# Wrangling genomics

2023-11-16

# Table of contents

## **Preface**

A lot of genomics analysis is done using command-line tools for three reasons:

- 1. you will often be working with a large number of files, and working through the commandline rather than through a graphical user interface (GUI) allows you to automate repetitive tasks,
- 2. you will often need more compute power than is available on your personal computer, and connecting to and interacting with remote computers requires a command-line interface, and
- 3. you will often need to customize your analyses, and command-line tools often enable more customization than the corresponding GUI tools (if in fact a GUI tool even exists).

In a previous lesson, you learned how to use the bash shell to interact with your computer through a command line interface. In this lesson, you will be applying this new knowledge to carry out a common genomics workflow - identifying variants among sequencing samples taken from multiple individuals within a population. We will be starting with a set of sequenced reads (.fastq files), performing some quality control steps, aligning those reads to a reference genome, and ending by identifying and visualizing variations among these samples.

As you progress through this lesson, keep in mind that, even if you aren't going to be doing this same workflow in your research, you will be learning some very important lessons about using command-line bioinformatic tools. What you learn here will enable you to use a variety of bioinformatic tools with confidence and greatly enhance your research efficiency and productivity.

#### Prerequisites

This lesson assumes a working understanding of the bash shell. If you haven't already completed the Shell Genomics lesson, and aren't familiar with the bash shell, please review those materials before starting this lesson.

This lesson also assumes some familiarity with biological concepts, including the structure of DNA, nucleotide abbreviations, and the concept of genomic variation within a population.

## Schedule

	Setup	Download files required for the lesson
00:00	1. Background and metadata	What data are we using? Why is this experiment important?
00:15	2. Assessing read quality	How can I describe the quality of my data?
01:05	3. Trimming and filtering	How can I get rid of sequence data that does not meet my quality standards?
02:00	4. Variant calling workflow	How do I find sequence variants between my sample and a reference genome?
03:00	5. Automating a variant calling workflow	How can I make my workflow more efficient and less error-prone?
03:45	Finish	

The actual schedule may vary slightly depending on the topics and exercises chosen by the instructor.

# 1 Background and metadata

#### Time

Teaching: 10 minExercises: 5 min

#### Questions

- What data are we using?
- Why is this experiment important?
- Why study E. coli?
- What is hypermutability?

#### Objectives

• Understand the data set.

## 1.1 Background

We are going to use a long-term sequencing data set from a population of Escherichia coli.

### 1.1.1 What is *E. coli*?

E. coli are rod-shaped bacteria that can survive under a wide variety of conditions including variable temperatures, nutrient availability, and oxygen levels. Most strains are harmless, but some are associated with food-poisoning.

### 1.1.2 Why is *E. coli* important?

 $E.\ coli$  are one of the most well-studied model organisms in science. As a single-celled organism,  $E.\ coli$  reproduces rapidly, typically doubling its population every 20 minutes, which means it can be manipulated easily in experiments. In addition, most naturally occurring strains of E.

coli are harmless. Most importantly, the genetics of *E. coli* are fairly well understood and can be manipulated to study adaptation and evolution.

### 1.2 The data

The data we are going to use is part of a long-term evolution experiment led by Richard Lenski.

The experiment was designed to assess adaptation in  $E.\ coli$ . A population was propagated for more than 40,000 generations in a glucose-limited minimal medium (in most conditions glucose is the best carbon source for  $E.\ coli$ , providing faster growth than other sugars). This medium was supplemented with citrate, which  $E.\ coli$  cannot metabolize in the aerobic conditions of the experiment. Sequencing of the populations at regular time points revealed that spontaneous citrate-using variant (Cit+) appeared between 31,000 and 31,500 generations, causing an increase in population size and diversity. In addition, this experiment showed hypermutability in certain regions. Hypermutability is important and can help accelerate adaptation to novel environments, but also can be selected against in well-adapted populations.

To see a timeline of the experiment to date, check out this figure, and this paper by Blount, Borland, and Lenski (2008) titled "Historical contingency and the evolution of a key innovation in an experimental population of Escherichia coli".

#### 1.2.1 View the metadata

We will be working with three sample events from the **Ara-3** strain of this experiment, one from 5,000 generations, one from 15,000 generations, and one from 50,000 generations. The population changed substantially during the course of the experiment, and we will be exploring how (the evolution of a **Cit+** mutant and **hypermutability**) with our variant calling workflow. The metadata file associated with this lesson can be downloaded directly here or viewed in **Github**. If you would like to know details of how the file was created, you can look at some notes and sources here.

This metadata describes information on the Ara-3 clones and the columns represent:

	Column Description
strain	strain name
generation	generation when sample frozen
clade	based on parsimony-based tree
reference	study the samples were originally sequenced for
population	ancestral population group
mutator	hypermutability mutant status
facility	facility samples were sequenced at

Column Description

run Sequence read archive sample ID

read\_type library type of reads read\_length length of reads in sample sequencing\_depth depth of sequencing

cit citrate-using mutant status

### Exercise

Based on the metadata, can you answer the following questions?

- 1. How many different generations exist in the data?
- 2. How many rows and how many columns are in this data?
- 3. How many citrate+ mutants have been recorded in Ara-3?
- 4. How many hypermutable mutants have been recorded in Ara-3?

### Solution

- 1. 25 different generations
- 2. 62 rows, 12 columns
- 3. 10 citrate+ mutants
- 4. 6 hypermutable mutants

## 1.3 Summary

! Key point

It is important to record and understand your experiment's metadata.

# 2 Assessing read quality

#### Time

Teaching: 30 minExercises: 20 min

#### Question

• How can I describe the quality of my data?

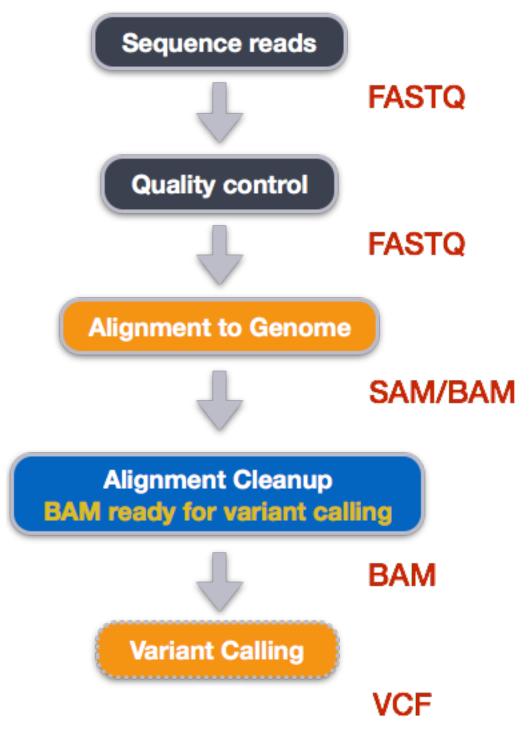
### Objectives

- Explain how a FASTQ file encodes per-base quality scores.
- Interpret a FastQC plot summarizing per-base quality across all reads.
- Use for loops to automate operations on multiple files.

### 2.1 Bioinformatic workflows

When working with high-throughput sequencing data, the raw reads you get off of the sequencer will need to pass through a number of different tools in order to generate your final desired output. The execution of this set of tools in a specified order is commonly referred to as a workflow or a pipeline.

An example of the workflow we will be using for our variant calling analysis is provided below with a brief description of each step.



- 1. Quality control Assessing quality
- 2. Quality control Trimming and/or filtering reads (if necessary)

- 3. Align reads to reference genome
- 4. Perform post-alignment clean-up
- 5. Variant calling

These workflows in bioinformatics adopt a plug-and-play approach in that the output of one tool can be easily used as input to another tool without any extensive configuration. Having standards for data formats is what makes this feasible. Standards ensure that data is stored in a way that is generally accepted and agreed upon within the community. The tools that are used to analyze data at different stages of the workflow are therefore built under the assumption that the data will be provided in a specific format.

### 2.2 Get the data

Often times, the first step in a bioinformatic workflow is getting the data you want to work with onto a computer where you can work with it. If you have outsourced sequencing of your data, the sequencing center will usually provide you with a link that you can use to download your data. Today we will be working with publicly available sequencing data.

We are studying a population of *Escherichia coli* (designated Ara-3), which were propagated for more than 50,000 generations in a glucose-limited minimal medium. We will be working with three samples from this experiment, one from 5,000 generations, one from 15,000 generations, and one from 50,000 generations. The population changed substantially during the course of the experiment, and we will be exploring how with our variant calling workflow.

The data are paired-end, so we will download two files for each sample. We will use the European Nucleotide Archive to get our data. The ENA "provides a comprehensive record of the world's nucleotide sequencing information, covering raw sequencing data, sequence assembly information and functional annotation." The ENA also provides sequencing data in the fastq format, an important format for sequencing reads that we will be learning about today.

### 2.2.1 Copy directory containing data on the HPC

Using the terminal, you can access the remote server using the ssh command (use your username)

```
$ ssh [username]@hpc.tsl.ac.uk
```

Now that you are logged in on the server, we must start an interactive session.

\$ interactive

Copy the directory containing the dataset to your home directory.

```
$ cp -r /tsl/data/dc_workshop/ ~
```

Now you can access the directory that now should contain the fastq files.

```
$ cd ~/dc_workshop/data/untrimmed_fastq
```

The data comes in a compressed format, which is why there is a .gz at the end of the file names. This makes it faster to transfer, and allows it to take up less space on our computer. Let's unzip one of the files so that we can look at the fastq format.

```
$ gunzip SRR2584863_1.fastq.gz
```

It takes a few seconds.

## 2.3 Quality control

#### 2.3.1 Details on the FASTQ format

Although it looks complicated (and it is), we can understand the fastq format with a little decoding. Some rules about the format include...

Line	Description
1	Always begins with '@' and then information about the read
2	The actual DNA sequence
3	Always begins with a '+' and sometimes the same info in line 1
4	Has a string of characters which represent the quality scores;
	must have same number of characters as line 2

We can view the first complete read in one of the files our data set by using head to look at the first four lines.

Line 4 shows the quality for each nucleotide in the read. Quality is interpreted as the probability of an incorrect base call (e.g. 1 in 10) or, equivalently, the base call accuracy (e.g. 90%). To make it possible to line up each individual nucleotide with its quality score, the numerical score is converted into a code where each individual character represents the numerical quality score for an individual nucleotide. For example, in the line above, the quality score line is:

#### CCCFFFFFGHHHHJIJJJJIIJJJIIIJJJIIIJJGFIIIJEDDFEGGJIFHHJIJJDECCGGEGIIJFHFFFACD:BBBDDACCCCAA@@

The numerical value assigned to each of these characters depends on the sequencing platform that generated the reads. The sequencing machine used to generate our data uses the standard Sanger quality PHRED score encoding, using Illumina version 1.8 onwards. Each character is assigned a quality score between 0 and 41 as shown in the chart below.

```
Quality encoding: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJ
```

Each quality score represents the probability that the corresponding nucleotide call is incorrect. This quality score is logarithmically based, so a quality score of 10 reflects a base call accuracy of 90%, but a quality score of 20 reflects a base call accuracy of 99%. These probability values are the results from the base calling algorithm and depend on how much signal was captured for the base incorporation.

Looking back at our read:

We can now see that there is a range of quality scores, but that the end of the sequence is very poor (# = a quality score of 2).

#### Exercise

What is the last read in the SRR2584863\_1.fastq file? How confident are you in this read?

#### Solution

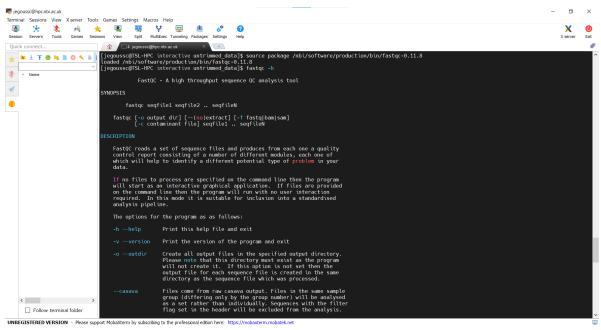
This read has more consistent quality at its end than the first read that we looked at, but still has a range of quality scores, most of them high. We will look at variations in position-based quality in just a moment.

At this point, let's load the relevant tools that are already installed on the server:

```
$ source package /nbi/software/production/bin/fastqc-0.11.8
loaded /nbi/software/production/bin/fastqc-0.11.8
```

One way to validate if the correct software is loaded and ready to work with is to check the software's manual:

\$ fastqc -h



If FastQC is not installed, you will get an error message:

```
$ fastqc -h
The program 'fastqc' is currently not installed. You can install it by typing:
sudo apt-get install fastqc
```

If this happens check with your instructor before trying to install it.

#### 2.3.2 Assessing read quality with FastQC

In real life, you will not be assessing the quality of your reads by visually inspecting your FASTQ files. Rather, you will be using a software program to assess read quality and filter out poor quality reads. We will first use a program called FastQC (andrews2010fastqcto?) visualize the quality of our reads. Later in our workflow, we will use another program to filter out poor quality reads.

FastQC has a number of features which can give you a quick impression of any problems your data may have, so you can take these issues into consideration before moving forward with your analyses. Rather than looking at quality scores for each individual read, FastQC looks at quality collectively across all reads within a sample. The image below shows one FastQC-generated plot that indicates a very high quality sample:

The x-axis displays the base position in the read, and the y-axis shows quality scores. In this example, the sample contains reads that are 40 bp long. This is much shorter than the reads we are working with in our workflow. For each position, there is a box-and-whisker plot showing the distribution of quality scores for all reads at that position. The horizontal red line indicates the median quality score and the yellow box shows the 1st to 3rd quartile range. This means that 50% of reads have a quality score that falls within the range of the yellow box at that position. The whiskers show the absolute range, which covers the lowest (0th quartile) to highest (4th quartile) values.

For each position in this sample, the quality values do not drop much lower than 32. This is a high quality score. The plot background is also color-coded to identify good (green), acceptable (yellow), and bad (red) quality scores.

Now let's take a look at a quality plot on the other end of the spectrum.

Here, we see positions within the read in which the boxes span a much wider range. Also, quality scores drop quite low into the "bad" range, particularly on the tail end of the reads. The FastQC tool produces several other diagnostic plots to assess sample quality, in addition to the one plotted above.

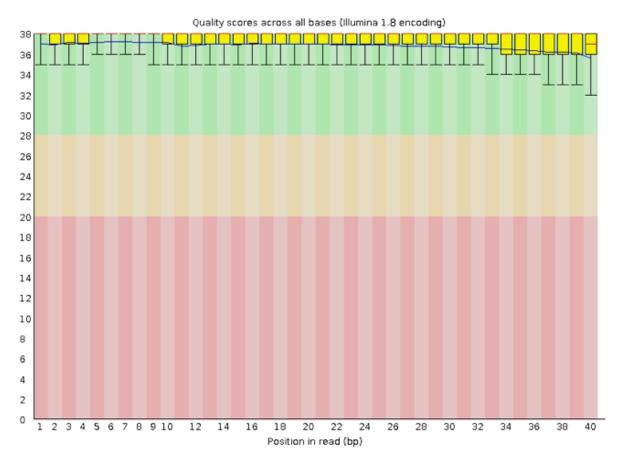


Figure 2.1: good\_quality

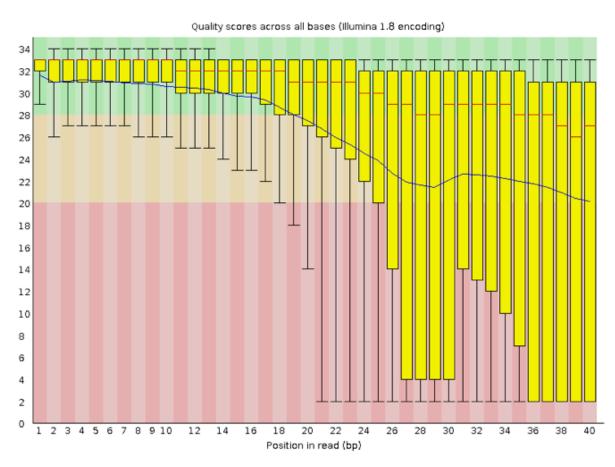


Figure 2.2: bad\_quality

### 2.3.3 Running FastQC

We will now assess the quality of the reads that we downloaded. First, make sure you are still in the untrimmed\_fastq directory

```
cd ~/dc_workshop/data/untrimmed_fastq/
```

#### Exercise

How big are the files? (Hint: Look at the options for the ls command to see how to show file sizes.)

```
$ 1s -1 -h
-rwx------ 1 [username] TSL_20 545M Aug 11 11:02 SRR2584863_1.fastq
-rwx------ 1 [username] TSL_20 183M Aug 11 11:03 SRR2584863_2.fastq.gz
-rwx------ 1 [username] TSL_20 309M Aug 11 11:04 SRR2584866_1.fastq.gz
-rwx------ 1 [username] TSL_20 296M Aug 11 11:07 SRR2584866_2.fastq.gz
-rwx------ 1 [username] TSL_20 124M Aug 11 11:00 SRR2589044_1.fastq.gz
-rwx------ 1 [username] TSL_20 124M Aug 11 11:01 SRR2589044_2.fastq.gz

There are six FASTQ files ranging from 124M (124MB) to 545M.
```

FastQC can accept multiple file names as input, and on both zipped and unzipped files, so we can use the \*.fastq\* wildcard to run FastQC on all of the FASTQ files in this directory.

```
$ fastqc *.fastq*
```

You will see an automatically updating output message telling you the progress of the analysis. It will start like this:

```
Started analysis of SRR2584863_1.fastq
Approx 5% complete for SRR2584863_1.fastq
Approx 10% complete for SRR2584863_1.fastq
Approx 15% complete for SRR2584863_1.fastq
Approx 20% complete for SRR2584863_1.fastq
Approx 25% complete for SRR2584863_1.fastq
Approx 30% complete for SRR2584863_1.fastq
Approx 35% complete for SRR2584863_1.fastq
Approx 45% complete for SRR2584863_1.fastq
Approx 45% complete for SRR2584863_1.fastq
Approx 45% complete for SRR2584863_1.fastq
```

In total, it should take about five minutes for FastQC to run on all six of our FASTQ files. When the analysis completes, your prompt will return. So your screen will look something like this:

```
Approx 80% complete for SRR2589044_2.fastq.gz
Approx 85% complete for SRR2589044_2.fastq.gz
Approx 90% complete for SRR2589044_2.fastq.gz
Approx 95% complete for SRR2589044_2.fastq.gz
Analysis complete for SRR2589044_2.fastq.gz
$
```

The FastQC program has created several new files within our data/untrimmed\_fastq/ directory.

```
$ ls
SRR2584863 1.fastq
                          SRR2584866 1 fastqc.html
                                                    SRR2589044_1_fastqc.html
                                                     SRR2589044_1_fastqc.zip
SRR2584863_1_fastqc.html
                          SRR2584866_1_fastqc.zip
SRR2584863_1_fastqc.zip
                          SRR2584866_1.fastq.gz
                                                    SRR2589044_1.fastq.gz
SRR2584863_2_fastqc.html
                          SRR2584866_2_fastqc.html
                                                    SRR2589044_2_fastqc.html
SRR2584863_2_fastqc.zip
                          SRR2584866_2_fastqc.zip
                                                    SRR2589044_2_fastqc.zip
SRR2584863_2.fastq.gz
                          SRR2584866_2.fastq.gz
                                                    SRR2589044_2.fastq.gz
```

For each input FASTQ file, FastQC has created a .zip file and a

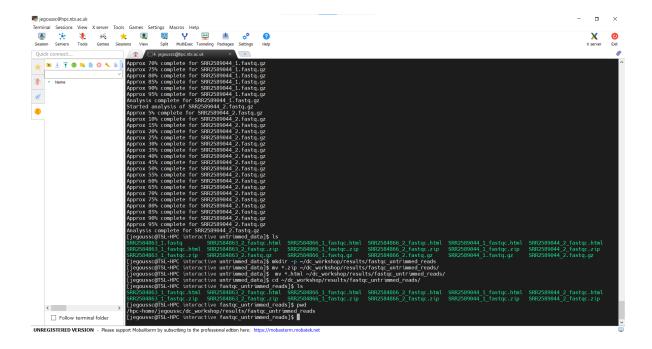
.html file. The .zip file extension indicates that this is actually a compressed set of multiple output files. We will be working with these output files soon. The .html file is a stable webpage displaying the summary report for each of our samples.

We want to keep our data files and our results files separate, so we will move these output files into a new directory within our results/ directory.

```
$ mkdir -p ~/dc_workshop/results/fastqc_untrimmed_reads
$ mv *.zip ~/dc_workshop/results/fastqc_untrimmed_reads/
$ mv *.html ~/dc_workshop/results/fastqc_untrimmed_reads/
```

Now we can navigate into this results directory and do some closer inspection of our output files.

```
$ cd ~/dc_workshop/results/fastqc_untrimmed_reads/
```



### 2.3.4 Viewing the FastQC results

If we were working on our local computers, we would be able to look at each of these HTML files by opening them in a web browser.

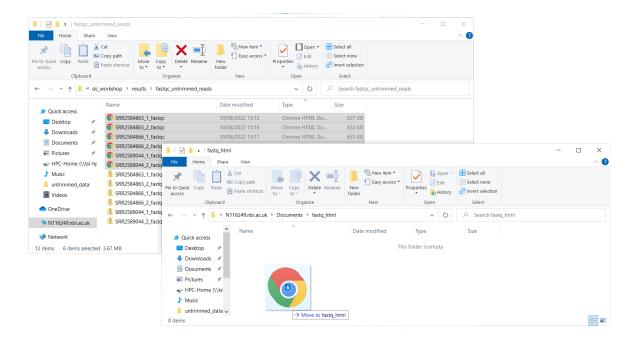
However, these files are currently sitting on our remote server, where our local computer can not see them. And, since we are only logging into the server via the command line - it does not have any web browser setup to display these files either.

So the easiest way to look at these webpage summary reports will be to transfer them to our local computers (i.e. your laptop).

To transfer a file from a remote server to our own machines, we will simply use the Windows navigation system.

- 1. From the Start Menu, select Run, then type \\tsl-hpc-data\HPC-Home
- 2. You may need to toggle Use different credentials on or off

A window with the directory showing the files in should pop up



Now we can go to our new directory and open the 6 HTML files.

Depending on your system, you should be able to select and open them all at once via a right click menu in your file browser.

#### Exercise

Discuss your results with a neighbor. Which sample(s) looks the best in terms of per base sequence quality? Which sample(s) look the worst?

### Solution

All of the reads contain usable data, but the quality decreases toward the end of the reads.

### 2.3.5 Decoding the other FastQC outputs

We have now looked at quite a few "Per base sequence quality" FastQC graphs, but there are nine other graphs that we have not talked about! Below we have provided a brief overview of interpretations for each of these plots. For more information, please see the FastQC documentation here

• Per tile sequence quality: the machines that perform sequencing are divided into tiles. This plot displays patterns in base quality along these tiles. Consistently low scores are

often found around the edges, but hot spots can also occur in the middle if an air bubble was introduced at some point during the run.

- Per sequence quality scores: a density plot of quality for all reads at all positions. This plot shows what quality scores are most common.
- Per base sequence content: plots the proportion of each base position over all of the reads. Typically, we expect to see each base roughly 25% of the time at each position, but this often fails at the beginning or end of the read due to quality or adapter content.
- Per sequence GC content: a density plot of average GC content in each of the reads.
- Per base N content: the percent of times that 'N' occurs at a position in all reads. If there is an increase at a particular position, this might indicate that something went wrong during sequencing.
- Sequence Length Distribution: the distribution of sequence lengths of all reads in the file. If the data is raw, there is often on sharp peak, however if the reads have been trimmed, there may be a distribution of shorter lengths.
- Sequence Duplication Levels: A distribution of duplicated sequences. In sequencing, we expect most reads to only occur once. If some sequences are occurring more than once, it might indicate enrichment bias (e.g. from PCR). If the samples are high coverage (or RNA-seq or amplicon), this might not be true.
- Overrepresented sequences: A list of sequences that occur more frequently than would be expected by chance.
- Adapter Content: a graph indicating where adapater sequences occur in the reads.
- **K-mer Content**: a graph showing any sequences which may show a positional bias within the reads.

#### 2.3.6 Working with the FastQC text output

Now that we have looked at our HTML reports to get a feel for the data, let's look more closely at the other output files. Go back to the tab in your terminal program that is connected to your interactive session on the server and make sure you are in our results subdirectory.

Our .zip files are compressed files. They each contain multiple different types of output files for a single input FASTQ file. To view the contents of a .zip file, we can use the program unzip to decompress these files. Let's try doing them all at once using a wildcard.

```
$ unzip *.zip
Archive: SRR2584863_1_fastqc.zip
caution: filename not matched:
                               SRR2584863 2 fastqc.zip
caution: filename not matched: SRR2584866_1_fastqc.zip
caution: filename not matched: SRR2584866_2_fastqc.zip
caution: filename not matched: SRR2589044_1_fastqc.zip
caution: filename not matched:
                               SRR2589044_2_fastqc.zip
```

This did not work. We unzipped the first file and then got a warning message for each of the other .zip files. This is because unzip expects to get only one zip file as input. We could go through and unzip each file one at a time, but this is very time consuming and error-prone. Someday you may have 500 files to unzip!

A more efficient way is to use a for loop like we learned in the Shell Genomics lesson to iterate through all of our .zip files. Let's see what that looks like and then we will discuss what we are doing with each line of our loop.

Warning

Do not copy-paste the four lines at once. You must type the for loop line by line!

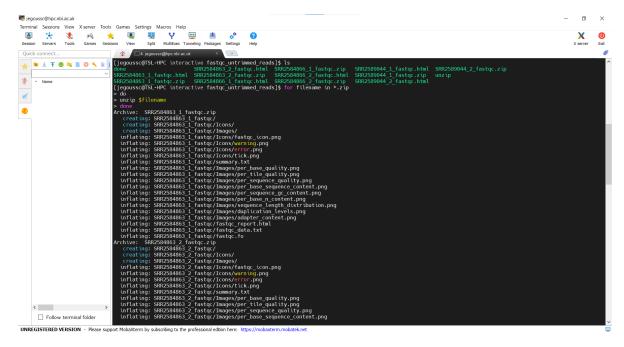
```
$ for filename in *.zip
> do
> unzip $filename
> done
```

In this example, the input is six filenames (one filename for each of our .zip files). Each time the loop iterates, it will assign a file name to the variable filename and run the unzip command. The first time through the loop, \$filename is SRR2584863\_1\_fastqc.zip. The interpreter runs the command unzip on SRR2584863\_1\_fastqc.zip. For the second iteration, \$filename becomes SRR2584863\_2\_fastqc.zip. This time, the shell runs unzip on SRR2584863\_2\_fastqc.zip. It then repeats this process for the four other .zip files in our directory.

When we run our for loop, you will see output that starts like this:

Archive: SRR2589044\_2\_fastqc.zip creating: SRR2589044\_2\_fastqc/ creating: SRR2589044\_2\_fastqc/Icons/

```
creating: SRR2589044_2_fastqc/Images/
inflating: SRR2589044_2_fastqc/Icons/fastqc_icon.png
inflating: SRR2589044_2_fastqc/Icons/warning.png
inflating: SRR2589044_2_fastqc/Icons/error.png
inflating: SRR2589044 2 fastqc/Icons/tick.png
inflating: SRR2589044_2_fastqc/summary.txt
inflating: SRR2589044 2 fastqc/Images/per base quality.png
inflating: SRR2589044_2_fastqc/Images/per_tile_quality.png
inflating: SRR2589044_2_fastqc/Images/per_sequence_quality.png
inflating: SRR2589044_2_fastqc/Images/per_base_sequence_content.png
inflating: SRR2589044_2_fastqc/Images/per_sequence_gc_content.png
inflating: SRR2589044_2_fastqc/Images/per_base_n_content.png
inflating: SRR2589044 2 fastqc/Images/sequence_length_distribution.png
inflating: SRR2589044_2_fastqc/Images/duplication_levels.png
inflating: SRR2589044_2_fastqc/Images/adapter_content.png
inflating: SRR2589044_2_fastqc/fastqc_report.html
inflating: SRR2589044_2_fastqc/fastqc_data.txt
inflating: SRR2589044_2_fastqc/fastqc.fo
```



The unzip program is decompressing the .zip files and creating a new directory (with subdirectories) for each of our samples, to store all of the different output that is produced by FastQC. There

are a lot of files here. The one we are going to focus on is the summary.txt file.

If you list the files in our directory now you will see:

```
SRR2584863_1_fastqc
                          SRR2584866_1_fastqc
                                                    SRR2589044_1_fastqc
SRR2584863_1_fastqc.html
                          SRR2584866_1_fastqc.html
                                                    SRR2589044_1_fastqc.html
SRR2584863 1 fastqc.zip
                          SRR2584866 1 fastqc.zip
                                                    SRR2589044 1 fastqc.zip
SRR2584863 2 fastqc
                                                    SRR2589044 2 fastqc
                          SRR2584866 2 fastqc
                                                    SRR2589044 2 fastqc.html
SRR2584863 2 fastqc.html
                          SRR2584866 2 fastqc.html
SRR2584863_2_fastqc.zip
                          SRR2584866_2_fastqc.zip
                                                    SRR2589044_2_fastqc.zip
```

The .html files and the uncompressed .zip files are still present, but now we also have a new directory for each of our samples. We can see for sure that it is a directory if we use the -F flag for ls.

```
$ 1s -F
SRR2584863 1 fastqc/
                          SRR2584866 1 fastqc/
                                                     SRR2589044 1 fastqc/
                          SRR2584866 1 fastqc.html
                                                    SRR2589044 1 fastqc.html
SRR2584863 1 fastqc.html
SRR2584863_1_fastqc.zip
                          SRR2584866_1_fastqc.zip
                                                     SRR2589044_1_fastqc.zip
SRR2584863_2_fastqc/
                          SRR2584866_2_fastqc/
                                                     SRR2589044_2_fastqc/
SRR2584863_2_fastqc.html
                          SRR2584866_2_fastqc.html
                                                    SRR2589044_2_fastqc.html
SRR2584863_2_fastqc.zip
                          SRR2584866_2_fastqc.zip
                                                     SRR2589044_2_fastqc.zip
```

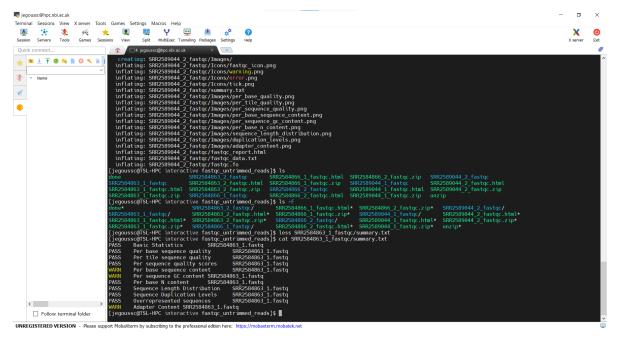
Let's see what files are present within one of these output directories.

```
$ ls -F SRR2584863_1_fastqc/
fastqc_data.txt fastqc.fo fastqc_report.html Icons/ Images/ summary.txt
```

Use less to preview the summary.txt file for this sample.

```
$ less SRR2584863 1 fastqc/summary.txt
        Basic Statistics
PASS
                                SRR2584863 1.fastq
PASS
        Per base sequence quality
                                         SRR2584863 1.fastq
PASS
        Per tile sequence quality
                                         SRR2584863_1.fastq
        Per sequence quality scores
PASS
                                         SRR2584863_1.fastq
        Per base sequence content
WARN
                                         SRR2584863_1.fastq
        Per sequence GC content SRR2584863_1.fastq
WARN
PASS
        Per base N content
                                 SRR2584863_1.fastq
        Sequence Length Distribution
PASS
                                         SRR2584863_1.fastq
        Sequence Duplication Levels
PASS
                                         SRR2584863_1.fastq
PASS
        Overrepresented sequences
                                         SRR2584863_1.fastq
WARN
        Adapter Content SRR2584863_1.fastq
```

The summary file gives us a list of tests that FastQC ran, and tells us whether this sample passed, failed, or is borderline (WARN). Remember, to quit from less you must type q.



## 2.4 Documenting your work

We can make a record of the results we obtained for all our samples

by concatenating all of our summary.txt files into a single file using the cat command. We will call this fastqc\_summaries.txt and move it to ~/dc\_workshop/docs.

```
$ mkdir -p ~/dc_workshop/docs/
$ cat */summary.txt > ~/dc_workshop/docs/fastqc_summaries.txt
```

#### Exercise

Which samples failed at least one of FastQC's quality tests? What test(s) did those samples fail?

#### Solution

We can get the list of all failed tests using grep.

```
$ cd ~/dc_workshop/docs
$ grep FAIL fastqc_summaries.txt
        Per base sequence quality
FAIL
                                         SRR2584863_2.fastq.gz
        Per tile sequence quality
FAIL
                                         SRR2584863_2.fastq.gz
FAIL
        Per base sequence content
                                         SRR2584863_2.fastq.gz
FAIL
        Per base sequence quality
                                         SRR2584866_1.fastq.gz
FAIL
        Per base sequence content
                                         SRR2584866_1.fastq.gz
FAIL
        Adapter Content SRR2584866 1.fastq.gz
FAIL
        Adapter Content SRR2584866_2.fastq.gz
        Adapter Content SRR2589044 1.fastq.gz
FAIL
FAIL
        Per base sequence quality
                                        SRR2589044_2.fastq.gz
        Per tile sequence quality
FAIL
                                        SRR2589044_2.fastq.gz
FAIL
        Per base sequence content
                                        SRR2589044_2.fastq.gz
FAIL
        Adapter Content SRR2589044_2.fastq.gz
```

### 2.5 Summary

### i Quality encodings vary

Although we have used a particular quality encoding system to demonstrate interpretation of read quality, different sequencing machines use different encoding systems. This means that, depending on which sequencer you use to generate your data, a # may not be an indicator of a poor quality base call.

This mainly relates to older Solexa/Illumina data, but it is essential that you know which sequencing platform was used to generate your data, so that you can tell your quality control program which encoding to use. If you choose the wrong encoding, you run the risk of throwing away good reads or (even worse) not throwing away bad reads!

### Same symbols but different meanings

Here we see > being used as a shell prompt, whereas > is also used to redirect output. Similarly, \$ is used as a shell prompt, but, as we saw earlier, it is also used to ask the shell to get the value of a variable.

If the shell prints > or \$ then it expects you to type something, and the symbol is a prompt.

If you type > or \$ yourself, it is an instruction from you that the shell should redirect output or get the value of a variable.

## ! Key points

- Quality encodings vary across sequencing platforms.
- for loops let you perform the same set of operations on multiple files with a single command.

# 3 Trimming and filtering

### Time

Teaching: 30 minExercises: 25 min

#### Question

• How can I get rid of sequence data that does not meet my quality standards?

### Objectives

- Clean FASTQ reads using Trimmomatic.
- Select and set multiple options for command-line bioinformatic tools.
- Write for loops with two variables.

## 3.1 Cleaning reads

In the previous episode, we took a high-level look at the quality of each of our samples using FastQC. We visualized per-base quality graphs showing the distribution of read quality at each base across all reads in a sample and extracted information about which samples fail which quality checks. Some of our samples failed quite a few quality metrics used by FastQC. This does not mean, though, that our samples should be thrown out! It is very common to have some quality metrics fail, and this may or may not be a problem for your downstream application. For our variant calling workflow, we will be removing some of the low quality sequences to reduce our false positive rate due to sequencing error.

We will use a program called Trimmomatic (Bolger, Lohse, and Usadel 2014) to filter poor quality reads and trim poor quality bases from our samples.

## Important

Make sure to be connected to the server.

Let's load Trimmomatic to our interactive session on the server.

### 3.1.1 Trimmomatic options

Trimmomatic has a variety of options to trim your reads. If we run the following command, we can see some of our options.

#### \$ trimmomatic

Which will give you the following output:

```
Usage:

PE [-version] [-threads <threads>] [-phred33|-phred64] [-trimlog <trimLogFile>] [-sum or:

SE [-version] [-threads <threads>] [-phred33|-phred64] [-trimlog <trimLogFile>] [-sum
```

-version

or:

This output shows us that we must first specify whether we have paired end (PE) or single end (SE) reads. Next, we specify what flag we would like to run. For example, you can specify threads to indicate the number of processors on your computer that you want Trimmomatic to use. In most cases using multiple threads (processors) can help to run the trimming faster. These flags are not necessary, but they can give you more control over the command. The flags are followed by positional arguments, meaning the order in which you specify them is important. In paired end mode, Trimmomatic expects the two input files, and then the names of the output files. These files are described below. While, in single end mode, Trimmomatic will expect 1 file as input, after which you can enter the optional settings and lastly the name of the output file.

option	meaning	
<inputfile1></inputfile1>	Input reads to be trimmed. Typically the file name will contain an _1 or _R1 in the name.	
<inputfile2></inputfile2>	Input reads to be trimmed. Typically the file name will contain an _2 or _R2 in the name.	
<outputfile1p< td=""><td>&gt;Output file that contains surviving pairs from the _1 file.</td></outputfile1p<>	>Output file that contains surviving pairs from the _1 file.	
<pre><outputfile1u>Output file that contains orphaned reads from the _1 file.</outputfile1u></pre>		
<pre><outputfile2p>Output file that contains surviving pairs from the _2 file.</outputfile2p></pre>		
<outputfile2u< td=""><td>&gt;Output file that contains or phaned reads from the _2 file.</td></outputfile2u<>	>Output file that contains or phaned reads from the _2 file.	

The last thing Trimmomatic expects to see is the trimming parameters:

step	meaning		
ILLUMINACLIP Perform adapter removal.			
SLIDINGWIND	Owerform sliding window trimming, cutting once the average quality within the		
	window falls below a threshold.		
LEADING	Cut bases off the start of a read, if below a threshold quality.		
TRAILING	Cut bases off the end of a read, if below a threshold quality.		
CROP	Cut the read to a specified length.		
HEADCROP	Cut the specified number of bases from the start of the read.		
MINLEN	Drop an entire read if it is below a specified length.		
TOPHRED33	Convert quality scores to Phred-33.		
TOPHRED64	Convert quality scores to Phred-64.		

We will use only a few of these options and trimming steps in our analysis. It is important to understand the steps you are using to clean your data. For more information about the Trimmomatic arguments and options, see the Trimmomatic manual.

### 3.1.2 Running Trimmomatic

Now we will run Trimmomatic on our data. To begin, navigate to your untrimmed\_fastq data directory:

```
$ cd ~/dc_workshop/data/untrimmed_fastq
```

We are going to run Trimmomatic on one of our paired-end samples. While using FastQC we saw that Nextera adapters were present in our samples. The adapter sequences came with the installation of Trimmomatic.

We will also use a sliding window of size 4 that will remove bases if their phred score is below 20 (like in our example above). We will also discard any reads that do not have at least 25 bases remaining after this trimming step. Three additional pieces of code are also added to the end of the ILLUMINACLIP step. These three additional numbers (2:40:15) tell Trimmomatic how to handle sequence matches to the Nextera adapters. A detailed explanation of how they work is advanced for this particular lesson. For now we will use these numbers as a default and recognize they are needed to for Trimmomatic to run properly.

### MINLEN:25 \ ILLUMINACLIP:NexteraPE-PE.fa:2:40:15

This command will take about two minutes to run.

### About multiline commands

Some of the commands we ran in this lesson are long! When typing a long command into your terminal, you can use the \ character to separate code chunks onto separate lines. This can make your code more readable.

code	meaning	
PE	that it will be taking a paired end file as input	
-threads 4	to use four computing threads to run (this will speed up our run)	
SRR2589044_1.fastq	the first input file name	
SRR2589044_2.fastq	the second input file name	
SRR2589044_1.trimmed.hfassttput file for surviving pairs from the _1 file		
SRR2589044_1un.trimmted.fatstoot file for orphaned reads from the _1 file		
SRR2589044_2.trimmedhfassttput file for surviving pairs from the _2 file		
SRR2589044_2un.trimmted.ofatstot file for orphaned reads from the _2 file		
ILLUMINACLIP: SRR_adaptelips the Illumina adapters from the input file using the adapter		
	sequences listed in SRR_adapters.fa	
SLIDINGWINDOW:4:20	to use a sliding window of size 4 that will remove bases if their phred	
	score is below 20	
MINLEN:25	minimum length of 25 nucleotides.	

The output should look like this:

```
TrimmomaticPE: Started with arguments:
 SRR2589044_1.fastq.gz SRR2589044_2.fastq.gz SRR2589044_1.trim.fastq.gz SRR2589044_1un.trim.
Multiple cores found: Using 2 threads
Using PrefixPair: 'AGATGTGTATAAGAGACAG' and 'AGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTCCGAGCCCACGAGAC'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTGACGCTGCCGACGA'
ILLUMINACLIP: Using 1 prefix pairs, 4 forward/reverse sequences, 0 forward only sequences, 0
Quality encoding detected as phred33
```

Input Read Pairs: 1107090 Both Surviving: 885220 (79.96%) Forward Only Surviving: 216472 (19

TrimmomaticPE: Completed successfully

#### Exercise

Use the output from your Trimmomatic command to answer the following questions.

- 1) What percent of reads did we discard from our sample?
- 2) What percent of reads did we keep both pairs?

#### Solution

- 1) 0.23%
- 2) 79.96%

You may have noticed that Trimmomatic automatically detected the quality encoding of our sample. It is always a good idea to double-check this or to enter the quality encoding manually.

We can confirm that we have our output files:

The output files are also FASTQ files. It should be smaller than our input file, because we have removed reads. We can confirm this:

```
$ ls SRR2589044* -l -h
-rwx----- 1 [username] TSL_20 124M Aug 11 11:00 SRR2589044_1.fastq.gz
-rwx----- 1 [username] TSL_20 105M Aug 11 15:56 SRR2589044_1.trim.fastq.gz
-rwx----- 1 [username] TSL_20 8.4M Aug 11 15:56 SRR2589044_1un.trim.fastq.gz
-rwx----- 1 [username] TSL_20 128M Aug 11 11:01 SRR2589044_2.fastq.gz
-rwx----- 1 [username] TSL_20 103M Aug 11 15:56 SRR2589044_2.trim.fastq.gz
-rwx----- 1 [username] TSL_20 276K Aug 11 15:56 SRR2589044_2un.trim.fastq.gz
```

We have just successfully run Trimmomatic on one of our FASTQ files! However, there is some bad news. Trimmomatic can only operate on one sample at a time and we have more than one sample. The good news is that we can use a for loop to iterate through our sample files quickly!

We unzipped one of our files before to work with it, let's compress it again before we run our for loop.

```
$ gzip SRR2584863_1.fastq
```

### ⚠ Warning

Do not copy-paste the four lines at once. You must type the for loop line by line!

Go ahead and run the for loop. It should take about 12 minutes for Trimmomatic to run for each of our six input files.

Once it is done running, take a look at your directory contents.

```
$ 1s
SRR2584863_1.fastq.gz
                              SRR2584866_2.fastq.gz
SRR2584863_1.trim.fastq.gz
                              SRR2584866_2.trim.fastq.gz
SRR2584863_1un.trim.fastq.gz
                             SRR2584866_2un.trim.fastq.gz
SRR2584863_2.fastq.gz
                              SRR2589044_1.fastq.gz
SRR2584863_2.trim.fastq.gz
                              SRR2589044_1.trim.fastq.gz
SRR2584863_2un.trim.fastq.gz SRR2589044_1un.trim.fastq.gz
SRR2584866_1.fastq.gz
                              SRR2589044_2.fastq.gz
SRR2584866_1.trim.fastq.gz
                              SRR2589044 2.trim.fastq.gz
SRR2584866_1un.trim.fastq.gz SRR2589044_2un.trim.fastq.gz
```

#### Note

You will notice that even though we ran Trimmomatic on file SRR2589044 before running the for loop, there is only one set of files for it. Because we matched the ending <code>\_1.fastq.gz</code>, we re-ran Trimmomatic on this file, overwriting our first results. That is ok, but it is good to be aware that it happened.

We have now completed the trimming and filtering steps of our quality control process! Before we move on, let's move our trimmed FASTQ files to a new subdirectory within our data/directory.

```
$ cd ~/dc_workshop/data/untrimmed_fastq
$ mkdir ../trimmed_fastq
$ mv *.trim* ../trimmed_fastq
$ cd ../trimmed_fastq
$ 1s
SRR2584863 1.trim.fastq.gz
                              SRR2584866 1.trim.fastq.gz
                                                            SRR2589044 1.trim.fastq.gz
SRR2584863 1un.trim.fastq.gz
                              SRR2584866 1un.trim.fastq.gz
                                                            SRR2589044 1un.trim.fastq.gz
SRR2584863_2.trim.fastq.gz
                              SRR2584866_2.trim.fastq.gz
                                                            SRR2589044_2.trim.fastq.gz
SRR2584863 2un.trim.fastq.gz
                              SRR2584866_2un.trim.fastq.gz
                                                            SRR2589044_2un.trim.fastq.gz
```

#### Bonus exercise

Now that our samples have gone through quality control, they should perform better on the quality tests run by FastQC. Go ahead and re-run FastQC on your trimmed FASTQ files and visualize the HTML files to see whether your per base sequence quality is higher after trimming.

#### Solution

```
$ fastqc ~/dc_workshop/data/trimmed_fastq/*.fastq*
```

Then output files can be transferred to the local computer to be visualized. After trimming and filtering, our overall quality is much higher, we have a distribution of sequence lengths, and more samples pass adapter content. However, quality trimming is not perfect, and some programs are better at removing some sequences than others. Because our sequences still contain 3' adapters, it could be important to explore other trimming tools like Cutadapt (Martin 2011) to remove these, depending on your downstream application. Trimmomatic did pretty well though, and its performance is good enough for our workflow.

## 3.2 Summary

### Key points

- The options you set for the command-line tools you use are important!
- Data cleaning is an essential step in a genomics workflow.

# 4 Variant calling workflow

#### Time

Teaching: 35 minExercises: 25 min

#### Question

• How do I find sequence variants between my sample and a reference genome?

#### Objectives

- Understand the steps involved in variant calling.
- Describe the types of data formats encountered during variant calling.
- Use command line tools to perform variant calling.

We mentioned before that we are working with files from a long-term evolution study of an  $E.\ coli$  population (designated Ara-3). Now that we have looked at our data to make sure that it is high quality, and removed low-quality base calls, we can perform variant calling to see how the population changed over time. We care how this population changed relative to the original population,  $E.\ coli$  strain REL606. Therefore, we will align each of our samples to the  $E.\ coli$  REL606 reference genome, and see what differences exist in our reads versus the genome.

## 4.1 Alignment to a reference genome

We perform read alignment or mapping to determine where in the genome our reads originated from. There are a number of tools to choose from and, while there is no gold standard, there are some tools that are better suited for particular NGS analyses. We will be using the Burrows Wheeler Aligner (BWA) by Li and Durbin (2010), which is a software package for mapping low-divergent sequences against a large reference genome.

The alignment process consists of two steps:

1. Indexing the reference genome

2. Aligning the reads to the reference genome

## 4.2 Setting up

### 4.2.1 Option 1: Copy from shared directory on the HPC

You should already have all the files from the previous section where we copied all files we need including the reference genome and small dataset to your directory. Now:

Unzip the reference genome

```
$ gunzip ecoli_rel606.fasta.gz
```

#### Exercise

We saved this file as data/ref\_genome/ecoli\_rel606.fasta.gz and then decompressed it. What is the real name of the genome?

#### Solution

```
$ head data/ref_genome/ecoli_rel606.fasta
```

The name of the sequence follows the > character. The name is CP000819.1 Escherichia coli B str. REL606, complete genome. Keep this chromosome name (CP000819.1) in mind, as we will use it later in the lesson.

We will also have a set of trimmed FASTQ files to work with. These are small subsets of our real trimmed data, and will enable us to run our variant calling workflow quite quickly.

```
$ ls ~/dc_workshop/data/trimmed_fastq_small
```

## 4.3 Index the reference genome

Let's make sure we are back in the dc\_workshop directory and that the two directories are where we hope they are:

```
$ cd ~/dc_workshop/
$ ls -F data/
ref_genome/ trimmed_fastq_small/ untrimmed_fastq/
```

```
$ ls -F data/ref_genome/
ecoli_rel606.fasta*
```

You will also need to create directories for the results that will be generated as part of this workflow. We can do this in a single line of code, because mkdir can accept multiple new directory names as input.

```
$ mkdir -p results/sam results/bam results/bcf results/vcf
```

Our first step is to index the reference genome for use by BWA.

```
$ source package /nbi/software/production/bin/bwa-0.7.5
loaded /nbi/software/production/bin/bwa-0.7.5
```

Indexing allows the aligner to quickly find potential alignment sites for query sequences in a genome, which saves time during alignment. Indexing the reference only has to be run once. The only reason you would want to create a new index is if you are working with a different reference genome or you are using a different tool for alignment.

```
$ bwa index data/ref_genome/ecoli_rel606.fasta
```

While the index is created, you will see output that looks something like this:

```
[bwa_index] Pack FASTA... 0.15 sec
[bwa_index] Construct BWT for the packed sequence...
[bwa_index] 3.01 seconds elapse.
[bwa_index] Update BWT... 0.05 sec
[bwa_index] Pack forward-only FASTA... 0.05 sec
[bwa_index] Construct SA from BWT and Occ... 0.89 sec
[main] Version: 0.7.5-r404
[main] CMD: bwa index data/ref_genome/ecoli_rel606.fasta
[main] Real time: 4.348 sec; CPU: 4.160 sec
```

### i BWA alignment options

BWA consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. The first algorithm is designed for Illumina sequence reads up to 100bp, while the other two are for sequences ranging from 70bp to 1Mbp. BWA-MEM and BWA-SW share similar features such as long-read support and split alignment, but BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate.

## 4.4 Align reads to the reference genome

The alignment process consists of choosing an appropriate reference genome to map our reads against and then deciding on an aligner. We will use the BWA-MEM algorithm, which is the latest and is generally recommended for high-quality queries as it is faster and more accurate.

An example of what a bwa command looks like is below. This command will not run, as we do not have the files ref\_genome.fa, input\_file\_R1.fastq, or input\_file\_R2.fastq.

```
$ bwa mem ref_genome.fasta input_file_R1.fastq input_file_R2.fastq > output.sam
```

Have a look at the bwa options page. While we are running bwa with the default parameters here, your use case might require a change of parameters.



Always read the manual page for any tool before using and make sure the options you use are appropriate for your data.

We are going to start by aligning the reads from just one of the samples in our data set (SRR2584866). Later, we will be iterating this whole process on all of our sample files.

```
$ bwa mem data/ref_genome/ecoli_rel606.fasta \
  data/trimmed_fastq_small/SRR2584866_1.trim.sub.fastq \
  data/trimmed_fastq_small/SRR2584866_2.trim.sub.fastq > \
  results/sam/SRR2584866.aligned.sam
```

You will see output that starts like this:

```
[M::main_mem] read 77446 sequences (10000033 bp)...
[M::mem_pestat] # candidate unique pairs for (FF, FR, RF, RR): (48, 36779, 21, 61)
[M::mem_pestat] analyzing insert size distribution for orientation FF...
[M::mem_pestat] (25, 50, 75) percentile: (420, 660, 1774)
[M::mem_pestat] low and high boundaries for computing mean and std.dev: (1, 4482)
[M::mem_pestat] mean and std.dev: (784.68, 700.87)
[M::mem_pestat] low and high boundaries for proper pairs: (1, 5836)
[M::mem_pestat] analyzing insert size distribution for orientation FR...
[M::mem_pestat] (25, 50, 75) percentile: (221, 361, 576)
[M::mem_pestat] low and high boundaries for computing mean and std.dev: (1, 1286)
```

It should take less than a minute to be complete.