipelines:

- Quality control (QC)
- Mapping
- SNP calling
- Bacterial, Eukaryote and Pacbio assembly
- Annotation
- RNA-Seq expression

For more information on study registration, external data tracking and the automated analysis pipelines, please see the Pathogen Informatics wiki.

2.2 Accessing pathogen data and analysis results

A series of scripts were developed so that users can access imported sequence data and the results of the analysis pipelines. These are referred to as the **pathfind** or **pf** scripts.

Command	Description
pf status	used to find the pipeline progress for a given study, sample or lane
pf data	used to find the FASTQ or PacBio files for a given study, sample or lane
pf info	used to match sample internal ids and and supplier ids for a given study, sample or lane
pf accession	used to obtain accession numbers for a given study, sample or lane
pf supplementary	used to get supplementary information about a given study, sample or lane
pf qc	used to find the Kraken results for a given study, sample or lane
pf map	used to find the location of BAM files produced by the mapping pipeline
pf snp	used to find the location of VCF files produced by the SNP calling pipeline

Command	Description
pf assembly	used to find the location of the contig FASTA files produced by the assembly pipeline
pf annotation	used to find the location of the GFF files produced by the annotation pipeline
pf rnaseq	used to find the location of expression counts produced by the RNA-Seq analysis pipeline
pf ref	used to find the location of a reference on pathogen disk

The pf scripts return information or locations for each lane that is found. To run the individual scripts the command structure we use is `pf` followed by the command you want to use i.e. `data` or `status` and then the options for that command.

```
pf <command> [options]
```

For example, to use `pf data` it would be:

```
pf data [options]
```

To specify which lanes we want to retrieve, we use the **`--type`** (**`-t`** or **`--type`**) and **`--id`** (**`-i`** or **`--id`**) options. These are **required** options for all of the `pf` commands except for `pf ref`.

There are four commonly used ID types (**`-t`**) you can use to search for information:

- **study**
  retrieve all lanes associated with a study using a study ID
- **lane**
  retrieve a single lane using a lane name
- **sample**
  retrieve a single sample using a sample name
- **file**
  retrieve all lanes which are listed in the file using the filename as the identifier

For `pf data` this would look like:

```
pf data -i <id> -t <ID type>
```

2.2.1 Getting help

You can look at an overview of all the pf commands using:

```
pf -h
```

Or, you can look at the usage and available options for a particular command using:

```
pf <command> -h
```

2.3 Exercise 1

This exercise uses `pf data` to walk you through using the four most commonly used ID types to search for information.

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

2.3.1 Retrieving all lanes associated with a study

To retrieve all of the lanes which are associated with a study we will set the type (**-t**) to **study** and the id (**-i**) to **664**.

Let's try searching for lanes associated with study 664.

```
pf data -t study -i 664
```

We can also search for a study using its name. In Sequencescape we can see that the name of study 644 is *"Streptococcus pneumoniae global lineages"*.

Let's try searching using the study name.

```
pf data -t study -i "Streptococcus pneumoniae global lineages"
```

Finally, we can count the number of lanes that were returned using `wc -l`.

```
pf data -t study -i 664 | wc -l
```

2.3.2 Retrieving a single lane

When we don't want all the lanes from a study, we can search for individual lanes. To do this we need to set the type (**-t**) to **lane** and give the lane name as our identifier (**-i**).

Let's try searching for a lane using the lane name.

```
pf data -t lane -i 5477_6#1
```

2.3.3 Retrieving a run

If there are multiple lanes you want to retrieve from a run, we can search for all lanes associated with that run. To do this we need to set the type (**-t**) to **lane** and give the run as our identifier (**-i**).

Let's try searching for lanes associated with run 5477_6.

```
pf data -t lane -i 5477_6
```

2.3.4 Retrieving a single sample

Perhaps you don't have the lane name but you do have the sample name. To seach using a sample name we need to set the type (**-t**) to **sample** and give the sample name as the id (**-i**).

Let's try searching for a sample using the sample name.

```
pf data -t sample -i Tw01_0055
```

2.3.5 Retrieving multiple lanes using a file

Last, but not least, we can retrieve information for a list of lanes which are stored in a file. First, let's take a look at our file of lanes.

```
cat data/lanes.txt
```

Here you can see we have one lane per line in our file. To use this file, we need to set the type (**-t**) to **file** and give the file name as the id (**-i**).

Let's try searching for information on the lanes in "data/lanes.txt".

```
pf data -t file -i data/lanes.txt
```

2.4 Questions

Q1: How many lanes are associated with study 607?
Hint: you can use `wc -l` to count the number of lines (lanes) returned by `pf data`

```
# Enter your answer here
```

Q2: How many lanes are returned if you search using the file "data/lanes_to_search.txt"?
Hint: you can use `wc -l` to count the number of lines (lanes) returned by `pf data`

```
# Enter your answer here
```

2.5 What's next?

For a quick recap of what the tutorial covers and the software you will need, head back to the tutorial overview.

Otherwise, let's get started with looking at finding your data.

3 Finding your data

3.1 Introduction

To search for the location(s) of data stored in the pathogen databases, we can use `pf data`. In the previous section, we looked at two options which are used by most of the pf scripts, **type** (**`-t`**) and **id** (**`-i`**).

In this section of the tutorial we will be looking at several other functions which `pf data` can perform that may be useful when finding, sharing or using your sequencing data.

By default, `pf data` will return a directory. It not only contains the imported sequence data, but also the results of any of the analysis pipelines which have been run on that data.

In this section of the tutorial we will cover:

- the `pf data` command format
- using `pf data` to find the top level directory where sequence data and analysis pipeline results are stored
- using `pf data` to find sequence data files
- using `pf data` to symlink files and directories
- using `pf data` to compress files and directories
- using `pf data` to generate sequencing data statistics

3.1.1 Filetypes

However, you might not want to know the top level directory location. You might want to know where the sequence data files are and what they are called so that you can use them in a down-stream analysis. To do this, we ask `pf data` to find the sequence files using the **filetype** (**–filetype** or **-f**).

3.1.2 Symlinking

Pathogen Informatics asks users not to copy sequence data or results that are already in the pathogen databases. This is because copying data uses up precious disk space.

Instead we ask users to **symlink** the data. Symlinks contain no data, simply referencing the location of the original file or directory. To most commands, the symlink looks like the original file, but the operations the command performs (e.g. reading from the file) are directed to the original file which the symlink is pointed to.

You can symlink a file or directory that's returned by a `pf data` search by using the **`--symlink`** or **`-l`** option.

3.1.3 Archiving or compressing data

You may want to transfer or share some of your sequencing data. The simplest way to do this is to **archive** or **compress** the data you want to transfer. To compress data returned by `pf data` you can use the **--archive** or **-a** option. This will compress the returned data and return a file with the extension ".tar.gz" that is much smaller and easier to share or transport.

3.1.4 Getting general information and statistics

For some of the `pf` scripts, you can also get an overview of the data returned by `pf data` using the **--stats** or **-s** option. This will write a spreadsheet which contains statistics and general information.

These include, but are not limited to:

- **general information**
  study ID, sample name, lane name…
- **sequencing information**
  number of cycles, number of reads, number of bases…
- **quality control (QC) results**
  reference used, percentage mapped, percentage paired, depth of coverage…
- **pipeline status**
  QC, mapping, SNP calling, assembly, annotation, RNA-Seq…

3.2 Exercise 2

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

You can see the available options for `pf data` using the **--help** or **-h** option.

Let's take a look at the usage information for `pf data`.

```
pf data -h
```

Here we can see that basic `pf data` command uses just the **type** (**--type** or **-t**) and **id** (**--id** or **-i**) options.

```
pf data --id <id> --type <ID type> [options]
```

Let's search for the location of data associated with lane 5477_6#1.

```
pf data -t lane -i 5477_6#1
```

The disk location `pf data` returned is the **top level** directory where all of data and results associated with lane 5477_6#1 are stored.

3.2.1 Filetypes

We may want to find the sequence data files which were imported so that we can use them for a subsequent analysis.

Let's find the FASTQ files which were imported for lane 5477_6#1.

```
pf data -t lane -i 5477_6#1 -f fastq
```

As this is Illumina paired end data, there are two gzipped (.gz) FASTQ-formatted sequence data files returned which correspond to the left (_1) and right (_2) reads.

3.2.2 Symlinking

We don't want to copy these files to where we're running the analysis because this uses up disk space unnecessarily. Instead, we'll symlink them.

First, let's try symlinking our two FASTA files from lane 5477_6#1.

```
pf data -t lane -i 5477_6#1 -f fastq -l
```

This should return a message like "Creating links in 'pathfind_5477_6_1' " which tells you where your files have been symlinked to. Here we can see that a new directory has been created with the prefix "pathfind_" and our lane name "5477_6_1". You'll also notice that the "#" in our lane name has been replcated by an underscore ("_").

Now, let's look in the new directory with `ls`.

```
ls pathfind_5477_6_1
```

There we see our two files "5477_6#1_1.fastq.gz" and "5477_6#1_1.fastq.gz".

But, if we take a closer look using `ls -l` we can see that those files are symlinks to our tutorial data files.

```
ls -l pathfind_5477_6_1
```

Now, let's try symlinking to a new directory called "my_lanes".

```
pf data -t lane -i 5477_6#1 -f fastq -l my_lanes
```

We can now see that a new directory called "my_lanes" has been created.

```
ls
```

And inside the "my lanes" directory are our two symlinked files.

```
ls -l my_lanes
```

So, we've been symlinking our FASTQ files. But, what if we want to symlink all of the data and results associated with our lane.

Instead of symlinking just our sequence data, let's symlink all of the data and results for lane 5477_6#1 to a new directory called "my_lane_data".

```
pf data -t lane -i 5477_6#1 -l my_lane_data
```

Looking inside "my_lane_data" we see a directory which has the same name as our lane, 5477_6#1. This directory is symlinked to the tutorial data directory for this lane.

```
ls -l my_lane_data
```

Finally, let's try symlinking the data and results for all lanes associated with a study.

```
pf data -t study -i 664 -l my_study_lanes
```

Here we see 11 symlinked directories which have the names of the 11 lanes associated with study 664.

```
ls -l my_study_lanes
```

3.2.3 Archiving data

Sometimes, you may want to transfer or share your data. The simplest way to do this is to archive or compress the sequence data.

Let's archive the data for lane 5477_6#1.

```
pf data -t lane -i 5477_6#1 -a
```

Here we see "pathfind_5477_6_1.tar.gz" has been created.

```
ls
```

We can uncompress "pathfind_5477_6_1.tar.gz" using `tar`.

```
tar xf pathfind_5477_6_1.tar.gz
```

This gives us a directory which shares the name of the lane we were looking for (with '#' replaced with an '_'). Inside that directory are our two sequence data files "5477_6#1_1.fastq.gz" and "5477_6#1_2.fastq.gz" as well as "stats.csv" which contains some general information and statistics.

3.2.4 Getting general information and statistics

We can get some general information and statistics about our sequence data using the `-s` or `--stats` option with `pf data`.

Let's try getting some statistics for lane 5477_6#1.

```
pf data -t lane -i 5477_6#1 -s
```

You can see this has generated a new file called "5477_6_1.pathfind_stats.csv".

```
ls
```

We can take a quick look at the contents of this file using `cat`.

```
cat 5477_6_1.pathfind_stats.csv
```

Now, let's try getting some statistics for all lanes in our file "lanes.txt" and calling the output file "my_lane_stats.csv".

```
pf data -t file -i data/lanes.txt -s my_lane_stats.csv
```

You should get a message which says your statistics have been written to "my_lane_stats.csv". We can take a look at this file. Perhaps just getting the first few columns using `awk`.

Note: we use '-F' with `awk` to tell it that the data we're parsing is comma-separated.

```
awk -F',' '{print $1"\t"$2"\t"$3}' my_lane_stats.csv
```

Here we can see that there is one row per lane in the statistics file (see the "Lane Name" column).

3.3 Questions

Q1: What is the location of the top level directory for data and results associated with lane 10018_1#1?

```
# Enter your answer here
```

Q2: What is the location of the FASTQ file(s) associated with lane 10018_1#1?

```
# Enter your answer here
```

Q3: Symlink the FASTQ files from study 607 into a directory called "study_607_links". How many FASTQ files were symlinked to "study_607_links?
Hint: you can use wc -l to count the number of files in the directory

```
# Enter your answer here
```

```
# Enter your answer here
```

Q4: What reference was used to map lane 10018_1#1 during QC and what percentage of the reads were mapped to the reference?
Hint: you'll need to get some statistics

```
# Enter your answer here
```

```
# Enter your answer here
```

3.4 What's next?

For a quick recap of what the pf scripts are, head back to the introduction.

Otherwise, let's move on to sample information and accessions.

4 Sample information and accessions

4.1 Introduction

Once your samples have been sequenced or imported, it can be useful to match up the internal lane identifiers with the sample and supplier identifiers. We can look at the relationship between lane and sample using `pf info` which will return values for:

- Lane name
- Sample name
- Supplier name
- Public name
- Strain

Alternatively, you might want to know the EBI sample and submission numbers for a particular lane or sample. To get this, you can use `pf accession` which will return:

- Sample name
- Sample accession
- Lane name
- Lane accession

For more information about EBI accession number format please see www.ebi.ac.uk/ena/submit/read-data-format.

You can also use pf to generate a spreadsheet with supplementary data, which can be useful for publication. `pf supplementary` will return:

- Sample name
- Sample accession
- Lane name
- Lane accession
- Supplier name
- Public name
- Strain
- Study ID
- Study accession

Optionally, `pf supplementary` can also return the sample description.

In this section of the tutorial we will cover:

- using `pf info` to get sample metadata
- using `pf accession` to get sample accessions
- using `pf supplementary` to get supplementary data.

4.2 Exercise 3

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

4.2.1 Metadata

We can get the metadata associated with our lanes using `pf info`.

Let's take a look at the usage information for `pf info`.

```
pf info -h
```

Let's get the sample name that corresponds to lane 5477_6#1.

```
pf info -t lane -i 5477_6#1
```

Here we can see that several pieces of metadata have been returned. One of these is the sample name: **Tw01_0055**.

Now, let's get the sample names for all lanes associated with study 664.

```
pf info -t study -i 664
```

We can write this information to file using the `-o` or `--outfile` option.

Let's write our lane metadata to file.

```
pf info -t study -i 664 -o
```

This has generated a new file "infofind.csv" which contains our comma-separated lane metadata.

```
cat infofind.csv
```

We can also give the output file a different name.

Let's call the metadata file for study 664 "study_664_info.csv".

```
pf info -t study -i 664 -o study_664_info.csv
```

This generates the file "study_664_info.csv" which contains our metadata.

```
cat study_664_info.csv
```

4.2.2 Accessions

If available, we can also get the EBI raw sequence data and sample accessions for the lanes associated with study 664 using `pf accession`.

Let's take a look at the usage information for `pf accession`.

```
pf accession -h
```

Let's get the EBI accessions for all lanes associated with study 664.

```
pf accession -t study -i 664
```

As with `pf info` we can also write the output of `pf accession` to a comma-delimited file.

Let's write the accessions associated with study 664 to a file called "study_664_accessions.csv".

```
pf accession -t study -i 664 -o study_664_accessions.csv
```

This generates the file "study_664_accessions.csv" which contains our comma-separated accessions.

```
cat study_664_accessions.csv
```

Finally, we can get the EBI URLs to download the raw data using the `-f` or `--fastq` option. By default, these will be written to a file called "fastq_urls.txt".

Let's get the URLs for downloading the FASTQ files for study 667 from the European Nucleodtide Archive (ENA).

```
pf accession -t study -i 664 -f
```

This generated a file called "fastq_urls.txt" which contained the URLs to download the raw sequencing data, one URL per file.

```
cat fastq_urls.txt
```

4.2.3 Supplementary data

We can get the supplementary data associated with our lanes using `pf supplementary`.

Let's take a look at the usage information for `pf supplementary`.

```
pf supplementary -h
```

Let's get the supplementary data for all lanes associated with study 664.

```
pf supplementary -t study -i 664
```

As with `pf info` and `pf accession` we can also write the output of `pf supplementary` to a comma-delimited file.

Let's write the supplementary data associated with study 664 to a file called "study_664_supplementary.csv".

```
pf supplementary -t study -i 664 -o study_664_supplementary.csv
```

This generates the file "study_664_supplementary.csv" which contains our comma-separated supplementary data.

```
cat study_664_supplementary.csv
```

Finally, we can include sample description in the supplementary information by using the `-d` or `--description` option.

Let's get the supplementary data for all lanes associated with study 664, including the sample description

```
pf supplementary -t study -i 664 -d
```

4.3 Questions

Q1: What is the sample name that corresponds with lane 10018_1#1?

```
# Enter your answer here
```

Q2: What lane name(s) correspond with sample APP_T1_OP2?

```
# Enter your answer here
```

Q3: What are the sample and lane names of the last lane in the file "data/lanes_to_search.txt".
Hint: use `tail -1` *to get the last line of the output*

```
# Enter your answer here
```

Q4: What are the sample and lane accessions for lane 5477_6#1?

```
# Enter your answer here
```

Q5: What are the two URLs which can be used to download the FASTQ files for lane 5477_6#1 from the ENA?

```
# Enter your answer here
```

```
# Enter your answer here
```

4.4 What's next?

You can head back to finding your data.

Otherwise, let's move on to looking at analysis pipeline status.

5 Analysis pipeline status

5.1 Introduction

You can use the `pf status` script to return information about the status of your samples within the automated analysis pipelines, allowing you to see which pipelines have been run on the data.

The automated analysis pipelines available include:

- Quality control (QC)
- Mapping
- SNP calling
- Bacterial, Eukaryote and Pacbio assembly
- Annotation
- RNA-Seq expression

Running `pf status` will return a table with one row per lane and one column per pipeline. In that table, you will see either a '-' meaning that the pipeline hasn't been run or, if the pipelines have been requested, 'Running', 'Done' or 'Failed' for each of the lanes.

Let's take lane 5477_6#1 as an example. Here is the output from `pf status`.

Name	Import	QC	Mapping	Archive	Improve	SNP call	RNASeq	Assemble	Annotate
5477_6#1	Done	Done	Done	Done	-	Done	-	Done	Done

This tells us that for lane 5477_6#1, the import, QC, mapping, archive, SNP calling, assembly and annotation pipelines have been run and are finished (Done).

In this section of the tutorial we will cover:

- using `pf status` to determine the status of samples in the various pathogen informatics pipelines

5.2 Exercise 4

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

We can get the status of all lanes associated with a study by setting the type (`-t` or `--type`) to "study" and giving the study ID or name as the identifier (`-i` or `--id`).

Let's get the status of the lanes associated with study 664.

```
pf status -t study -i 664
```

Here you can see that the import, QC, mapping, archive, SNP calling, assembly and annotation pipelines have been run and are finished (Done) for all of the lanes associated with study 664.

Let’s try this again using the study name, “Streptococcus pneumoniae global lineages”, instead.

```
pf status -t study -i "Streptococcus pneumoniae global lineages"
```

You can see that we get the same result as if we’d used the study ID. It’s important to remember to put the study name in quotes (") because it has spaces in in.

Let’s try using our study name without the quotes.

```
pf status -t study -i Streptococcus pneumoniae global lineages
```

Oh, errors and the usage. This is why you should get into the habbit of putting the study name between double quotes.

Let’s get the status of the lane 5477_6#1.

```
pf status -t lane -i 5477_6#1
```

Alternatively, we can get the sample name for that lane with `pf info` and use the sample name to get the status.

Let’s get the corresponding sample name for lane 5477_6#1.

```
pf info -t lane -i 5477_6#1
```

Now let’s use the sample name that was returned, Tw01_0055, to get the status.

```
pf status -t sample -i Tw01_0055
```

Finally, let’s get the status of the lanes in “data/lanes.txt”.

```
pf status -t file -i data/lanes.txt
```

5.3 Questions

Q1: Has the assembly pipeline been run on lane 10018_1#1? If so, what is the status?

```
# Enter your answer here
```

Q2: Which lanes in study 607 has the assembly pipeline been run on?
Hint: you could use `awk` to get the assembly column (9th column)

```
# Enter your answer here
```

Q3: How many lanes in study 607 has the mapping pipeline been run on?
Hint: you could use `awk` to get the mapping column (4th column) and `wc` to count the number of lines returned

```
# Enter your answer here
```

5.4 What's next?

For a quick recap of how to get metadata and accessions, head back to sample information and accessions.

Otherwise, let's move on to how to get your QC pipeline results.

6 Quality control (QC) pipeline results

6.1 Introduction

When your sample data is in the Pathogen Informatics databases, it becomes available to the automated analysis pipelines. After the analysis pipelines have been requested and run, you can use the `pf` scripts to return the results of each of the automated analysis pipelines.

First up, we're going to look at how you can get the output from the QC pipeline. The QC pipeline generates a series of QC statistics about your data and runs Kraken which assigns each read to a taxon and will broadly tell you what's been sequenced. To get the QC results, we use `pf qc` which returns the location of the Kraken report for a given study, sample or lane.

In this section of the tutorial we will cover:

- using `pf qc` to get Kraken reports
- using `pf qc` to get a summary of the Kraken report at different taxonomic levels

6.2 Exercise 5

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

Let's take a look at the `pf qc` usage.

```
pf qc -h
```

Now, let's get the QC pipeline results for lane 5477_6#1.

```
pf qc -t lane -i 5477_6#1
```

This returned the location of the Kraken report on disk.

Let's take a quick look at the Kraken report.

```
pf qc -t lane -i 5477_6#1 | xargs head
```

Notice that we used `xargs` to give the filename that was returned to another command, in this case `head`.

We can get a summary of this Kraken report using the `--summary` or `-s` option that will generate a new file called "qc_summary.csv" containing the taxon level Kraken results.

Let's get our *taxon* (strain) level QC summary for lane 5477_6#1.

```
pf qc -t lane -i 5477_6#1 -s
```

```
head qc_summary.csv
```

Here you can see the taxon level Kraken results i.e 1.08% of the reads were assigned to the *Streptococcus pneumoniae* strain Hungary19A-6.

We can look at the results for different taxonomic levels using the `--level` or `-L` option.

Let's try looking at the species level QC results for lane 5477_6#1.

```
pf qc -t lane -i 5477_6#1 -L S -s qc_species_summary.csv -F
```

```
head qc_species_summary.csv
```

Here we can see that 87.61% of the reads were classified as *Streptococcus pneumoniae*. This is promising as the sample is from *Streptococcus pneumoniae*.

6.2.1 QC Grind

The QC pipeline also generates a series of QC statistics for a given study, sample or lane which can be found on QC Grind.

6.3 Questions

Q1: What percentage of the reads from lane 10018_1#1 were "unclassified" by Kraken?
Hint: you can use `xargs` and `head` to look at the start of the Kraken report returned by `pf qc`

```
# Enter your answer here
```

Q2: What percentage of the reads from the lane 10018_1#1 were classified to the genus *Actinobacillus* by Kraken?
Hint: look at the level options in the `pf qc` usage

```
# Enter your answer here
```

```
# Enter your answer here
```

6.4 What's next?

For a quick recap of how to get metadata and accessions, head back to analysis pipeline status.

Otherwise, let's move on to how to get your mapping pipeline results.

7 Mapping pipeline results

7.1 Introduction

When your sample data is in the Pathogen Informatics databases, it becomes available to the automated analysis pipelines. After the analysis pipelines have been requested and run, you can use the `pf` scripts to return the results of each of the automated analysis pipelines.

The mapping pipeline maps your raw sequence reads to a reference that you selected. We can use `pf map` to return the location of the BAM files that were produced by the mapping pipeline.

In this section of the tutorial we will cover:

- using `pf map` to get BAM files generated by the mapping pipeline
- filtering `pf map` results by mapper and reference
- using `pf map` to symlink BAM files generated by the mapping pipeline
- using `pf map` to get mapping statistics

7.2 Exercise 6

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

Let's take a look at the `pf map` usage.

```
pf map -h
```

Now, let's get the mapping pipeline results for lane 5477_6#1.

```
pf map -t lane -i 5477_6#1
```

This returns the locations of the BAM files which were produced by the mapping pipeline.

A quick way to get information about which mapper and reference were used by the mapping pipeline is to use the `--details` or `-d` option.

Let's get the mapping details for lane 5477_6#1.

```
pf map -t lane -i 5477_6#1 -d
```

Here we can see that "smalt" was use as the **mapper** and "Streptococcus_pneumoniae_Taiwan19F-14_v1" was used as the **reference**.

You can request the mapping pipeline be run more than once using different mappers or reference. To filter the output by mapper we can use the `--mapper` or `-M` option and the `--reference` or `-R` option to filter by reference.

Let's look for mapping pipeline results for lane 5477_6#1 which used the mapper "smalt".

```
pf map -t lane -i 5477_6#1 -M smalt
```

Here we got the same results as before. But, what if we try looking for results produced by a different mapper?

Let's look for mapping pipeline results for lane 5477_6#1 which used the mapper "bwa".

```
pf map -t lane -i 5477_6#1 -M bwa
```

This gave us "No data found" as BWA hasn't been run on this lane.

Let's look for mapping pipeline results for lane 5477_6#1 which used the reference "Streptococcus_pneumoniae_Taiwan19F-14_v1".

```
pf map -t lane -i 5477_6#1 \
           -R "Streptococcus_pneumoniae_Taiwan19F-14_v1"
```

Notice that we only get one BAM file (.bam) and its index (.bai) returned. This is because the mapping pipeline has been run twice on this lane, once using the reference "Streptococcus_pneumoniae_Taiwan19F-14_v1" and once with "Streptococcus_pneumoniae_ATCC_700669_v1".

We could then symlink the BAM files into a directory using the `--symlink` or `-l` option.

Let's symlink our BAM files for lane 5477_6#1 to "my_bam_files".

```
pf map -t lane -i 5477_6#1 \
           -R "Streptococcus_pneumoniae_Taiwan19F-14_v1" \
           -l my_bam_files
```

```
ls my_bam_files
```

We can also get some statistics from our mapping results using the `--stats` or `-s` option.

Let's get some mapping statistics for lane 5477_6#1.

```
pf map -t lane -i 5477_6#1 -s
```

This generated a new file called "5477_6_1.mapping_stats.csv" which contains our mapping statistics.

```
cat 5477_6_1.mapping_stats.csv
```

Notice that there are two rows for lane 5477_6#1. This is because the mapping pipeline was run twice on this lane using different references.

7.3 Questions

Q1: How many BAM files are returned by default for lane 5477_6#10?

```
# Enter your answer here
```

Q2: Which mappers have been used with the mapping pipeline for lane 5477_6#10?
Hint: the mapper is in the 3rd column of the details

```
# Enter your answer here
```

Q3: Which references have been used with the mapping pipeline for lane 5477_6#10?
Hint: the reference is in the 2nd column of the details

```
# Enter your answer here
```

Q4: What percentage of the reads from lane 5477_6#10 were mapped to "Streptococcus_pneumoniae_OXC141_v1"?
Hint: you can use `awk` to filter the statistics file by column 8 (reference) (make sure you set -F',' as the stats are comma-delimited!)

```
# Enter your answer here
```

```
# Enter your answer here
```

7.4 What's next?

For a quick recap of how to get QC pipeline results, head back to QC pipeline results.

Otherwise, let's move on to how to get your SNP pipeline results.

8 SNP pipeline results

8.1 Introduction

When your sample data is in the Pathogen Informatics databases, it becomes available to the automated analysis pipelines. After the analysis pipelines have been requested and run, you can use the `pf` scripts to return the results of each of the automated analysis pipelines.

The SNP calling pipeline is composed of two parts, SNP calling and pseudogenome construction. We can use `pf snp` to return the location of the variant calling format (VCF) files that were produced by the SNP calling pipeline.

In this section of the tutorial we will cover:

- using `pf snp` to get VCF files generated by the SNP calling pipeline
- filtering `pf snp` results by mapper and reference
- using `pf snp` to symlink VCF files generated by the SNP calling pipeline
- using `pf snp` to generate a pseudogenome

8.2 Exercise 7

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

Let's take a look at the `pf snp` usage.

```
pf snp -h
```

Now, let's get the SNP calling pipeline results for lane 5477_6#1.

```
pf snp -t lane -i 5477_6#1
```

This returns the locations of the gzipped VCF files (.vcf.gz) and their indices (.vcf.gz.tbi) which were produced by the SNP calling pipeline.

The mapping pipeline is run before the SNP calling pipeline. A quick way to get information about which mapper and reference were used by the mapping pipeline is to use the `--details` or `-d` option.

Let's get the SNP calling details for lane 5477_6#1.

```
pf snp -t lane -i 5477_6#1 -d
```

Here we can see that "smalt" was use as the **mapper** for both the "Streptococcus_pneumoniae_ATCC_700669_v1" and "Streptococcus_pneumoniae_Taiwan19F-14_v1" **references**.

You can request the SNP calling pipeline be run more than once using different mappers or reference. To filter the output by mapper we can use the `--mapper` or `-M` option and the `--reference` or `-R` option to filter by reference.

Let's look for SNP calling pipeline results for lane 5477_6#1 which used the mapper "smalt".

```
pf snp -t lane -i 5477_6#1 -M smalt
```

Here we got the same results as before. But, what if we try looking for results produced by a different mapper?

Let's look for SNP calling pipeline results for lane 5477_6#1 which used the mapper "bwa".

```
pf snp -t lane -i 5477_6#1 -M bwa
```

This gave us "No data found" as BWA mapping hasn't been run on this lane.

Let's look for SNP calling pipeline results for lane 5477_6#1 which used the reference "Streptococcus_pneumoniae_Taiwan19F-14_v1".

```
pf snp -t lane -i 5477_6#1 \
           -R "Streptococcus_pneumoniae_Taiwan19F-14_v1"
```

Notice that we only get one VCF file (.bam) and its index (.tbi) returned. This is because the SNP calling pipeline has been run twice on this lane, once using the reference "Streptococcus_pneumoniae_Taiwan19F-14_v1" and once with "Streptococcus_pneumoniae_ATCC_700669_v1".

We could then symlink the gzipped VCF files into a directory using the `--symlink` or `-l` option.

Let's symlink our gzipped VCF files for lane 5477_6#1 to "my_vcf_files".

```
pf snp -t lane -i 5477_6#1 \
           -R "Streptococcus_pneumoniae_Taiwan19F-14_v1" \
           -l my_vcf_files
```

Finally, we can generate the **pseudogenome**, a custom genome that incorporates filtered variants with respect to the reference. To generate the pseudogenome, we use the `--pseudogenome` or `-p` option.

Let's get the pseudogenomes for lane 5477_6#1.

```
pf snp -t lane -i 5477_6#1 -p
```

Notice that there are two pseudogenome files for lane 5477_6#1. This is because the SNP calling has been run twice.

8.3 Questions

Q1: How many lanes from run 10018_1 has the SNP calling pipeline been completed on?
Hint: is there another `pf` command you can use to check the status of run?

```
# Enter your answer here
```

Q2: How many gzipped VCF files are returned by default for lane 10018_1#20?

```
# Enter your answer here
```

Q3: Which mapper and reference was used by the SNP calling pipeline for lane 10018_1#20?

```
# Enter your answer here
```

Q4: Generate the pseudogenome for lane 10018_1#20 excluding the reference.
Hint: look at the `pf snp` usage

```
# Enter your answer here
```

Q5: Symlink the gzipped VCF files generated by the SNP calling pipeline for run 10018_1 to a new directory called "10010_1_vcfs".

```
# Enter your answer here
```

8.4 What's next?

For a quick recap of how to get QC pipeline results, head back to mapping pipeline results.

Otherwise, let's move on to how to get your assembly pipeline results.

9 Assembly pipeline results

9.1 Introduction

When your sample data is in the Pathogen Informatics databases, it becomes available to the automated analysis pipelines. After the analysis pipelines have been requested and run, you can use the `pf` scripts to return the results of each of the automated analysis pipelines.

The genome assembly pipeline used depends on sequence data and organism:

- bacteria assembly
- virus assembly
- eukaryote assembly
- pacbio assembly

We can use `pf assembly` to return the location of assembly pipeline results.

In this section of the tutorial we will cover:

- using `pf assembly` to get assembly pipeline results
- filtering `pf assembly` results by program
- using `pf assembly` to symlink assembly pipeline results
- using `pf assembly` to get assembly statistics

9.2 Exercise 8

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

Let's take a look at the `pf assembly` usage.

```
pf assembly -h
```

Now, let's get the assembly pipeline results for run 5477_6#1.

```
pf assembly -t lane -i 5477_6#1
```

This returns the locations of the FASTA-formatted contig files which were produced by the assembly pipeline.

By default, `pf assembly` will return the scaffolded contigs. But, what if you want to see all of the assembled contigs. To get these we can use the `--filetype` or `-f` option.

```
pf assembly -t lane -i 5477_6#1 -f all
```

This returns a third file, "unscaffolded_contigs.fa".

Notice that the results are located in a directories which are named after the assembler that was used to generate the assembly e.g. "spades_assembly". This tells us that SPAdes was the program used to generate the assembly. A quick way to filter assembly pipeline results by program is to use the `--progam` or `-P` option.

Let's get all assembly pipeline results for run 5477_6 which were generated using "spades".

```
pf assembly -t lane -i 5477_6 -P spades
```

Here we can see that SPAdes was used to generate assemblies for lanes 5477_6#1 and 5477_6#3. We can symlink these assemblies into a directory using the `--symlink` or `-l` option.

Let's symlink the assembly pipeline results for run 5477_6 which were generated with SPAdes to "5477_6_spades".

```
pf assembly -t lane -i 5477_6 -P spades -l 5477_6_spades
```

```
ls 5477_6_spades
```

We can also get some statistics from our assembly results using the `--stats` or `-s` option.

Let's get some assembly statistics for lane 10018_1#2.

```
pf assembly -t lane -i 5477_6#1 -s
```

This generated a new file called "5477_6_1.assemblyfind_stats.csv" which contains our assembly statistics.

```
cat 5477_6_1.assemblyfind_stats.csv
```

9.3 Questions

Q1: How many assembly files are returned by default for lane 10018_1#50?

```
# Enter your answer here
```

Q2: Which program was used to generate the assembly for lane 10018_1#51?
Hint: look at the location path

```
# Enter your answer here
```

Q3: Symlink the assembly/assemblies generated by "IVA" for run 10018_1 into a new directory called "iva_results".
Hint: don't forget to filter the results if more than one program has been used

```
# Enter your answer here
```

Q4: How many contigs were assembled by velvet for lane 5477_6#2 and what is the N50?
Hint: you'll need to get some statistics for this lane and filter by program

```
# Enter your answer here
```

```
# Enter your answer here
```

9.4 What's next?

For a quick recap of how to get QC pipeline results, head back to SNP calling pipeline results.

Otherwise, let's move on to how to get your annotation pipeline results.

10 Annotation pipeline results

10.1 Introduction

When your sample data is in the Pathogen Informatics databases, it becomes available to the automated analysis pipelines. After the analysis pipelines have been requested and run, you can use the `pf` scripts to return the results of each of the automated analysis pipelines.

The annotation pipeline prepares genomes for submission to EMBL/GenBank in a standardised manner, with all the annotation files tracked centrally and available with the `pf annotation` command.

In this section of the tutorial we will cover:

- using `pf annotation` to get GFF files generated by the annotation pipeline
- filtering `pf annotation` results by assembler
- using `pf annotation` to symlink files generated by the annotation pipeline
- using `pf annotation` to get annotation statistics
- using `pf annotation` to check whether a gene is found in the GFF files generated by the annotation pipeline

10.2 Exercise 9

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

Let's take a look at the `pf annotation` usage.

```
pf annotation -h
```

Now, let's get the annotation pipeline results for lane 5477_6#1.

```
pf annotation -t lane -i 5477_6#1
```

This returns the locations of the GFF files which were produced by the annotation pipeline. Two paths were returned. This is because the assembly pipeline was run twice on this lane with two different assemblers: SPAdes and Velvet. Now, look closely at the annotation file paths. Did you notice that each annotation file that's returned is within an assembly directory? This is because your annotations and assemblies are linked.

We can filter the returned results, looking only for those results associated with a particular assembly program e.g. SPAdes, Velvet or IVA. To do this, we can use the `--program` or `-P` option as we did with `pf assembly`.

Let's get all annotation pipeline results for run 5477_6#1 which were generated using "spades".

```
pf annotation -t lane -i 5477_6#1 -P spades
```

That leaves us with just one annotation file in the "spades_assembly" directory.

Several different file formats are generated by the annotation pipeline. By default, `pf annotation` returns the GFF file produced by the annotation pipeline. We can also ask `pf annotation` to return a different filetype e.g. the protein FASTA file of the translated CDS sequences. To do this we use the `--filetype` or `-f` option.

Let's look for the protein FASTA file of the translated CDS sequences for lane 5477_6#1.

```
pf annotation -t lane -i 5477_6#1 -P spades -f faa
```

Here we got files returned with the extension ".faa" instead of the default ".gff" files. We can now symlink these FASTA files into a directory using the `--symlink` or `-l` option.

Let's symlink our protein FASTA files for our SPAdes assembly from lane 5477_6#1 to "my_protein_files".

```
pf annotation -t lane -i 5477_6#1 -P spades \
           -f faa -l my_protein_files
```

```
ls my_protein_files
```

We can also get some statistics from our annotation results using the `--stats` or `-s` option.

Let's get some annotation statistics for our SPAdes assembly from lane 5477_6#1.

```
pf annotation -t lane -i 5477_6#1 -P spades -s
```

This generated a new file called "5477_6_1.annotationfind_stats.csv" which contains our annotation statistics.

```
cat 5477_6_1.annotationfind_stats.csv
```

Finally, we can check to see if a gene is present in our sample using the `--gene` or `-g` option.

Let's see if any of our assemblies for lane 5477_6#1 contain the gene "*dnaG*".

```
pf annotation -t lane -i 5477_6#1 -g dnaG
```

Here we can see that both of our samples contain the *dnaG* gene. To check this, we can can use `grep`.

Use `grep` to search for "*dnaG*" in the SPAdes annotation for lane 5477_6#1.

```
pf annotation -t lane -i 5477_6#1 -P spades | xargs grep dnaG
```

10.3 Questions

Q1: How many GFF files are returned by default for lane 10018_1#1?

```
# Enter your answer here
```

Q2: What is the location of the annotation for the SPAdes assembly of lane 10018_1#1?

```
# Enter your answer here
```

Q3: What is the location of the translated CDS sequence file for the SPAdes assembly of lane 10018_1#1?
Hint: think about file extensions

```
# Enter your answer here
```

Q4: How many of the assemblies for run 5477_6 contain the gene "*dnaG*"?

```
# Enter your answer here
```

```
# Enter your answer here
```

10.4 What's next?

For a quick recap of how to get QC pipeline results, head back to assembly pipeline results.

Otherwise, let's move on to how to get your RNA-Seq expression pipeline results.

11 RNA-Seq pipeline results

11.1 Introduction

When your sample data is in the Pathogen Informatics databases, it becomes available to the automated analysis pipelines. After the analysis pipelines have been requested and run, you can use the `pf` scripts to return the results of each of the automated analysis pipelines.

The RNA-Seq expression pipeline maps your raw sequence reads to a reference and counts the number of reads associated with each gene. We can use `pf rnaseq` to return the location of the count files that were produced by the RNA-Seq expression pipeline.

In this section of the tutorial we will cover:

- using `pf rnaseq` to get count files generated by the RNA-Seq expression pipeline
- filtering `pf rnaseq` results by mapper and reference
- using `pf rnaseq` to symlink count files generated by the RNA-Seq expression pipeline
- using `pf rnaseq` to summarise the relationship between lane, sample and file names
- using `pf rnaseq` to get mapping statistics

11.2 Exercise 10

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

Let's take a look at the `pf rnaseq` usage.

```
pf rnaseq -h
```

Now, let's get the RNA-Seq expression pipeline results for lane 8479_4#17.

```
pf rnaseq -t lane -i 8479_4#17
```

By default, this returns the locations of the expression count files which were produced by the RNA-Seq expression pipeline.

For human and mouse data, there will also be featurecount files available which are produced by featureCounts. You can get these files using the `--filetype` or `-f` option.

```
pf rnaseq -t lane -i 8479_4#17 -f featurecounts
```

The mapping pipeline is run before the RNA-Seq expression pipeline. A quick way to get information about which mapper and reference were used by the mapping pipeline is to use the `--details` or `-d` option.

Let's get the mapping details for lane 8479_4#17.

```
pf rnaseq -t lane -i 8479_4#17 -d
```

Here we can see that "bwa" and "tophat" were used as the **mapper** and "Mus_musculus_mm9" and "Mus_musculus_mm10" were used as **references**.

You can request the RNA-Seq expression pipeline be run more than once using different mappers or reference. To filter the output by mapper we can use the `--mapper` or `-M` option and the `--reference` or `-R` option to filter by reference.

Let's look for RNA-Seq expression pipeline results for lane 8479_4#17 which used the mapper "bwa".

```
pf rnaseq -t lane -i 8479_4#17 -M bwa
```

Let's look for RNA-Seq expression pipeline results for lane 8479_4#17 which used the reference "Mus_musculus_mm9".

```
pf rnaseq -t lane -i 8479_4#17 -R "Mus_musculus_mm9"
```

We can symlink our count files into a directory using the `--symlink` or `-l` option.

Let's symlink our count files for lane 8479_4#17 to "my_count_files".

```
pf rnaseq -t lane -i 8479_4#17 -l my_count_files
```

```
ls my_count_files
```

You may want to know the relationship between the lane name, sample name and file name. We can get this relationship using the `--summary` or `-S` option.

```
pf rnaseq -t lane -i 8479_4#17 -S
```

This generates a new file called "8479_4_17.rnaseqfind_summary.tsv" which contains the output from `pf info` for this lane (Lane, Sample, Supplier_Name, Public_Name, Strain), the filename that is symlinked (Filename) and the full location to that file (File_Path). This may also be useful as a starting point for your DEAGO targets file.

```
cat 8479_4_17.rnaseqfind_summary.tsv
```

We can also get some mapping statistics from our RNA-Seq results using the `--stats` or `-s` option.

Let's get some mapping statistics for lane 8479_4#17.

```
pf rnaseq -t lane -i 8479_4#17 -s
```

This generated a new file called "8479_4_17.rnaseqfind_stats.csv" which contains our mapping statistics.

```
cat 8479_4_17.rnaseqfind_stats.csv
```

Notice that there are two rows for lane 8479_4#17. This is because the mapping pipeline was run twice on this lane using different references and mappers.

11.3 Questions

Q1: How many count files are returned by default for run 8479_4?

```
# Enter your answer here
```

Q2: Which mapper was used with the mapping pipeline for lane 8479_4#18?
Hint: the mapper is in the 3rd column of the details

```
# Enter your answer here
```

Q3: Which reference was used with the mapping pipeline for lane 8479_4#18?
Hint: the reference is in the 2nd column of the details

```
# Enter your answer here
```

Q4: What is the location or path of the featurecounts file for lane 8479_4#18?

```
# Enter your answer here
```

Q5: Which of the lanes in run 8479_4 has the lowest percentage of mapped reads?
Hint: you can use `awk` to print out column 2 (lane name) and column 12 (mapped %) of your run statistics

```
# Enter your answer here
```

```
# Enter your answer here
```

Q6: What is the sample name and symlinked file name associated with lane 8479_4#18?
Hint: you might want to summarise the results

11.4 What's next?

For a quick recap of how to get annotation pipeline results, head back to annotation pipeline results.

Otherwise, let's move on to how to find a reference.

12 Finding a reference

12.1 Introduction

For a reference to be used with the Pathogen Informatics analysis pipelines, the reference must first be in the pathogen databases. All complete bacterial genomes from RefSeq are automatically imported and updated. There are user-submitted references too.

We can look at the available reference sequences using `pf ref`. This differs from the other `pf` scripts that we've looked at in that it doesn't require a type (`-t`). It only needs a partial id (`id` or `i`).

In this section of the tutorial we will cover:

- using `pf ref` to find a FASTA-formatted reference
- using `pf ref` to find the GFF annotation for a reference
- using `pf ref` to symlink a reference

12.2 Exercise 11

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

We can find a reference using part of the reference name using `pf ref`.

Let's take a look at the usage information for `pf ref`.

```
pf ref -h
```

So, if we wanted to look which mouse (*Mus_musculus*) references are available we can run:

```
pf ref -i Mus_musculus
```

Notice that there are no spaces between the genus, species and strain. Instead, these are replaced with underscores!

The command above would give you:

```
No exact match for "Mus_musculus". Did you mean:
  [1] Mus_musculus_mm10
  [2] Mus_musculus_mm9
  [a] all references
Which reference?
```

You would then enter the number corresponding to the reference location you need. Say we want to find the reference for "Mus_musculus_mm9", we would enter 1 which would give us:

```
/path/to/refs/Mus/musculus/Mus_musculus_mm10.fa
```

We can't do that in this notebook as it will wait forever in the background for us to enter an option. So, instead we need to use the `--all` or `-A` option to list all of the available references that match our query.

Let's see which *Salmonella* references are available.

```
pf ref -i Salmonella -A
```

This gives us the location of the reference FASTA file on disk. However, maybe we just want to see the reference names. We can do this using the `--reference-names` or `-R` option. These can be useful where you need to specify a reference name when requesting the analysis pipelines on the command line.

Now, let's get the *Salmonella* reference names.

```
pf ref -i Salmonella -A -R
```

Notice the version numbers at the end of the reference name. There is usually a naming convention with the references based on their source:

- RefSeq accession (e.g. GCF_001887015_1) - complete genome imported from RefSeq
- version (v) >=1 (e.g. v1) - genome requested by user and imported from public repository (e.g. ENA/GenBank)
- version (v) <1 (e.g. v0.1) - internal genome assembly requested by user

But, perhaps you don't want the FASTA file, perhaps you want the reference annotation (i.e. GFF file). To get this, we need to use the `--filetype` or `-f` option.

Let's get the annotation (GFF) locations for the available *Salmonella* references.

```
pf ref -i Salmonella -A -f gff
```

Finally, you might want to use the reference files in an analysis. The simplest way is to symlink them using the `--symlink` or `-l` option.

Let's symlink our *Salmonella* reference genomes to a directory called "salmonella_refs".

```
pf ref -i Salmonella -A -l salmonella_refs
```

```
ls salmonella_refs
```

12.3 Questions

Don't forget to use the **`-A`** option for all of these questions if you're running the `pf ref` commands inside the notebook!

Q1: How many *Streptococcus pneumoniae* references are available?
Hint: you can use `wc` *to count the number of references returned*

```
# Enter your answer here
```

Q2: How many of those *Streptococcus pneumoniae* references were imported from a public repository ?
Hint: think about the version in the suffix, using the `-R` *option to get only the names might make things clearer*

```
# Enter your answer here
```

Q3: What is the location of the annotation (GFF) file for *Streptococcus pneumoniae P1031*.

```
# Enter your answer here
```

Q4: Symlink the annotation (GFF) file for *Streptococcus pneumoniae P1031* to your current directory.

```
# Enter your answer here
```

12.4 What's next?

You can head back to RNA-Seq expression pipeline results.

Otherwise, let's move on to looking at troubleshooting.

13 Troubleshooting

First, let's tell the system the location of our tutorial configuration file.

```
export PF_CONFIG_FILE=$PWD/data/pathfind.conf
```

13.1 Getting help with the commands

Remember that you can always look at the usage for `pf` and the individual `pf` scripts (e.g. `pf data`) if you get stuck.

```
pf -h
```

```
pf data -h
```

13.2 Getting "No data found"

There are several reasons why you may be getting "No data found" when you search using the `pf` scripts. All of these can be found on the Pathogen Informatics Wiki FAQs here:

http://mediawiki.internal.sanger.ac.uk/index.php/Pathogen_Informatics_FAQs.

If you're still stuck after reading the wiki, you can email Pathogen Informatics at **path-help@sanger.ac.uk**.

13.3 Finding out about the analysis pipelines

There are links to the various analysis pipeline documentation on the main wiki page:

`http://mediawiki.internal.sanger.ac.uk/index.php/Pathogen_Informatics`

Congratulations, you have now reached the end of this tutorial!

To go back to the beginning, you'll want to head to the index.
Or, for the answers to the tutorial questions, head to the answer section.