Genomics Tutorial 2019.

Instructors:

- Josie Paris & the Workshop Team!
- Konrad Paszkiewicz *in absentia* (don't worry he's not dead).

Important Notes

username: genomics
password: see the whiteboard

Objectives:

Part 1: Short Read Genomics: An Introduction

- Understand how short reads are generated.
- Understand paired-end reads
- See possible sources of errors
- Learn about adaptors

Part 2: QC, Alignment and Variant Calling

- Interpret FASTQ quality metrics
- Remove poor quality data
- Trim adaptor/contaminant sequences from FASTQ data
- Count the number of reads before and after trimming and quality control
- Align reads to a reference sequence to form a SAM file (Sequence AlignMent file) using BWA
- Convert the SAM file to BAM format (Binary AlignMent format)
- Identify and select high quality SNPs and Indels using SAMtools
- Identify missing or truncated genes with respect to the reference genome
- Identify SNPs which overlap with known coding regions

Part 3: Assembly of Unmapped Reads

- Extract reads which do not map to the reference sequence
- Assemble these reads de novo using SPAdes
- Generate summary statistics for the assembly
- Identify potential genes within the assembly
- Search for matches within the NCBI database via BLAST and against the Pfam database

- Visualize the taxonomic distribution of BLAST hits
- Perform gene prediction and annotation using RAST

Part 4: De-novo Assembly Using Short Reads

- Perform QC and adaptor-trim Illumina reads.
- Assemble these reads de novo using SPAdes
- Generate summary statistics for the assembly
- Understand how to incorporate long PacBio reads into the assembly.
- Identify open reading frames within the assembly
- Search for matches within the NCBI database via BLAST and against the Pfam database
- Visualize species distribution of potential matches

Table of Contents

Part 1: Short Read Genomics: An Introduction

1. Introduction

Welcome to the genomics tutorial! Generating large amounts of data in biology is easy these days. In little more than a fortnight we can generate more data than the entire human genome project generated in over a decade of work. Making biological sense out of that data, understanding its limitations and how the analysis algorithms work is now the major challenge for researchers. The aim of this workshop is to take you through an example project. On the way, you will learn how to evaluate the quality of data as provided by a sequencing facility, how to align the data against a known and annotated reference genome and how to perform a de-novo assembly. In addition you will also learn how to compare results between different samples.

This workshop is broken into 4 parts. You should feel free to take as long as you like on each part. It is much more important that you have a thorough understanding of each part, rather than try to race through the entire workshop material.

The four parts are:

1. Short Read Introduction
2. Remapping a strain of *E.coli* to a reference sequence
3. Assembly of unmapped reads
4. Complete *de-novo* assembly of all reads

For this tutorial we will assume little background knowledge, except for a basic familiarity with the Linux operating system and the cloud. We will cover the basics of how genomic DNA libraries are generated and sequenced, and the principles behind short read paired-end sequencing. We will look at why data can vary in quality, why adaptor sequences need to be filtered out and how to quality control data. You may well do similar tasks in other tutorials at this workshop, especially quality control and assembly techniques, this is good practice!

Then we will take the plunge and align the filtered reads to a reference genome, call variants and compare them against the published genome to identify missing, truncated or altered genes. This will involve the use of a publicly available set of bacterial *E.coli* Illumina reads and reference genome.

In parts 3 and 4 we will look at how one can identify novel sequences which are not present in the reference genome.

A word on notation. If you see something like this:

```
cd ~/genomics_tutorial/reference_sequence
```

It means, type the highlighted text into your terminal. Please type the text, using all the tricks (e.g. tab completion) that you have learnt in the Unix tutorial. Copying and pasting will sometimes not work with certain characters and can cause errors. Also, please keep an eye for underscores!

Principles of Illumina-based Sequencing

There are several sequencers currently on the market. These include PacBio, MinION and the various Illumina platforms (HiSeq, NextSeq, NovaSeq, MiSeq etc). Other (now obsolescent) platforms included Life Tech SoLID and Roche 454 and many more are likely to appear in the future! Regardless of the sequencer, all of these rely on making hundreds of thousands of clonal copies of a fragment of DNA and sequencing the ensemble of fragments using DNA polymerase or in the case of the SOLiD via ligation. This is simply because the detectors (basically souped-up digital cameras), cannot detect fluorescence (Illumina, SolID, 454) or pH changes (Ion Torrent) from a single molecule. The 'third-generation' Pacific Biosciences SMRT (Single Molecule Real Time) RSII and Sequel sequencers are able to detect fluorescence from a single molecule of DNA. However, the machines

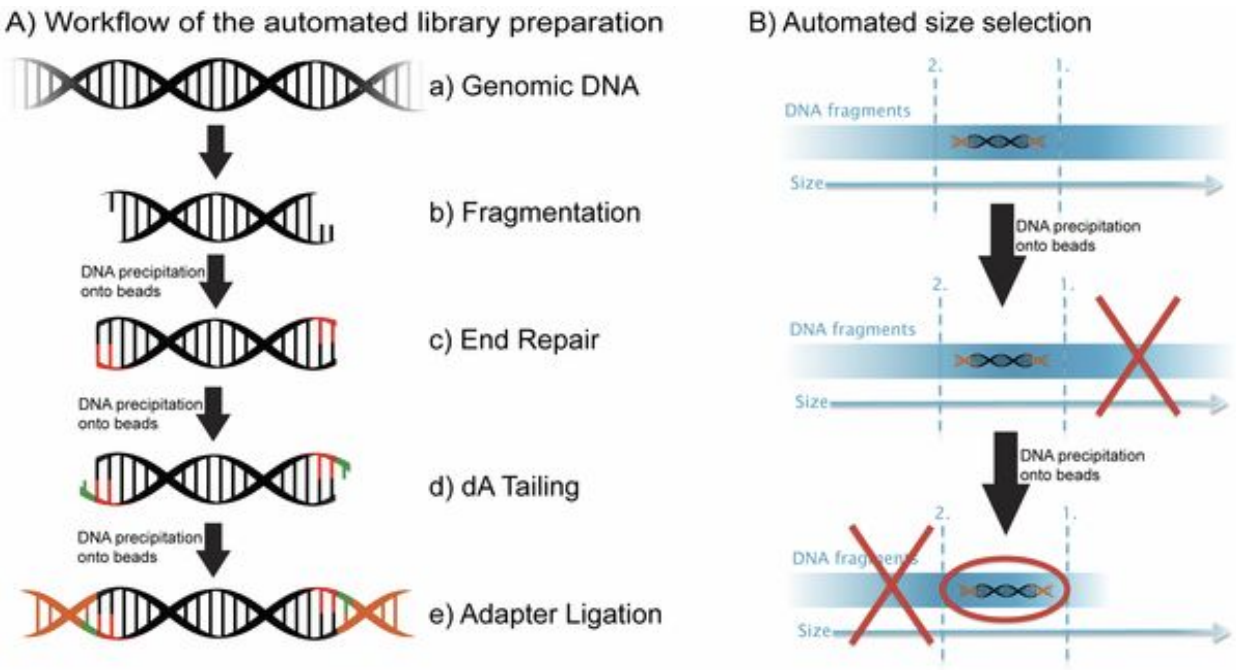
are very large (the RSII is almost 2 tons) and produces less than a tenth of the data of an Illumina MiSeq run and for long reads >10kb error rates are generally around 10-12%. The Oxford Nanopore MinION is another ‘third-generation’ single-molecule system which measures changes in electrical current through a Nanopore as a single molecule is ratcheted through it. Although error rates are also high (5-10%), and per-base costs are higher, the technology has improved rapidly and will probably replace second generation systems over the next few years.

We will mainly look at the Illumina sequencing pipeline here, but the basic principles apply to other second-generation sequencers. If you would like further details on other platforms then we recommend reading: ***Mardis ER. Next-generation DNA sequencing methods. Annual Reviews Genomics Hum Genet 2008; 9 :387–402.***

A typical sequencing run would begin with the user supplying 1ng-1ug of genomic DNA to a sequencing facility along with quality control information in the form of an automated electrophoresis output (e.g. Agilent Bioanalyser/Tapestation trace) or gel image and quantification information.

DNA Library Preparation

For most sequencing applications, paired-end libraries are generated. Genomic DNA is sheared into 300-500bp fragments (usually via sonication) and size-selected accordingly. Ends are repaired and an overhanging adenine base is added, after which oligonucleotide adaptors are ligated. In many cases the adaptors contain unique DNA sequences of 6-12bp which can be used to identify the sample if they are 'multiplexed' together for sequencing. This type of sequencing is used extensively when sequencing small genomes such as those of bacteria because it lowers the overall per-genome cost.



A) Steps a through e explain the main steps in Illumina sample preparation: a) the initial genomic DNA, b) fragmentation of genomic DNA into 500bp fragments, c) end repair, d) addition of A bases to the fragment ends and e) ligation of the adaptors to the fragments.

B) Overview of the automated the size selection protocol: The first precipitation discards fragments larger than the desired interval. The second precipitation selects all fragments larger than the lower boundary of the desired interval.

Borgström E, Lundin S, Lundeberg J, 2011 Large Scale Library Generation for High Throughput Sequencing. PLoS ONE 6: e19119. doi:10.1371/journal.pone.0019119

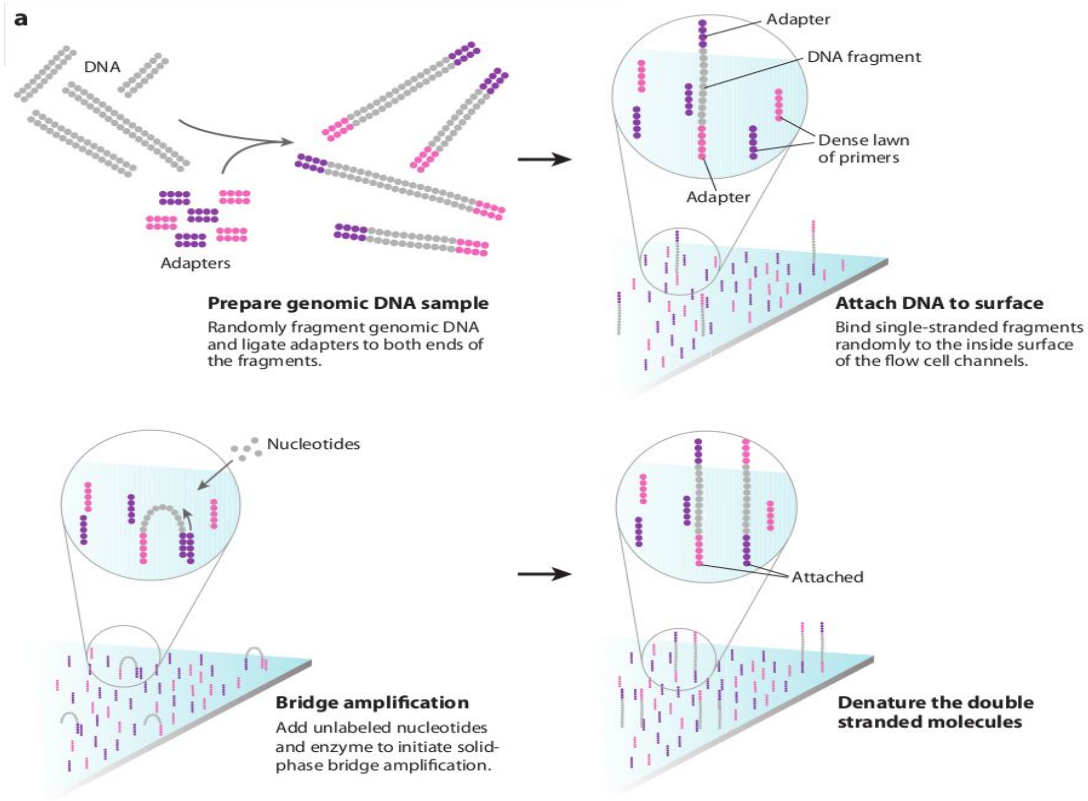
Sequencing

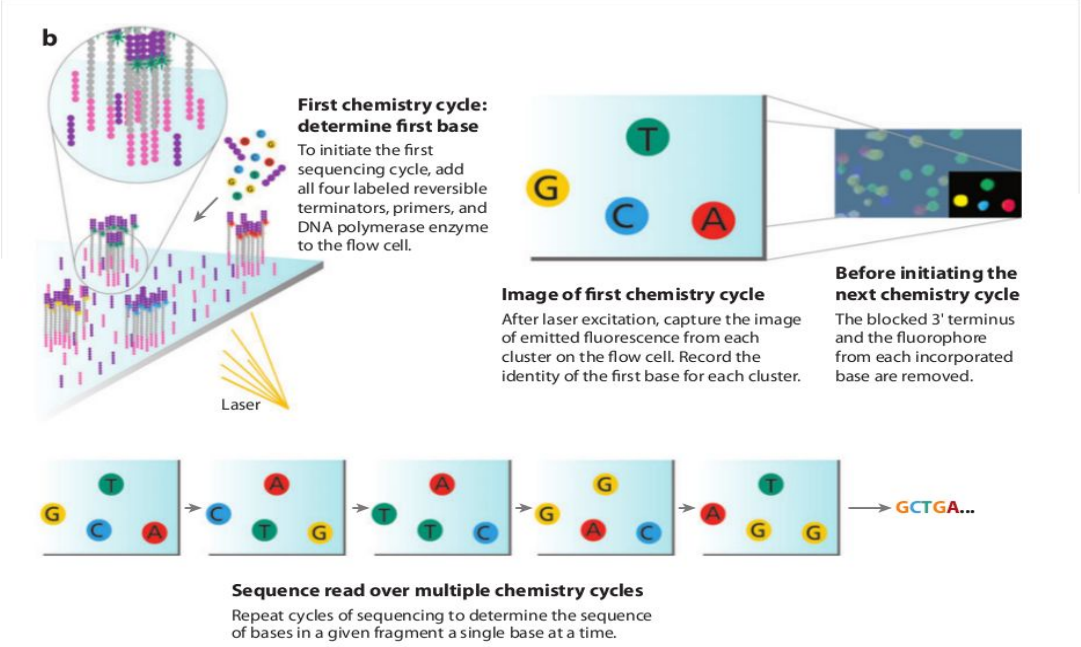
(adapted from Margulis, E.R., reference below)

Once sufficient libraries have been prepared, the task is to amplify single strands of DNA to form monoclonal clusters. The single molecule amplification step for the Illumina HiSeq 2500 starts with an Illumina-specific adapter library and takes place on the oligo-derivatized surface of a flow cell, and is performed by an automated device called a cBot Cluster Station. The flow cell is either a 2 or 8-channel sealed glass microfabricated device that allows bridge amplification of fragments on its surface, and uses DNA polymerase to produce multiple DNA copies, or clusters, that each represent the single molecule that initiated the cluster amplification.

Separate or multiple libraries can be added to each of the eight channels, or the same library can be used in all eight, or combinations thereof. Each cluster contains approximately one million copies of the original fragment, which is sufficient for reporting incorporated bases at the required signal intensity for detection during sequencing. The Illumina system utilizes a sequencing- by-synthesis approach in which all four nucleotides are added simultaneously to the flow cell channels, along with DNA polymerase, for incorporation into the oligo-primed cluster fragments (see figure below for details). Specifically, the nucleotides carry a base-unique fluorescent label and the 3 -OH group is chemically blocked such that each incorporation is a unique event. An imaging step follows each base incorporation step, during which each flow cell lane is imaged in three 100-tile segments by the instrument optics at a cluster density of 600,000-800,000 per mm^2. After each imaging step, the 3' blocking group is chemically removed to prepare each strand for the next incorporation by DNA polymerase. This series of steps continues for a specific number of cycles, as determined by user-defined instrument settings, which permits discrete read lengths of 40–300 bases. A base-calling algorithm assigns sequences and associated quality values to each read and a quality checking pipeline evaluates the Illumina data from each run.

The next figures summarise the process:


a
DNA
Adapters
Prepare genomic DNA sample
Randomly fragment genomic DNA and ligate adapters to both ends of the fragments.
Adapter
DNA fragment
Dense lawn of primers
Adapter
Attach DNA to surface
Bind single-stranded fragments randomly to the inside surface of the flow cell channels.
Nucleotides
Bridge amplification
Add unlabeled nucleotides and enzyme to initiate solid-phase bridge amplification.
Attached
Denature the double stranded molecules



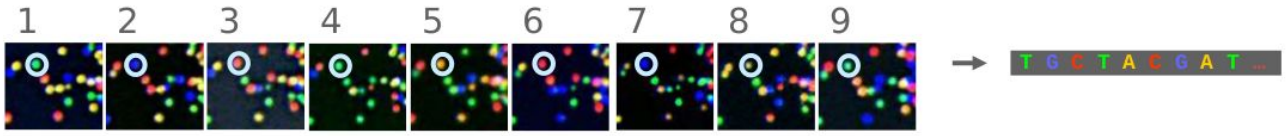
The Illumina sequencing-by-synthesis approach: Cluster strands created by bridge amplification are primed and all four fluorescently labelled, 3 -OH blocked nucleotides are added to the flow cell with DNA polymerase. The cluster strands are extended by one nucleotide. Following the incorporation step, the unused nucleotides and DNA polymerase molecules are washed away, a scan buffer is added to the flow cell, and the optics system scans each lane of the flow cell by imaging units called tiles. Once imaging is completed, chemicals that affect cleavage of the fluorescent labels and the 3 -OH blocking groups are added to the flow cell, which prepares the cluster strands for another round of fluorescent nucleotide incorporation. Next-Generation DNA Sequencing Methods Mardis, E.R. Annu. Rev. Genomics Hum. Genet. 2008. 9:387–402

A short movie of the Illumina sequencing-by-synthesis approach can be found here: https://www.youtube.com/watch?v=fCd6B5HRaZ8

Base-calling

Base-calling involves evaluating the raw intensity values for each fluorophore and comparing them to determine which base is actually present at a given position during a cycle. To call bases on the Illumina platform, the positions of clusters need to be identified during the first few cycles. This is because they are formed in random positions on the flowcell as the annealing process is stochastic.

If there are too many clusters, the edges of the clusters will begin to merge and the image analysis algorithms will not be able to distinguish one cluster from another (remember, the software is dealing with upwards of half a million clusters per square millimeter – that's a lot of dots!).



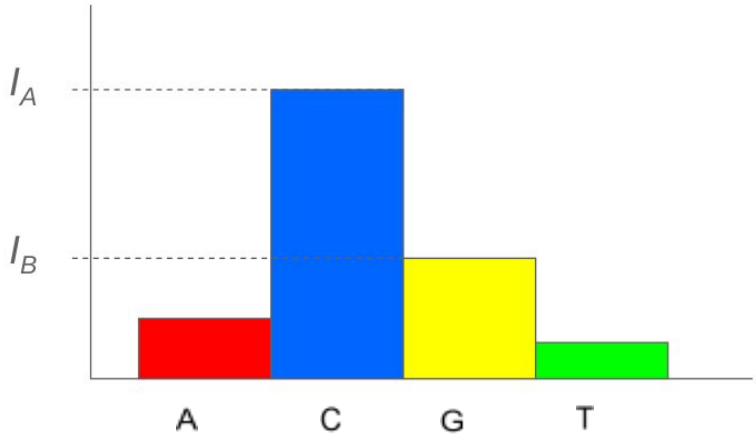
The above figure illustrates the principles of base-calling from cycles 1 to 9. If we focus on the highlighted cluster, one can observe that the colour (wavelength) of light observed at each cycle changes along with the brightness (intensity). This is due to the incorporation of complementary ddNTPs containing fluorophores. So at cycle 1 we have a T base, at 2 a G base and so on. If the colour or intensity is ambiguous the sequencer will mark it as an N. Other clusters are also visible in the images; these will represent different monoclonal clusters with different sequences.

The base calling algorithms turn the raw intensity values into T,G,C,A or N base calls. There are a variety of methods to do this and the one mentioned here is by no means the only one available, but it is often used as the default method on the Illumina systems. Known as the 'Chastity filter' it will only call a base if the intensity divided by the sum of the highest and second highest intensity is less than a given threshold (usually 0.6). Otherwise the base is marked with an N. In addition the standard Illumina pipeline will reject an entire read if two or more of these failures occur in the first 4 bases of a read (it uses these cycles to determine the boundary of a cluster).

Note that these processes are carried out at the sequencing facility and you will not need to perform any of these tasks under normal circumstances. They are explained here as useful background information.

CHASTITY formula:

$$C = \frac{I_A}{I_A + I_B}$$

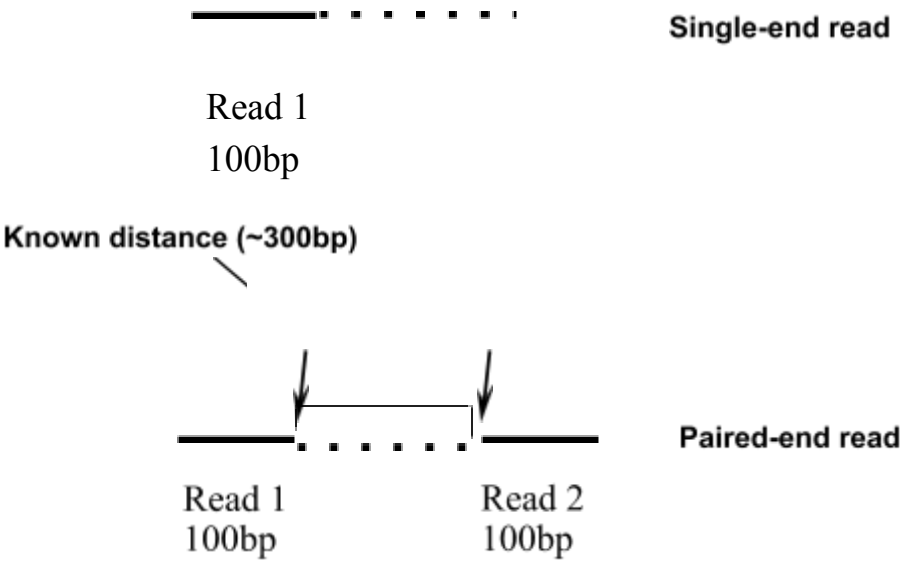


What are paired-end reads and why are they necessary?

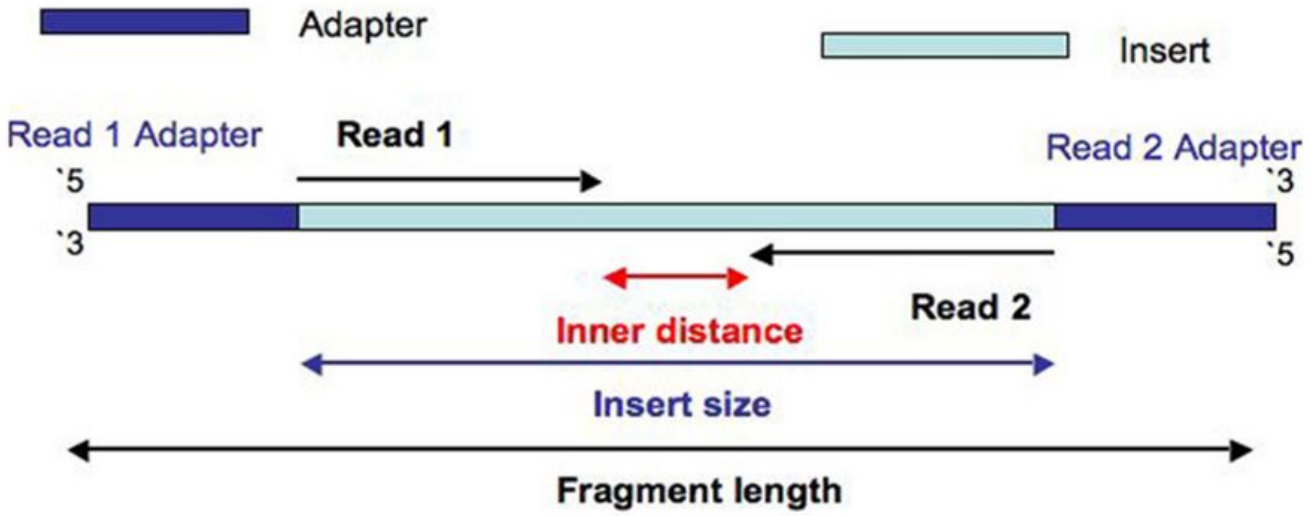
Paired-end sequencing is a remarkably simple and powerful modification to the standard sequencing protocol. It is nearly always worth obtaining paired-end reads if performing genomic sequencing. Typically sequencers of any type are only able to sequence a portion of DNA (e.g. 100bp

in the case of Illumina) before the fidelity of the enzyme and de-phasing of clusters (see later) increase the error rate beyond tolerable levels. As a result, on the Illumina system, a fragment which is 500bp long will only have the first 100bp sequenced.

If the size selection is tight enough and you know that nearly all the fragments are close to 500bp long, you can repeat the sequencing reaction from the other end of the fragment. This will yield two reads for each DNA fragment separated by a known distance. In the figure below the dashed regions represent the complete DNA fragment and the solid lines the regions we are able to sequence:



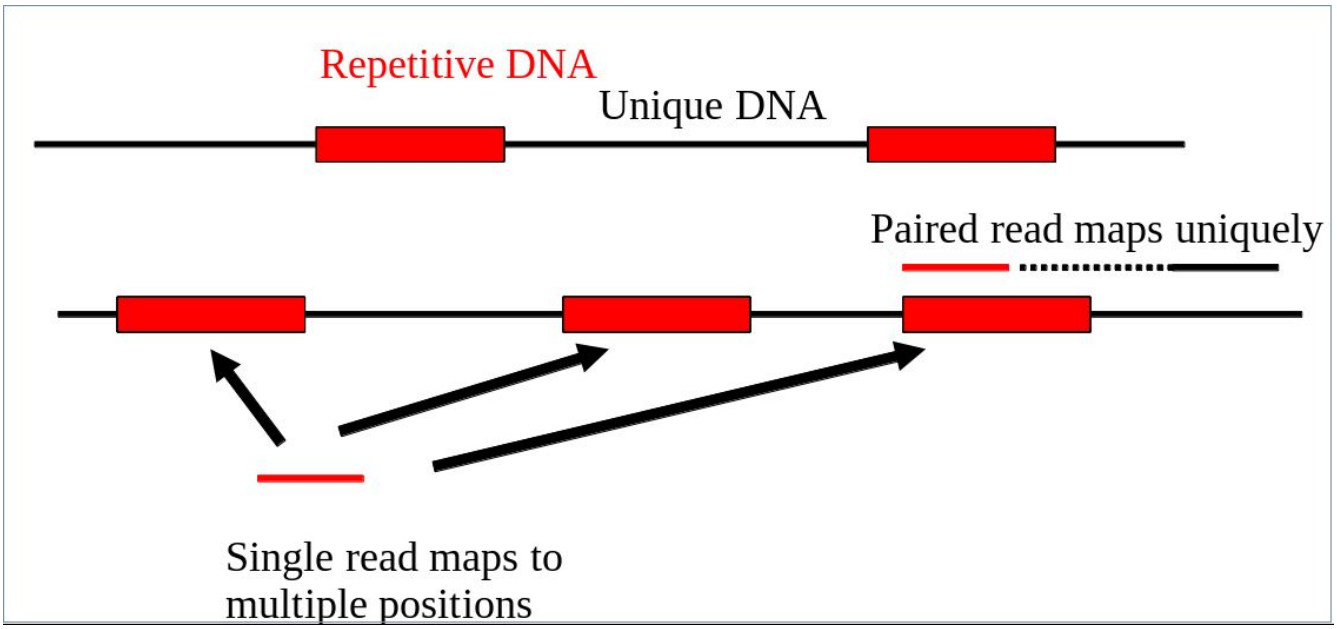
In the diagram below you can see a description of the nomenclature used when talking about paired end reads



The added information gained by knowing the distance between the two reads can be invaluable for spanning repetitive regions. In the figure below, the light coloured regions indicate repetitive sections of DNA. If a read contains only repetitive DNA, an alignment algorithm will be able to map the read to many locations in a reference genome. However, with paired-end reads, there is a greater chance that

at least one of the two reads will map to a unique region of DNA. In this way one of the reads can be used to anchor the other read in the pair and help resolve the repetitive region. Paired-end reads are often used when performing de-novo genome sequencing (i.e. when a reference is not available to align against) because they enable contiguous regions of DNA to be ordered, or when characterizing variants such as large insertions or deletions.

Other forms of paired-end sequencing with much larger distances (e.g. 10kb) are possible with so called 'mate-pair' libraries. These are usually used in specific projects to help order contigs in de-novo sequencing projects. We will not cover them here, but the principles behind them are similar.

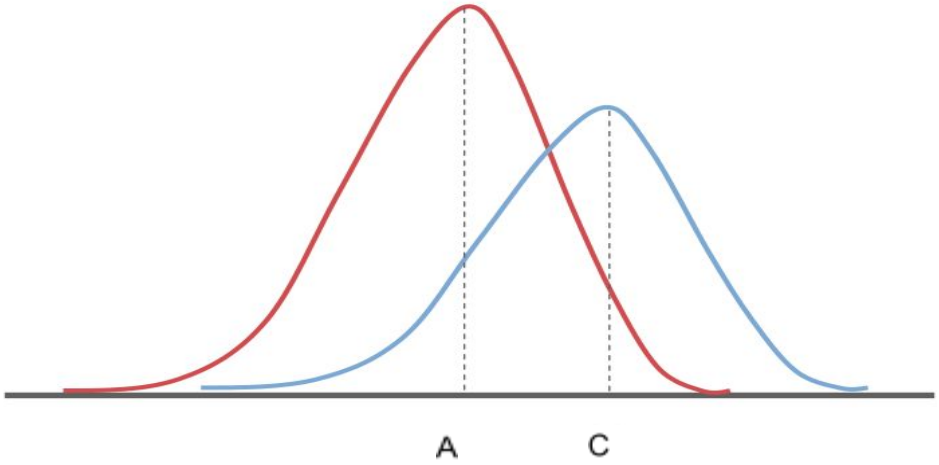


Inherent Sources of Error

No measurement is without a certain degree of error. This is true in sequencing. As such there is a finite probability that a base will not be called correctly. There are several possible sources:

Frequency Cross-talk and Normalisation Errors

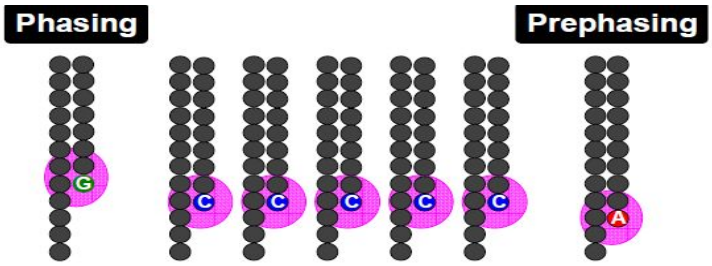
When reading an A base, a small amount of C will also be measured due to frequency overlap and vice-versa. Similarly with G and T bases. Additionally, from the figure below, it should be clear that the extent to which the dyes fluoresce differs. As such it is necessary to normalize the intensities. This normalisation process can also introduce errors.



Frequency response curve for A and C dyes
(Intensity y-axis and frequency on the x-axis)

Phasing/Pre-phasing

This occurs when a strand of DNA lags or leads the other DNA strands within a cluster. This introduces additional background noise into the signal and reduces the intensity of the true base. In the example below we have a cluster with 7 strands of DNA (very small, but this is just an example). Five strands are on a C-base, whilst 1 is lagging behind (called phasing) on a G base and the remaining strand is running ahead of the pack (confusingly called pre-phasing) on an A base. As such the C signal will be reduced and A and G boosted for the rest of the sequencing run. Too much phasing or pre-phasing (i.e. > 15-20%) usually causes problems for the base calling algorithm and result in clusters being filtered out.



Other issues:

- **Biases introduced by sample preparation –** your sequencing is only as good as your experimental design and DNA extraction. Also, remember that sometimes samples will be put through several cycles of PCR before sequencing (unless they are PCR-free libraries). This also introduces a potential source of bias.
- **High AT or GC content sequences** – this reduces the complexity of the sequence and can result in higher error rates.
- **Homopolymeric sequences** – long stretches of a single base can make it difficult to determine phasing and pre-phasing rates. This can introduce errors in determining the precise length of a hompolymeric stretch of sequence. (This much more of a problem on the old 454 and Ion Torrent than Illumina platforms but still worth bearing in mind). Especially if you encounter indels which have been called in homopolymeric tracts.
- Some motifs can cause loops and other steric clashes.

*See **Nakamura et al, Sequence-specific error profile of Illumina sequencers** **Nuc. Acid Res. first published online May 16, 2011 doi:10.1093/nar/gkr344***

Reads Containing Adaptors

Some reads will contain adaptor sequences after sequencing, usually at the end of the read. This is usually because of short sample DNA fragments, which result in the polymerase reading into the adaptor region. Occasionally this can also happen because of mis-priming. It is important to remove or trim sequences containing these reads as the adaptor sequences can prevent reads mapping to a reference sequence and will adversely affect de-novo assembly.

Part 2: QC, Alignment and Variant Calling

Introduction

In this section of the workshop we will be analysing a strain of *E.coli* which was sequenced at the Exeter Sequencing Service. It is closely related to the K-12 substrain MG1655 (http://www.ncbi.nlm.nih.gov/nuccore/U00096). We want to obtain a list of single nucleotide polymorphisms (SNPs), insertions/deletions (indels) and any genes which have been deleted.

Quality Control

In this section of the workshop we will be learning about evaluating the quality of an Illumina MiSeq sequencing run. The process described here can be used with any FASTQ formatted file from any platform (e.g. Illumina, PacBio etc).

Sequencers produce vast quantities of data. A single Illumina MiSeq lane can produce up to 15 Gigabases (Gbp) of data. However, the error rates of these platforms are 10-100x higher than Sanger sequencing. They also have very different error profiles. Unlike Sanger sequencing, where the most reliable sequences tend to be in the middle, NGS platforms tend to be most reliable near the beginning of each read.

Quality control usually involves:

- Calculating the number of reads before quality control
- Calculating GC content, identifying overrepresented sequences
- Remove or trim reads containing adaptor sequences
- Remove or trim reads containing low quality bases
- Calculating the number of reads after quality control
- Rechecking GC content, identifying overrepresented sequences

Quality control is necessary because:

- CPU time required for alignment and assembly is reduced

- Data storage requirements are reduced
- Reduce potential for bias in variant calling and/or de-novo assembly

Quality scores

To account for the possible errors and provide an estimate of confidence in a given base-call, the Illumina sequencing pipeline assigns a quality score to each base called. Nowadays, all quality scores are calculated using the Phred scale (*Ewing B, Green P: Basecalling of automated sequencer traces using phred. II. Error probabilities. Genome Research 8:186-194 (1998)*). Each base call has an associated base call quality which estimates chance that the base call is incorrect.

Q10 = 1 in 10 chance of incorrect base call
Q20 = 1 in 100 chance of incorrect base call
Q30 = 1 in 1000 chance of incorrect base call
Q40 = 1 in 10,000 chance of incorrect base call

For most Illumina runs you should see quality scores between Q20 and Q40.
Note that these as only estimates of base-quality based on calibration runs performed by the manufacturer against a sample of known sequence with (typically) a GC content of 50%. Extreme GC biases and/or particular motifs or homopolymers can cause the quality scores to become unreliable. Accurate base qualities are an essential part in ensuring variant calls are correct. As a rough and ready rule we generally assume that with Illumina data anything less than Q20 is not useful data and should be excluded.

Once you understand the FASTQ format try to work out what is happening to the quality scores here and why:

FASTQ Format

A FASTQ entry consists of 4 lines

```
@D3P26HQ1:110:d0eh1acxx:8:1101:1116:2122 1:N:0:
AGGTGTCTCCTACAACCAAAGCTACAACAGAGCAATGGGCTATCTGGTGGGATTTAAAGGGGTGAAAATGCATCCCCCTTAAAATNAAAGTGGTTTT
+
ADDADCFHHHDHGHIII<GIICH4FGCIHIEGFHGHGIIIGDHFDFG?DEHH>FGIG=E@GGADDDCCCCC@A>ABB>BBC:A>A#,228(4>:??B
```

1. A header line beginning with '@' containing information about the name of the sequencer, and the position at which the originating cluster was located and whether it passed purity filters.
2. The DNA sequence of the read
3. A header line or line beginning with just '+'
4. Quality scores for each base encoded in ASCII format

To reduce storage requirements, the FASTQ quality scores are stored as single characters and converted to numbers by obtaining the ASCII quality score and subtracting either 33 or 64. For example, the above FASTQ file is Sanger formatted and the character '!' has an ASCII value of 33. Therefore the corresponding base would have a Phred quality score of 33-33=Q0 (i.e. totally unreliable). On the other hand a base with a quality score denoted by '@' which has an ASCII value of 64 would have a Phred quality score of 64-33=Q31 (i.e. less than 1/1000 chance of being incorrect).

Just to confuse matters, there are several different methods of encoding quality scores in the ASCII format. Although as of 2011, Illumina 1.8+/Phred+33 is used universally (and most likely, this will not change in the future).

```
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.....................................
.........

.........................XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.............
.........
  ..............................IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII.....................
  ..................................JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ.....................
 LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL.....................................................
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
 |                         |    |        |                              |                     |
33                        59   64       73                            104                   126
 0........................26...31.......40
                           -5....0........9.............................40
                                0........9.............................40
                                   3.....9.............................40
 0.2......................26...31.........41

 S - Sanger        Phred+33,  raw reads typically (0, 40)
 X - Solexa        Solexa+64, raw reads typically (-5, 40)
 I - Illumina 1.3+ Phred+64,  raw reads typically (0, 40)
 J - Illumina 1.5+ Phred+64,  raw reads typically (3, 40)
     with 0=unused, 1=unused, 2=Read Segment Quality Control Indicator (bold)
     (Note: See discussion above).
 L - Illumina 1.8+ Phred+33,  raw reads typically (0, 41)
```

Note that the latest Illumina CASAVA 1.8 pipeline (released June 2011), outputs in fastq-sanger rather than Illumina 1.3+. Thus Illumina 1.3+ and other Illumina scoring metrics are unlikely to be encountered if you are using Illumina sequencing data generated after July 2011.

Quality control – Evaluating the Quality of Illumina Data

The first task when one receives sequencing data is to evaluate its quality and determine whether all the cash you have handed over was well-spent! To do this we will use the FastQC toolkit (https://www.bioinformatics.babraham.ac.uk/projects/fastqc/). FastQC offers a graphical visualisation of QC metrics, but *does not* have the ability to filter data.

Task 1

Open a terminal window. From your home directory change into: workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/ directory and list the directory contents, e.g.

```
cd ~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/
ls -l
```

***Note that you will also see two other directories here as well: blast_precompute and denovo_assembly. Don't worry about these directories for now as we will come back to them later in the tutorial.

For the purposes of this tutorial, we have already cleaned the data so you will see four files

Raw reads:

- read 1 (E_Coli_CGATGT_L001_R1_001.fastq)
- read 2 (E_Coli_CGATGT_L001_R2_001.fastq)

Cleaned reads:

- read 1 (E_Coli_CGATGT_L001_R1_001.filtered.fastq)
- read 2 (E_Coli_CGATGT_L001_R2_001.filtered.fastq)

These are paired-end data and so reads from the same pair can be identified because they will have the same header. Many programs require that the read 1 and read 2 files have the reads in the same order. We will look at the raw reads. To view the first few headers we can use the head and grep commands:

```
head E_Coli_CGATGT_L001_R1_001.fastq | grep MISEQ
head E_Coli_CGATGT_L001_R2_001.fastq | grep MISEQ
```

```
[ec2-user@ip-10-169-87-62 ecoli_exeter]$ head E_Coli_CGATGT_L001_R1_001.fastq | grep MISEQ
@MISEQ:8:000000000-A7VC1:1:1101:14839:1482 1:N:0:CGATGT
@MISEQ:8:000000000-A7VC1:1:1101:18239:1496 1:N:0:CGATGT
@MISEQ:8:000000000-A7VC1:1:1101:13371:1512 1:N:0:CGATGT
[ec2-user@ip-10-169-87-62 ecoli_exeter]$ head E_Coli_CGATGT_L001_R2_001.fastq | grep MISEQ
@MISEQ:8:000000000-A7VC1:1:1101:14839:1482 2:N:0:CGATGT
@MISEQ:8:000000000-A7VC1:1:1101:18239:1496 2:N:0:CGATGT
@MISEQ:8:000000000-A7VC1:1:1101:13371:1512 2:N:0:CGATGT
```

The only difference in the headers for the two reads is the read number. Of course this is no guarantee that all the headers in the file are consistent. To get some more confidence repeat the above commands using 'tail' instead of 'head' to compare reads at the end of the files.

You can also check that there is an identical number of reads in each file using cat, grep and wc –l:

```
cat E_Coli_CGATGT_L001_R1_001.fastq | grep MISEQ | wc -l
```

```
cat E_Coli_CGATGT_L001_R2_001.fastq | grep MISEQ | wc -l
```

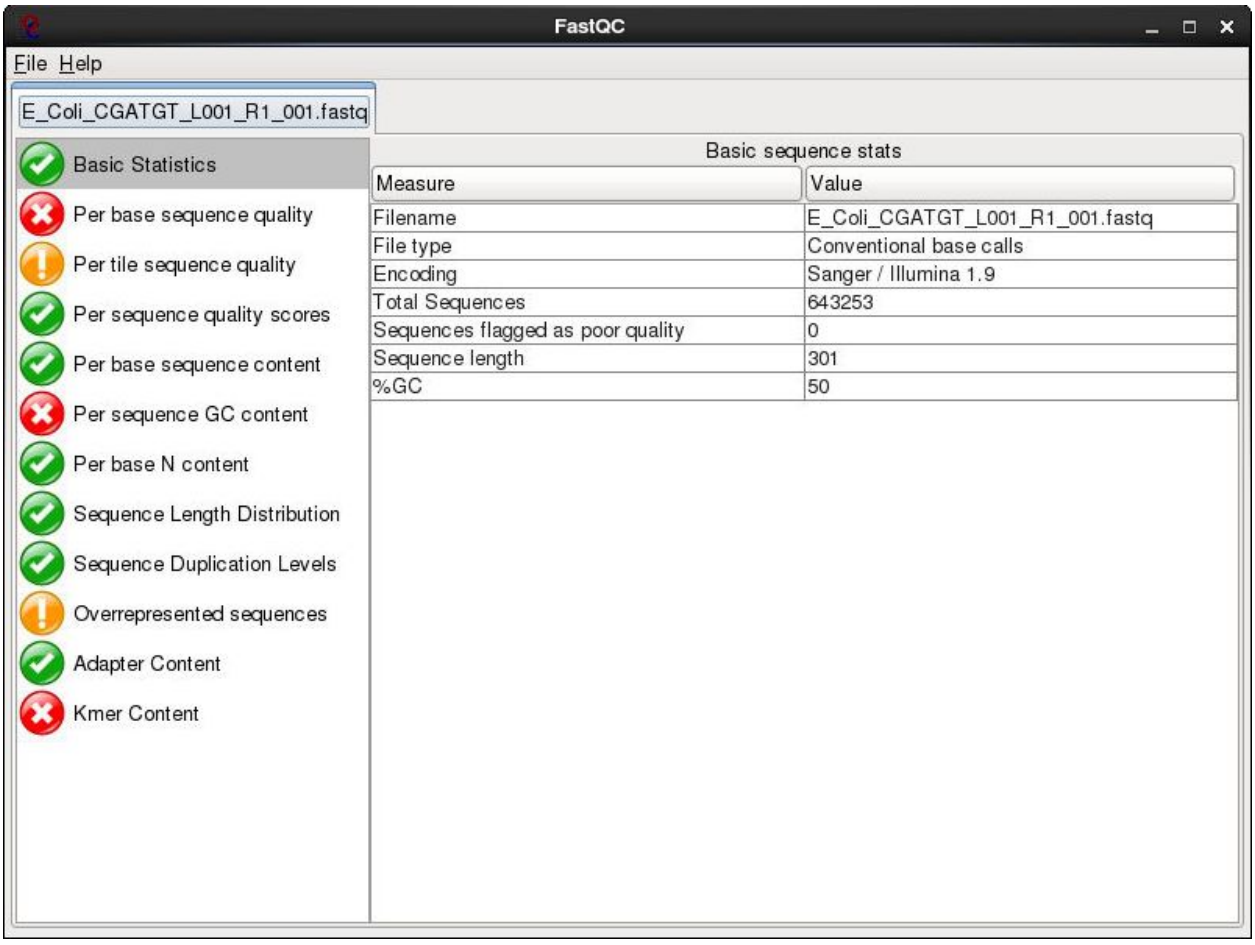
Now, let's run the fastqc program on the data. Unlike the QC lab, we will open up a Graphical User Interface (GUI) and load the data this way. To do this, run:

```
fastqc &
```

Load the `E_Coli_CGATGT_L001_R1_001.fastq` file from the `~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter` directory.

The fastqc program performs a number of tests which determines whether a green tick (pass), exclamation mark (warning), or red cross (fail) is displayed. However, it is important to realise that fastqc has no knowledge of what your library is or should look like. All of its tests are based on a completely random library with 50% GC content. Therefore if you have a sample which does not match these assumptions, it may 'fail' the library. For example, if you have a high AT or high GC organism it may fail the per sequence GC content. If you have any barcodes or low complexity libraries (e.g. small RNA libraries) they may also fail some of the sequence complexity tests.

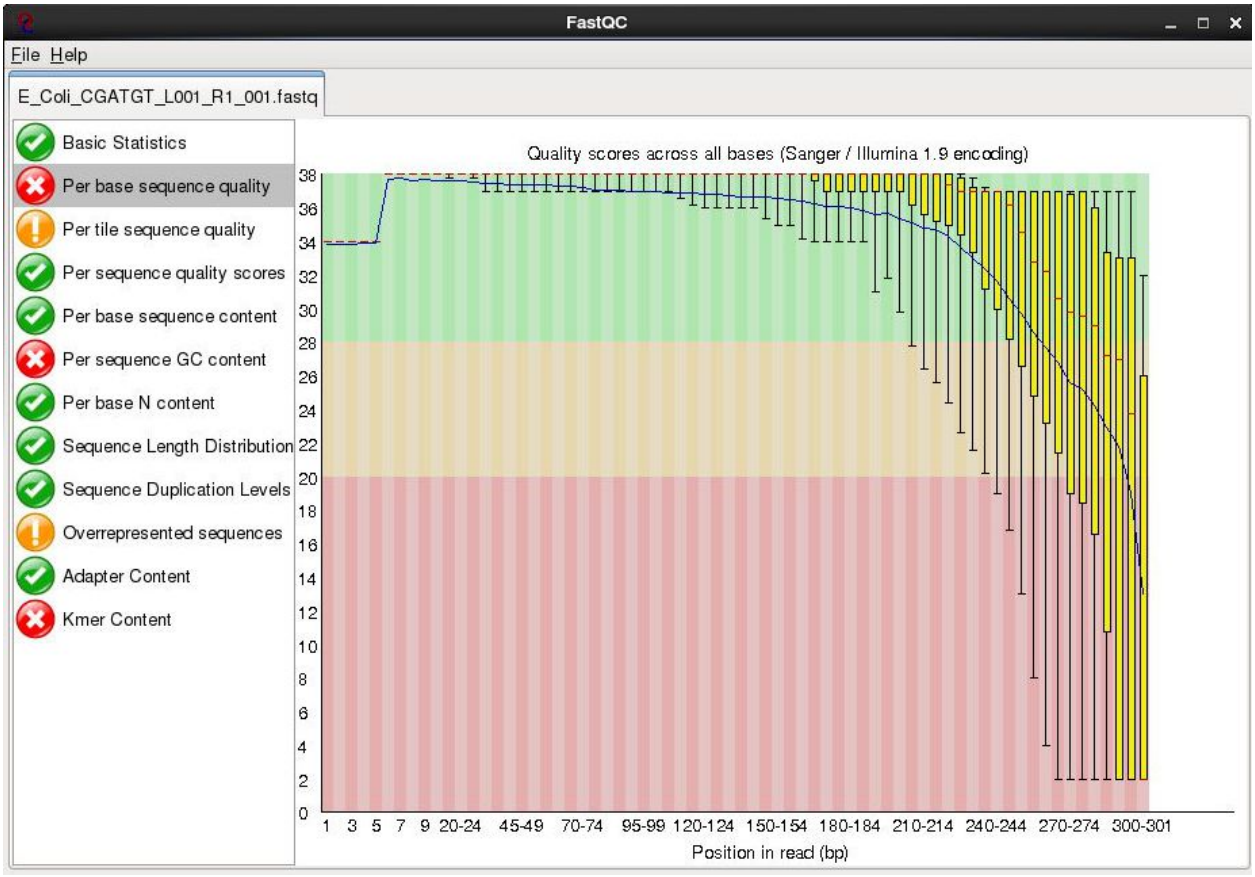
The bottom line is that you need to be aware of what your library is and whether what fastqc is reporting makes sense for that type of library.



In this case we have a number of errors and warnings which at first sight suggest there has been a problem - but don't worry too much yet. Let's go through them in turn.

Quality Scores

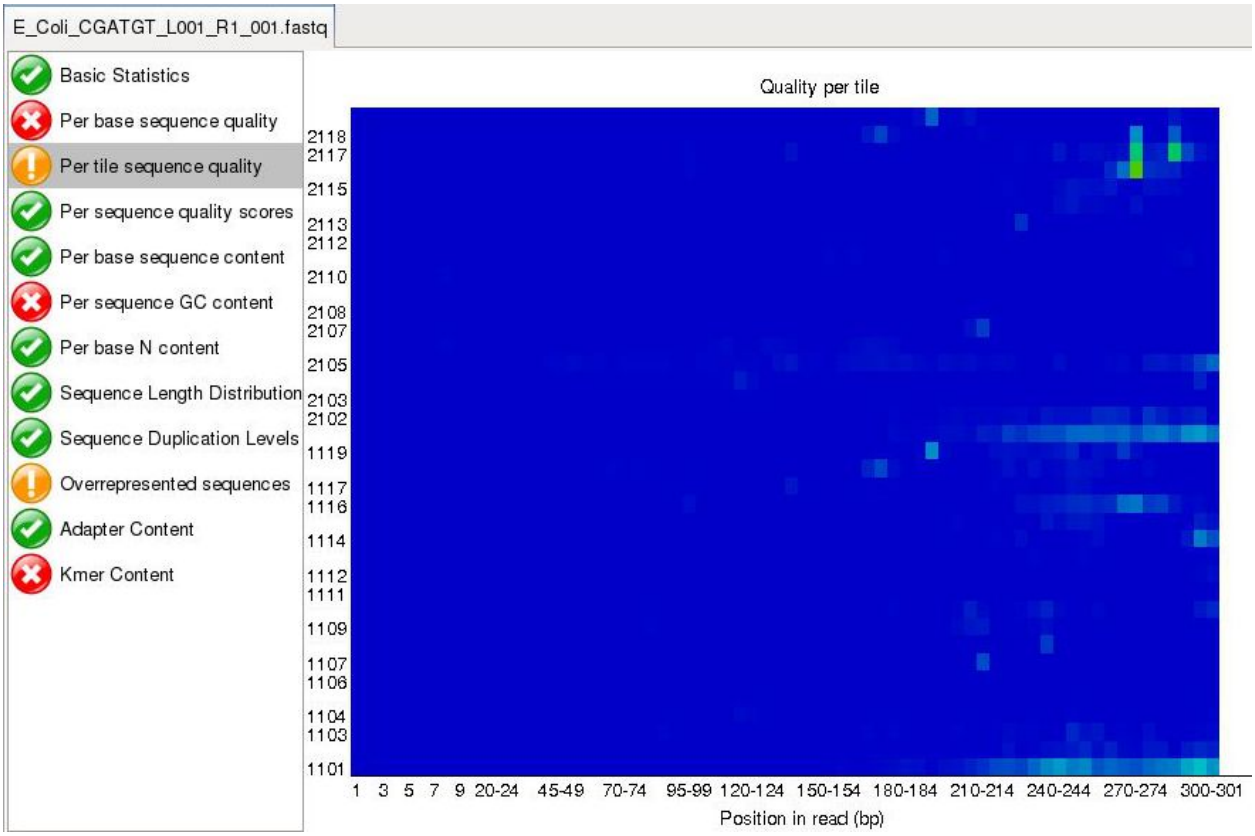
This is one of the most important metrics. If the quality scores are poor, either the wrong FASTQ encoding has been guessed by fastqc (see the title of the chart), or the data itself is poor quality. This view shows an overview of the range of quality values across all bases at each position in the FASTQ file. Generally anything with a median quality score greater than Q20 is regarded as acceptable; anything above Q30 is regarded as 'good'. For more details, see the help documentation in fastqc.



In this case this check is red - and it is true that the quality drops off at the end of the reads. It is normal for read quality to get worse towards the end of the read. You can see that at 250 bases the quality is still very good.

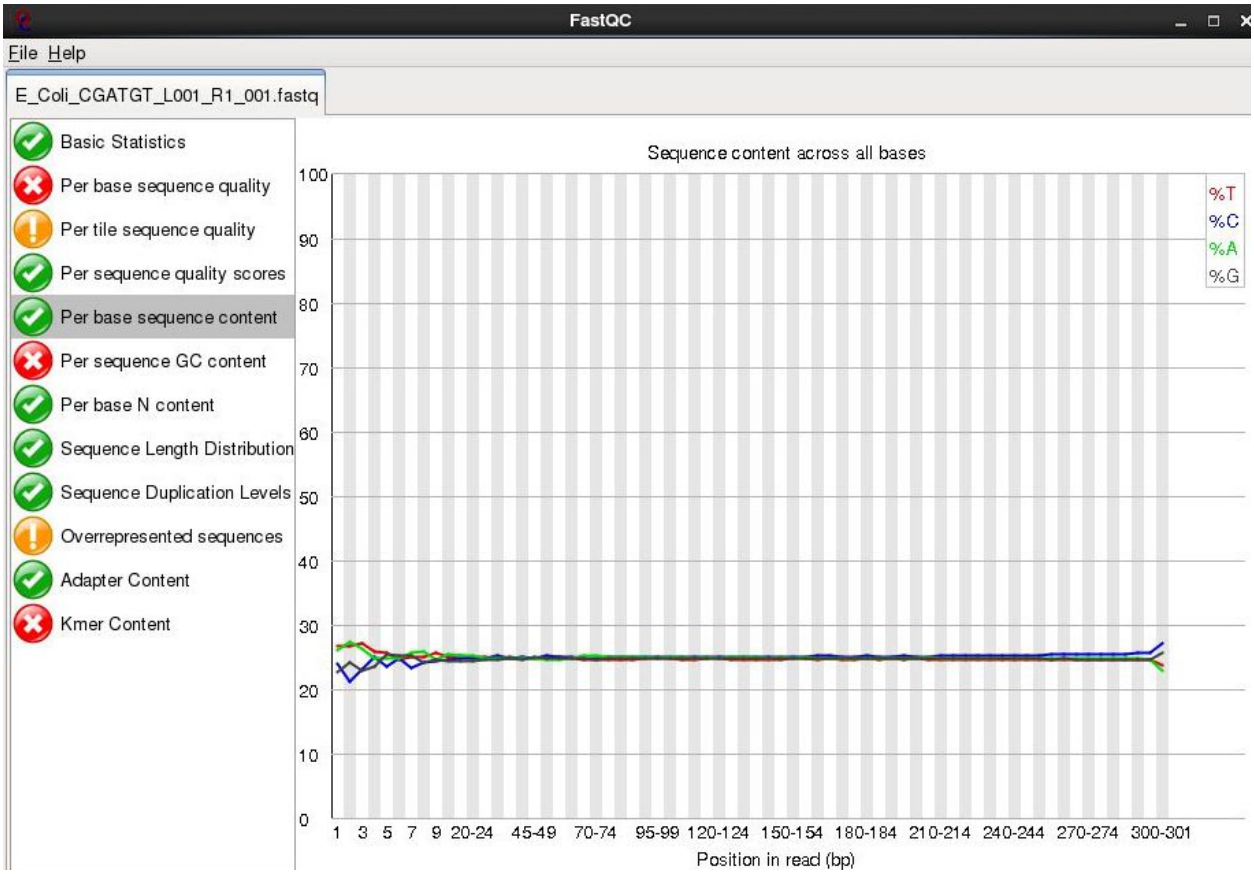
Per tile Sequence Quality

This is a purely technical view on the sequencing run, it is more important for the team running the sequencer. The sequencing flowcell is divided up into areas called cells. The colour of the tiles indicate the read quality and you can see that the quality drops off in some cells faster than others. This maybe because of the way the sample flowed over the flowcell or a mark or smear on the lens of the optics.



Per-base Sequence Content:

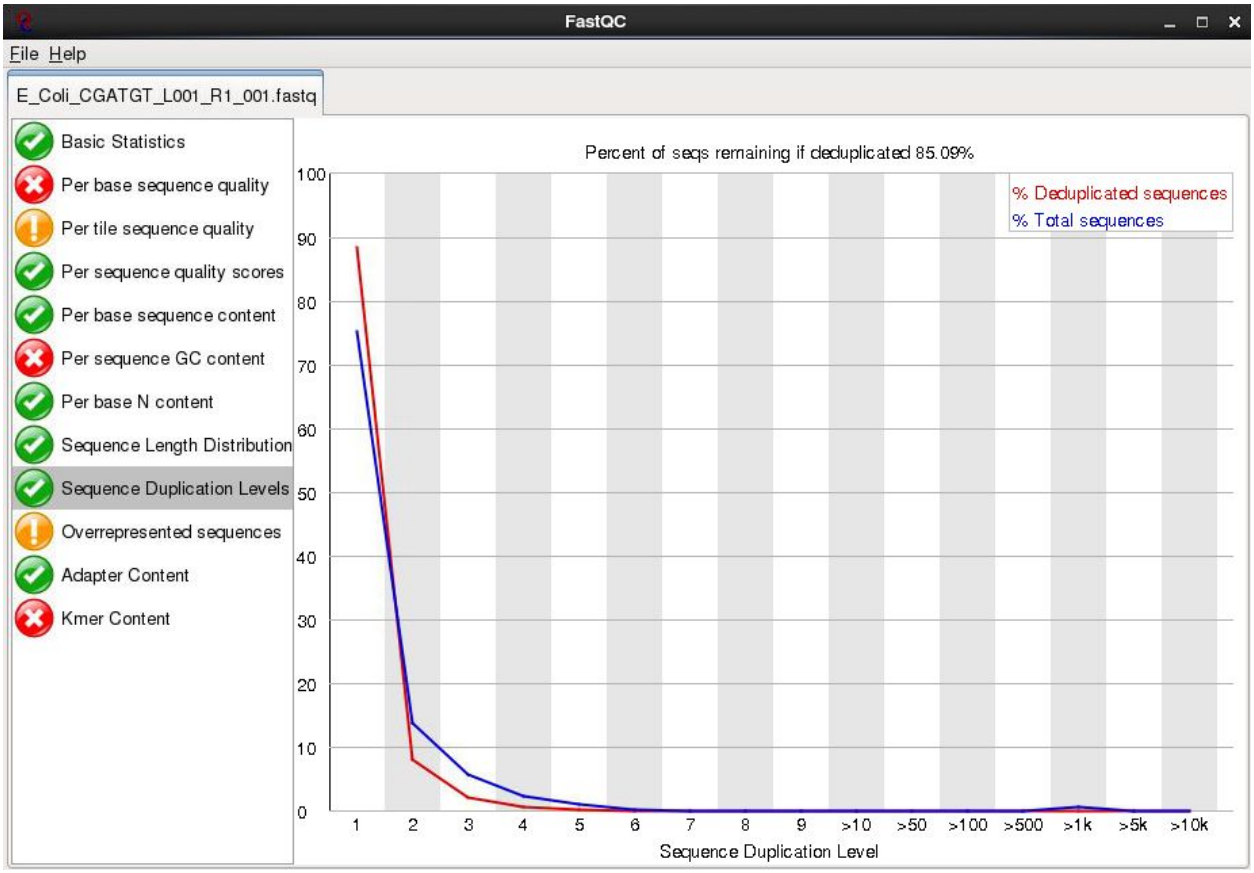
For a completely randomly generated library with a GC content of 50% one expects that at any given position within a read there will be a 25% chance of finding an A,C,T or G base. Here we can see that our library satisfies these criteria, although there appears to be some minor bias at the beginning of the read. This may be due to PCR duplicates during amplification or during library preparation. It is unlikely that one will ever see a perfectly uniform distribution. See *http://sequencing.exeter.ac.uk/guide-to-your-data/quality-control/* for examples of good vs bad runs as well as the fastqc help for more details.



Sequence Duplication Levels:

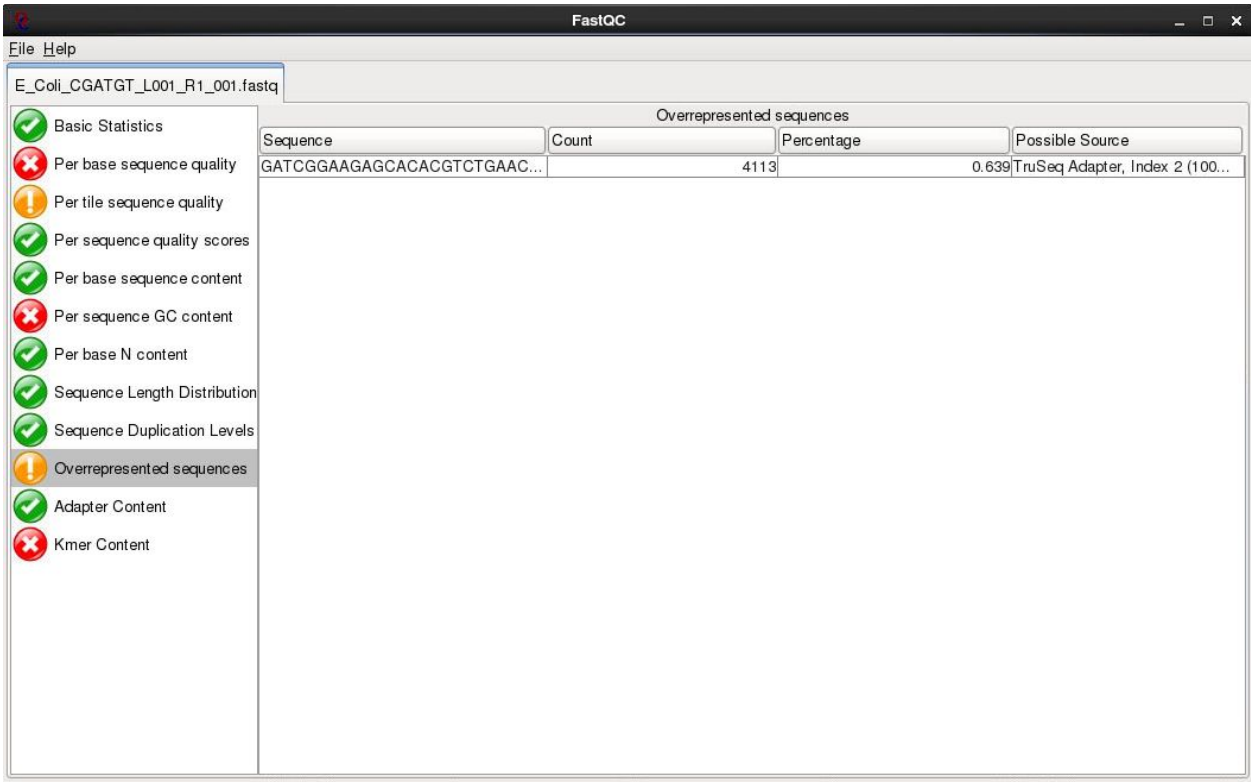
In a library that covers a whole genome uniformly most sequences will occur only once in the final set. A low level of duplication may indicate a very high level of coverage of the target sequence, but a high level of duplication is more likely to indicate some kind of enrichment bias (e.g. PCR over-amplification).

This module counts the degree of duplication for every sequence in the set and creates a plot showing the relative number of sequences with different degrees of duplication.



Overrepresented Sequences

This checks for sequences that occur more frequently than expected in your data. It also checks any sequences it finds against a small database of known sequences. In this case it has found that a small number of reads 4000 out of 600000 appear to contain a sequence used in the preparation for the library. A typical cause is that the original DNA was shorter than the length of the read - so the sequencing overruns the actual DNA and runs in to the adaptors used to bind it to the flowcell.



There are other reports available:

Have a look at them and at what the author of FastQC has to say here: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3%20Analysis%20Modules/ or check out their youtube tutorial video: https://www.youtube.com/watch?v=bz93ReOv87Y.

Remember the error and warning flags are his (albeit experienced) judgement of what typical data should look like. It is up to you to use some initiative and understand whether what you are seeing is typical for your dataset and how that might affect any analysis you are performing.

Task 2

Do the same for the raw read 2 as we have for raw read 1. Open fastqc and analyse the read 2 file. Look at the various plots and metrics which are generated. How similar are they?

Also look at the cleaned reads. How do they differ? You should notice very little change (since comparatively few reads were filtered). However, you should notice a significant improvement in quality and the absence of adaptor sequences.

We can perform a quick check (although this by no means guarantees) that the sequences in read 1 and read 2 are in the same order by checking the ends of the two files and making sure that the headers are the same.

```
head E_Coli_CGATGT_L001_R1_001.filtered.fastq | grep MISEQ
head E_Coli_CGATGT_L001_R2_001.filtered.fastq | grep MISEQ
tail E_Coli_CGATGT_L001_R1_001.filtered.fastq | grep MISEQ
tail E_Coli_CGATGT_L001_R2_001.filtered.fastq | grep MISEQ
```

Check the number of reads in each filtered file. They should be the same. To do this use the grep command to search for the number of times the header appears, e.g.

```
grep -c “MISEQ” E_Coli_CGATGT_L001_R1_001.filtered.fastq
```

Do the same for the `E_Coli_CGATGT_L001_R2_001.filtered.fastq` file.

Note: Typically when submitting raw Illumina data to NCBI or EBI you would submit unfiltered data, so don't delete your original fastq files!

A note on quality control using MultiQC

MultiQC **(**https://multiqc.info/) is software which will aggregate reports (such as fastqc) across a whole experiment. For example, it will aggregate fastqc reports for multiple samples. Say you have 1000 samples in an experiment, you’re not going to want to open 1000 tabs on an internet browser! MultiQC supports many other QC softwares (such as outputs from qualimap, quast, bcftools). At the time of writing (2019), MultiQC supports 72 commonly-used bioinformatics tools. Take a look!

A note on checking for contaminants:

A number of tools are available now which also enable to you to quickly search reads and assign them to particular species or taxonomic groups. These can serve as a quick check to make sure your samples or libraries are not contaminated with DNA from other sources. If you are performing a de-novo assembly for example and unwittingly have DNA sequence present from multiple organisms, you will risk poor results and chimeric contigs.

Some ‘contaminants’ can turn out to be inevitable by-products of sampling and DNA extraction. This is often the case with algae or other symbionts. In addition, some groups have made some amazing discoveries such as the discovery of a third symbiont (which turned out to be a yeast) in lichen. http://science.sciencemag.org/content/353/6298/488.full

Some tools you can use to check the taxonomic classification of reads include:

- Kraken

- Centrifuge
- Blobology
- Kaiju
- Blast (in conjunction with subsampling your reads) and Krona to plot results.

Blobtools is also a useful tool for quality control post assembly. Blobtools visualizes the GC content, coverage, and taxonomic classification of assembled contigs to enable screening for potential contaminants. You can read more about Blobtools here , but we won't do either of these steps today.

Aligning Illumina Data to a Reference Sequence

Now that we have checked the quality of our raw data, we can begin to align the reads against a reference sequence. In this way we can compare how the reference sequence and the strain we have sequenced compare to one another.

To do this we will be using a program called BWA (Burrows Wheeler Aligner *Li H. and Durbin R. (2009) Fast and accurate short read alignment with Burrows-Wheeler Transform. Bioinformatics, 25:1754-60.*). This uses an algorithm called (unsurprisingly) Burrows Wheeler to rapidly map reads to the reference genome. BWA also allows for a certain number of mismatches to account for variants which may be present in strain 1 vs the reference genome. Unlike other alignment packages such as Bowtie (version 1) BWA allows for insertions or deletions as well. (Note, there is now a Bowtie2 tool that allows for insertions and deletions, but we'll continue to use BWA here). There are also a host of newer aligners such as minimap2 (https://github.com/lh3/minimap2) that allow for long-read sequencing and employ different algorithms. In fact, Heng Li (creator of both BWA and minimap2) suggests that minimap2 is now superior, although bwa is still recommended for short read genomic data. For information on this, please see Heng Li's post here: http://lh3.github.io/2018/04/02/minimap2-and-the-future-of-bwa

By mapping reads against a reference, what we mean is that we want to go from a FASTQ file listing lots of reads, to another type of file (which we'll describe later) which lists the reads AND where/if it maps against the reference genome. The figure below illustrates what we are trying to achieve here. Along the top in grey is the reference sequence. The coloured sequences below indicate individual sequences and how they map to the reference. If there is a real variant in a bacterial genome we would expect that (nearly) all the reads would contain the variant at the relevant position rather than the same base as the reference genome. Remember that error rates for any single read on second generation platforms tend to be around 0.5-1%. Therefore a 300bp read is on average likely to contain at 2-3 errors.

Let's look at 2 potential sources of artefacts.

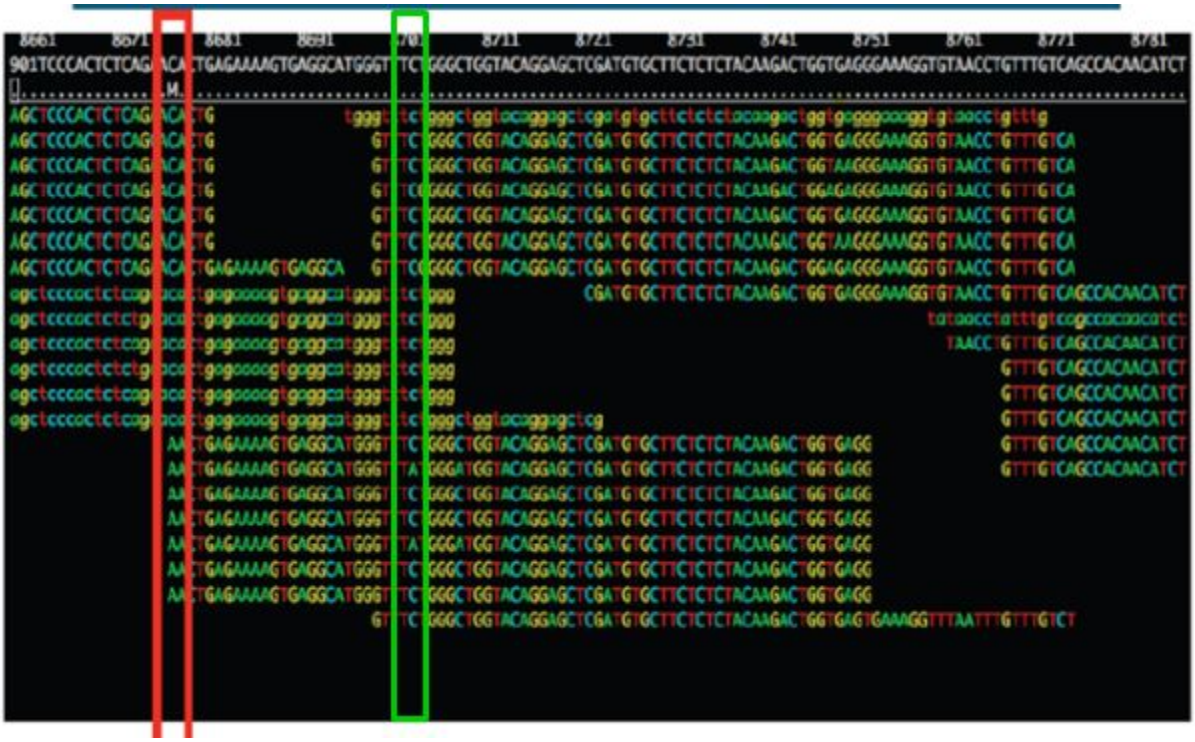
Sequencing Error

The region highlighted in green on the right shows that most reads agree with the reference sequence (i.e. C-base). However, 2 reads near the bottom show an A-base. In this situation we can safely assume that the A-bases are due to a sequencing error rather than a genuine variant since the

‘variant’ has only one read supporting it. If this occurred at a higher frequency however, we would struggle to determine whether it was a genuine variant or an error.

PCR Duplication

The highlighted region in red on the left shows where there appears to be a variant. A C-base is present in the reference and half the reads, whilst an A-base is present in a set of reads which all start at the same position.



Is this a genuine difference or a sequencing or sample prep error? What do you think? If this was a real sample, would you expect all the reads containing an A to start at the same location?

The answer is probably not. This 'SNP' is in fact probably an artefact of PCR duplication. I.e. the same fragment of DNA has been replicated many times more than the average and happens to contain an error at the first position. We can filter out such reads during alignment to the reference (see later).

Note that the entire region above seems to contain lots of PCR duplicates with reads starting at the same location. In the case of the region highlighted in red, this will likely cause a false SNP call. The area in green also contains PCR duplicates – the As at these positions are probably either sequencing errors or errors introduced during PCR.

It's always important to think critically about any finding - don't assume that whatever bioinformatic tools you are using are perfect. Or that you have used them perfectly.

Indexing a Reference Genome

Before we can start aligning reads to a reference genome, the genome sequence needs to be indexed. This means sorting the genome into easily searched chunks, a bit like an index in a book.

Task 3: Genera ting an index file from the reference sequence

Change directory to the reference directory:

```
cd ~/workshop_materials/genomics_tutorial/data/reference/U00096/
```

In this directory we have 2 files. U00096.fna is a FASTA file which contains the reference genome sequence. The U00096.gff file contains the annotation for this genome. We will use this later.

First, let's looks at the bwa command itself. Type:

```
bwa
```

This should yield something like:

```
Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.17-r1194-dirty
Contact: Heng Li <lh3@sanger.ac.uk>

Usage:   bwa <command> [options]

Command: index         index sequences in the FASTA format
         mem           BWA-MEM algorithm
         fastmap       identify super-maximal exact matches
         pemerge       merge overlapping paired ends (EXPERIMENTAL)
         aln           gapped/ungapped alignment
         samse         generate alignment (single ended)
         sampe         generate alignment (paired ended)
         bwasw         BWA-SW for long queries

         shm           manage indices in shared memory
         fa2pac        convert FASTA to PAC format
         pac2bwt       generate BWT from PAC
         pac2bwtgen    alternative algorithm for generating BWT
         bwtupdate     update .bwt to the new format
         bwt2sa        generate SA from BWT and Occ

Note: To use BWA, you need to first index the genome with `bwa index'.
      There are three alignment algorithms in BWA: `mem', `bwasw', and
      `aln/samse/sampe'. If you are not sure which to use, try `bwa mem'
      first. Please `man ./bwa.1' for the manual.
```

BWA is actually a suite of programs which all perform different functions. We are only going to use two during this workshop, bwa index, bwa mem

If we type:

```
bwa index
```

We can see more options for the bwa index command:

```
Usage:   bwa index [options] <in.fasta>

Options: -a STR    BWT construction algorithm: bwtsw, is or rb2 [auto]
         -p STR    prefix of the index [same as fasta name]
         -b INT    block size for the bwtsw algorithm (effective with -a bwtsw) [10000000]
         -6        index files named as <in.fasta>.64.* instead of <in.fasta>.*
```

By default bwa index will use the `IS` algorithm to produce the index. This works well for most genomes, but for very large ones (e.g. vertebrate) you may need to use bwtsw. For bacterial genomes the default algorithm will work fine.

Now we will create a reference index for the genome using BWA:

```
bwa index U00096.fna
```

If you now list the directory contents, you will notice that the BWA index program has created a set of new files. These are the index files BWA needs.

Task 4: Aligning Reads to the Indexed Reference Sequence

Now we can begin to align read 1 and read 2 to the reference genome. First of all change back into the `~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/` directory and create a subdirectory to contain our remapping results.

```
cd ~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/
```

```
mkdir remapping_to_reference
```

```
cd remapping_to_reference
```

We'll use the 'bwa mem' alignment algorithm to map the reads to the target genome. Let's explore the alignment options BWA MEM has to offer. Type:

```
bwa mem
```

```
Usage: bwa mem [options] <idxbase> <in1.fq> [in2.fq]

Algorithm options:

       -t INT        number of threads [1]
       -k INT        minimum seed length [19]
       -w INT        band width for banded alignment [100]
       -d INT        off-diagonal X-dropoff [100]
       -r FLOAT      look for internal seeds inside a seed longer than {-k} * FLOAT [1.5]
       -y INT        seed occurrence for the 3rd round seeding [20]
       -c INT        skip seeds with more than INT occurrences [500]
       -D FLOAT      drop chains shorter than FLOAT fraction of the longest overlapping chain [0.50]
       -W INT        discard a chain if seeded bases shorter than INT [0]
       -m INT        perform at most INT rounds of mate rescues for each read [50]
       -S            skip mate rescue
       -P            skip pairing; mate rescue performed unless -S also in use

Scoring options:

       -A INT        score for a sequence match, which scales options -TdBOELU unless overridden [1]
       -B INT        penalty for a mismatch [4]
       -O INT[,INT]  gap open penalties for deletions and insertions [6,6]
       -E INT[,INT]  gap extension penalty; a gap of size k cost '{-O} + {-E}*k' [1,1]
       -L INT[,INT]  penalty for 5'- and 3'-end clipping [5,5]
       -U INT        penalty for an unpaired read pair [17]

       -x STR        read type. Setting -x changes multiple parameters unless overridden [null]
                     pacbio: -k17 -W40 -r10 -A1 -B1 -O1 -E1 -L0  (PacBio reads to ref)
                     ont2d: -k14 -W20 -r10 -A1 -B1 -O1 -E1 -L0  (Oxford Nanopore 2D-reads to ref)
                     intractg: -B9 -O16 -L5  (intra-species contigs to ref)

Input/output options:
```

The basis format of the command is:

`Usage: bwa mem [options] <idxbase> <in1.fq> <in2.fq>`

We can see that we need to provide BWA with a FASTQ files containing the raw reads (denoted by <in.fq> and <in2.fq>) to align to a reference file (listed as <idxbase>). There are also a number of options. The most important are the maximum number of differences in the seed (-k i.e. the first 32 bp of the sequence vs the reference), the number of processors the program should use (your machine has 2 processors).

Our reference sequence is in
`~/workshop_materials/genomics_tutorial/data/reference/U00096/U00096.fna`

Our filtered reads in
`~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/E_Coli_CGATGT_L001_R1_001.filtered.fastq`
`~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/E_Coli_CGATGT_L001_R2_001.filtered.fastq`

So to align our paired reads using processors and output to file `E_Coli_CGATGT_L001_filtered.sam`:

type, all on one line:

```
bwa mem -t 2
~/workshop_materials/genomics_tutorial/data/reference/U00096/U00096.fna
~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/E_Coli_
CGATGT_L001_R1_001.filtered.fastq
~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/E_Coli_
CGATGT_L001_R2_001.filtered.fastq > E_Coli_CGATGT_L001_filtered.sam
```

This will take about 5 minutes to complete.
There will be quite a lot of output but the end should look like:

```
[M::process] read 70094 sequences (18721937 bp)...
[M::mem_pestat] # candidate unique pairs for (FF, FR, RF, RR): (1, 26431, 0, 3)
[M::mem_pestat] skip orientation FF as there are not enough pairs
[M::mem_pestat] analyzing insert size distribution for orientation FR...
[M::mem_pestat] (25, 50, 75) percentile: (518, 580, 642)
[M::mem_pestat] low and high boundaries for computing mean and std.dev: (270, 890)
[M::mem_pestat] mean and std.dev: (577.01, 102.93)
[M::mem_pestat] low and high boundaries for proper pairs: (146, 1014)
[M::mem_pestat] skip orientation RF as there are not enough pairs
[M::mem_pestat] skip orientation RR as there are not enough pairs
[M::mem_process_seqs] Processed 75290 reads in 7.280 CPU sec, 3.628 real sec
[M::mem_pestat] # candidate unique pairs for (FF, FR, RF, RR): (1, 24724, 0, 0)
[M::mem_pestat] skip orientation FF as there are not enough pairs
[M::mem_pestat] analyzing insert size distribution for orientation FR...
[M::mem_pestat] (25, 50, 75) percentile: (519, 581, 641)
[M::mem_pestat] low and high boundaries for computing mean and std.dev: (275, 885)
[M::mem_pestat] mean and std.dev: (577.10, 100.85)
[M::mem_pestat] low and high boundaries for proper pairs: (153, 1007)
[M::mem_pestat] skip orientation RF as there are not enough pairs
[M::mem_pestat] skip orientation RR as there are not enough pairs
[M::mem_process_seqs] Processed 70094 reads in 6.572 CPU sec, 3.306 real sec
[main] Version: 0.7.15-r1142-dirty
[main] CMD: bwa mem -t 2 ../../reference/U00096/U00096.fna E_Coli_CGATGT_L001_R1_001.filtered.fastq E_Coli_CGATGT_L001_R2_001.filtered.fastq
[main] Real time: 63.015 sec; CPU: 124.824 sec
```

Viewing the alignment

Once the alignment is complete, list the directory contents and check that the alignment file is present.

```
ls -lh
```

```
genomics@genomics-build-2:/mnt/genomics_storage/workshop_data/genomics_tutorial/data/sequencing/ecoli_exeter/remapping_to_ref
total 780M
-rw-r--r-- 1 genomics workshop 780M Jan  2 15:37 E_Coli_CGATGT_L001_filtered.sam
```

Note: ls -lh outputs the size of the file in human readable format (780Mb in this case – yours may be slightly different depending on the storage options you selected when you started the AMI)

The raw alignment is stored in what is called SAM format (Simple AlignMent format). It is in plain text format and you can view it if you wish using the 'less' command. Do not try to open the whole file in a text editor as you will likely run out of memory!

```
less E_Coli_CGATGT_L001_filtered.sam
```

```
MISEQ:8:000000000-A7VC1:1:1101:17200:1633       83      gi|545778205|gb|U00096.3|       881006  60      137M    =
       880711  -432    GGTAAAGATGCCGGGGCGACGGGAAAGCCGGAACGGCGTGGTTCATCGGTAATGTTCCGCAAACCGGGCGATCAGGTTTCGGTGGCAGACT
TGAACAAAGGTGTGATTATCCAGTCCGGTAATGACGCCTGTATTGC       @9,@D>@8+8+++>@>+?A+AE?A86+++B:+8++>+B,B:,,,8,,EA,AC,8++++,,B
@8++C@@,C:,,CC8++++EC,CC,C@<,,,CCC,CCC@C,C,CC,9E8CCFEEDGGGGGGGGGGGGGGGCCCC9       NM:i:5  MD:Z:12T13T4C2T23T78
    AS:i:112        XS:i:0
MISEQ:8:000000000-A7VC1:1:1101:17200:1633       163     gi|545778205|gb|U00096.3|       880711  60      84M     =
       881006  432     TACTCGGGTGGCCTTTCTCCCGCACTACTCCTCTCTCCTTCGTGCTCTTCCAGCGGGTTCTGCATTTTTCTTCCTTTTTTCCCC    8,A
6C,,+;++;,;CC,<,,;+8++7,;6CC<C@C<CECCC,;,,9CCC,<,,;+++88BC,<,<99@B,9:BEB@@@=+:??A    NM:i:9  MD:Z:12A3A1G0A4A19G4G
19A9G4     AS:i:39 XS:i:0
MISEQ:8:000000000-A7VC1:1:1101:10456:1673       83      gi|545778205|gb|U00096.3|       1864278 60      42M     =
       1863862 -458    GGGTAAAACTTGTGAAATCGATCTTGAATCACATGGCGAATT      CC;,@C<<,,9EAFFFC7GGGGFGGGGGGGGGGGGGCCCC9
      NM:i:0  MD:Z:42 AS:i:42 XS:i:0
```

Each alignment line has 11 mandatory fields for essential alignment information such as mapping position, and a variable number of optional fields for flexible or aligner specific information. For further details as to what each field means see https://en.wikipedia.org/wiki/SAM_(file_format) or http://samtools.sourceforge.net/SAM1.pdf

Task 9: Convert SAM to BAM File

Before we can visualise the alignment however, we need to convert the SAM file to a BAM (Binary AlignMent format) which can be read by most software analysis packages. To do this we will use another suite of programs called samtools. Type:

```
samtools view
```

```
Usage: samtools view [options] <in.bam>|<in.sam>|<in.cram> [region ...]

Options:
  -b       output BAM
  -C       output CRAM (requires -T)
  -1       use fast BAM compression (implies -b)
  -u       uncompressed BAM output (implies -b)
  -h       include header in SAM output
  -H       print SAM header only (no alignments)
  -c       print only the count of matching records
  -o FILE  output file name [stdout]
  -U FILE  output reads not selected by filters to FILE [null]
  -t FILE  FILE listing reference names and lengths (see long help) [null]
  -L FILE  only include reads overlapping this BED FILE [null]
  -r STR   only include reads in read group STR [null]
  -R FILE  only include reads with read group listed in FILE [null]
  -q INT   only include reads with mapping quality >= INT [0]
  -l STR   only include reads in library STR [null]
  -m INT   only include reads with number of CIGAR operations consuming
           query sequence >= INT [0]
  -f INT   only include reads with all bits set in INT set in FLAG [0]
  -F INT   only include reads with none of the bits set in INT set in FLAG [0]
  -x STR   read tag to strip (repeatable) [null]
  -B       collapse the backward CIGAR operation
  -s FLOAT integer part sets seed of random number generator [0];
           rest sets fraction of templates to subsample [no subsampling]
  -@, --threads INT
           number of BAM/CRAM compression threads [0]
  -?       print long help, including note about region specification
  -S       ignored (input format is auto-detected)
      --input-fmt-option OPT[=VAL]
               Specify a single input file format option in the form
               of OPTION or OPTION=VALUE
  -O, --output-fmt FORMAT[,OPT[=VAL]]...
               Specify output format (SAM, BAM, CRAM)
      --output-fmt-option OPT[=VAL]
               Specify a single output file format option in the form
               of OPTION or OPTION=VALUE
  -T, --reference FILE
               Reference sequence FASTA FILE [null]
```

We can see that we need to provide samtools view with a reference genome in FASTA format file (-T), the -b and -S flags to say that the output should be in BAM format and the input in SAM, plus the alignment file.

Remember our reference sequence is in:
`~/workshop_materials/genomics_tutorial/data/reference/U00096/U00096.fna`

Type (all on one line):

```
samtools view -bS -T
~/workshop_materials/genomics_tutorial/data/reference/U00096/U00096.fna
E_Coli_CGATGT_L001_filtered.sam > E_Coli_CGATGT_L001_filtered.bam
```

This should take around 2 minutes. Note that for larger datasets you may wish to set multiple threads as well with the --threads option.

```
ls -lh
```

It's always good to check that your files have processed correctly if something goes wrong it's better to catch it immediately.

Note that the bam file is smaller than the sam file - this is to be expected as the binary format is more efficient.

Task 5: Sort BAM File

Once this is complete we then need to sort the BAM file so that the reads are stored in the order they appear along the chromosomes. We can do this using the samtools sort command. In this instance, we will sort

```
samtools sort -n E_Coli_CGATGT_L001_filtered.bam -o
E_Coli_CGATGT_L001_filtered.sorted.bam
```

The -n option sorts the sam file by read name (which is needed below for samtools fixmate)
This will take another minute or so.

Task 6: Remove Suspected PCR Duplicates

Especially when using paired-end reads, samtools can do a reasonably good job of removing potential PCR duplicates (see the first part of this workshop if you are unsure what this means).

Again, samtools has a great little command to do this called markdup. To run this tool, we will have to generate some intermediate files (see Task 9 for cleaning these up afterwards), using other samtools programs. At each stage of the process, we will document which tool has been run by appending the name of the tool on the end of the file.

To run samtools markdup, you'll first need to run samtools fixmate which fills in various coordinates and flags from a name-sorted alignment (hence why used the -n option above)

On the command-line type:

```
samtools fixmate -m E_Coli_CGATGT_L001_filtered.sorted.bam
E_Coli_CGATGT_L001_filtered.sorted.fixmate.bam
```

And then sort the file again (this time without the -n option!), which will sort the file by genomic coordinates.

```
samtools sort E_Coli_CGATGT_L001_filtered.sorted.fixmate.bam -o
E_Coli_CGATGT_L001_filtered.sorted.fixmate.position.bam
```

Now we can run samtools markdup to remove PCR duplicates!

```
samtools markdup E_Coli_CGATGT_L001_filtered.sorted.fixmate.position.bam
E_Coli_CGATGT_L001_filtered.sorted.fixmate.position.markdup.bam
```

You may notice some warnings about inconsistent BAM file for pair - this is just a warning that a pair of reads does not align together on the genome within the expected tolerance - it is normal to expect some of these, and you can ignore.

Task 7: Index the BAM File

Most programs used to view BAM formatted data require an index file to locate the reads mapping to a particular location quickly. You can think of this as an index in a book, telling you where to go to find particular phrases or words. We'll use the samtools index command to do this.

Type:

```
samtools index
E_Coli_CGATGT_L001_filtered.sorted.fixmate.position.markdup.bam
```

We should obtain a .bai file (known as a BAM-index file).

Task 8: Obtain Mapping Statistics

Finally we can obtain some summary statistics.

```
samtools flagstat
E_Coli_CGATGT_L001_filtered.sorted.fixmate.position.markdup.bam >
mappingstats.txt
```

This should only take a few seconds. Once complete view the mappingstats.txt file using a text-editor (e.g. gedit or nano) or the 'more' command.

```
genomics@ip-172-30-3-215$ more mappingstats.txt
1269900 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
1184 + 0 supplementary
18402 + 0 duplicates
918262 + 0 mapped (72.31% : N/A)
1268716 + 0 paired in sequencing
634358 + 0 read1
634358 + 0 read2
913166 + 0 properly paired (71.98% : N/A)
915664 + 0 with itself and mate mapped
1414 + 0 singletons (0.11% : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
genomics@ip-172-30-3-215$
```

So here we can see we have 1269900 reads in total, none of which failed QC.
72.31% of reads mapped to the reference genome and 71.98% mapped with the expected 500-600bp distance between them. 1414 reads could not have their read-pair mapped.

0 reads have mapped to a different chromosome than their pair (0 has a mapping quality > 5 – this is a Phred scaled quality score much as we say in the FASTQ files). If there were any such reads they would likely due to repetitive sequences

Task 9: Cleaning up

We have a number of leftover intermediate files which we can now remove to save space.

Type (all on one line):

```
rm E_Coli_CGATGT_L001_filtered.sam E_Coli_CGATGT_L001_filtered.sorted.bam
E_Coli_CGATGT_L001_filtered.sorted.fixmate.bam
E_Coli_CGATGT_L001_filtered.sorted.fixmate.position.bam
```

In case you get asked if you are sure to remove 4 arguments type in “yes” and hit enter.
You should now be left with the processed alignment file, the index file and the mapping stats.

Well done! You have now mapped, filtered and sorted your first whole genome data-set!
Let's take a look at it!

Task 10: QualiMap

Qualimap (http://qualimap.bioinfo.cipf.es/) is a program that summarises the alignment in much more detail than the mapping stats file we produced. It's a technical tool which allows you to assess the sequencing for any problems and biases in the sequencing and the alignment rather than a tool to deduce biological features.

There are a few options to the program, We want to run bamqc. Type:

```
qualimap bamqc
```

to get some help on this command.

To get the report, first make sure you are in the directory:
`~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/remapping_to_reference`
then run the command:

```
qualimap bamqc -outdir bamqc -bam
E_Coli_CGATGT_L001_filtered.sorted.fixmate.position.markdup.bam -gff
~/workshop_materials/genomics_tutorial/data/reference/U00096/U00096.gff
```

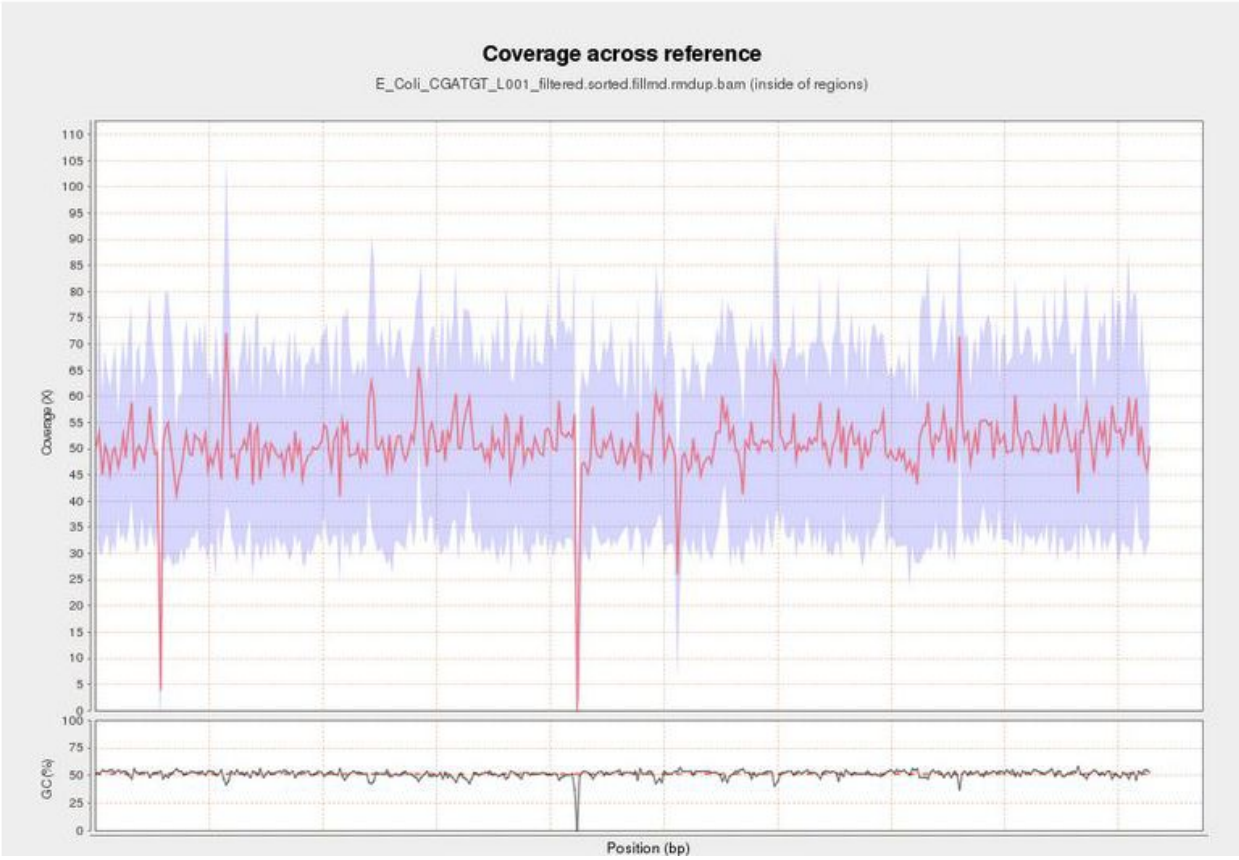
this creates a subfolder called `bamqc`

move into this directory and run

```
firefox qualimapReport.html
```

There is a lot in the report so just a few highlights:

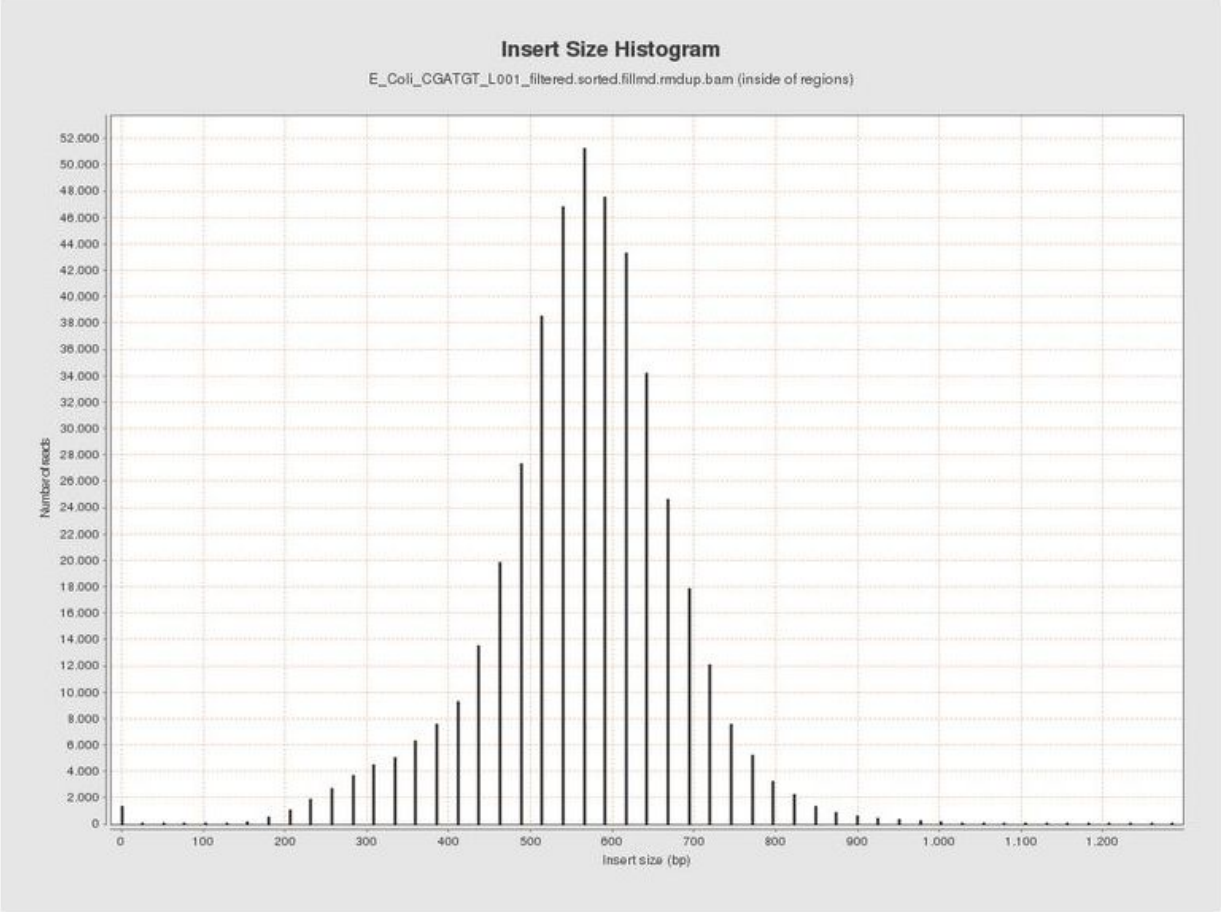
Coverage across reference



This shows the number of reads that 'cover' each section of the genome. The red line shows a rolling average around 50x - this means that on average every part of the genome was sequenced 50X. It is important to have sufficient depth of coverage in order to be confident that any features you find in your data are real and not a result of sequencing errors.

What do you think the regions of low/zero coverage correspond to?

Insert Size Histogram



The Insert Size Histogram displays the range of sizes of the DNA fragments. It shows how well your DNA was size selected before sequencing. Note that the 'insert' refers to the DNA that was inserted between the sequencing adaptors, so equates to the size range of the DNA that was used.
In this case we have 300 base pair paired end reads and our insert size varies around 600 bases - so there should only be a small gap between the reads that was not sequenced.

Have a look at some of the other graphs produced.

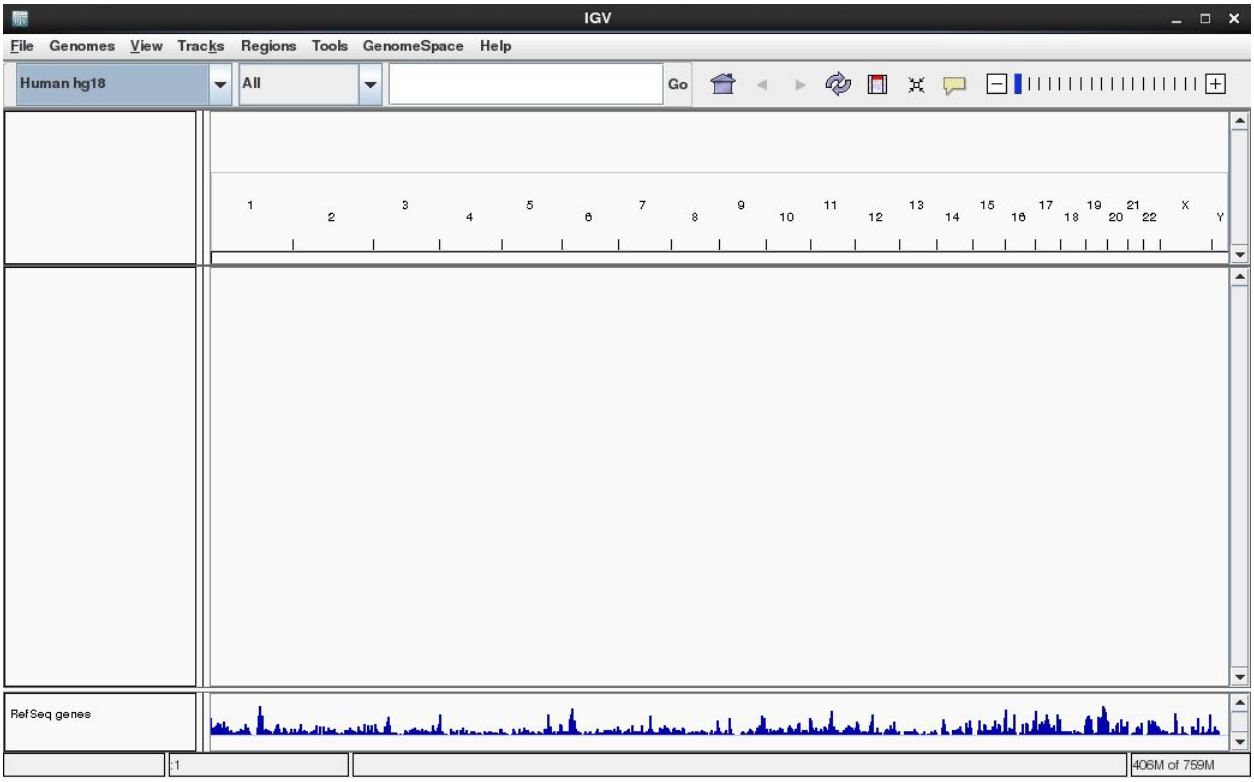
Task 11: Load the Integrative Genomics Viewer

The Integrative Genome Viewer (IGV) is a tool developed by the Broad Institute for browsing interactively the alignment data you produced. It has a wealth of features and we can only cover some basics to get you started. Go to http://www.broadinstitute.org/igv/ to get more information.

In your terminal, type

```
igv.sh
```

Or you can click the icon on the desktop.
IGV viewer should appear:

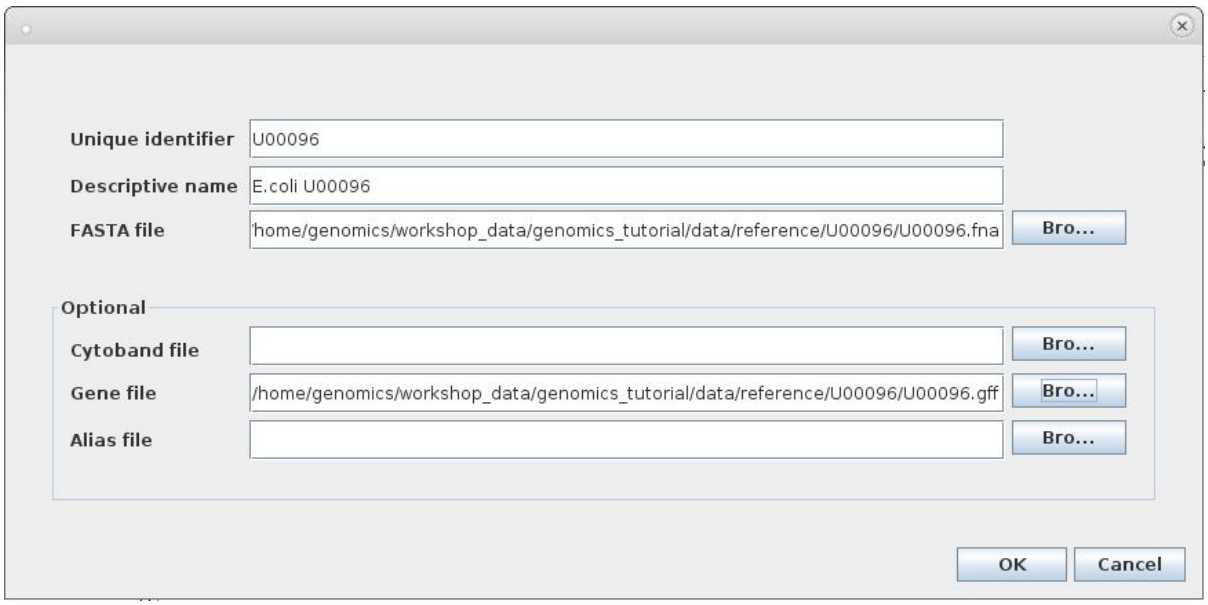


Notice that by default a human genome has been loaded.

Task 12a: Import the E.coli U00096 Reference Genome to IGV

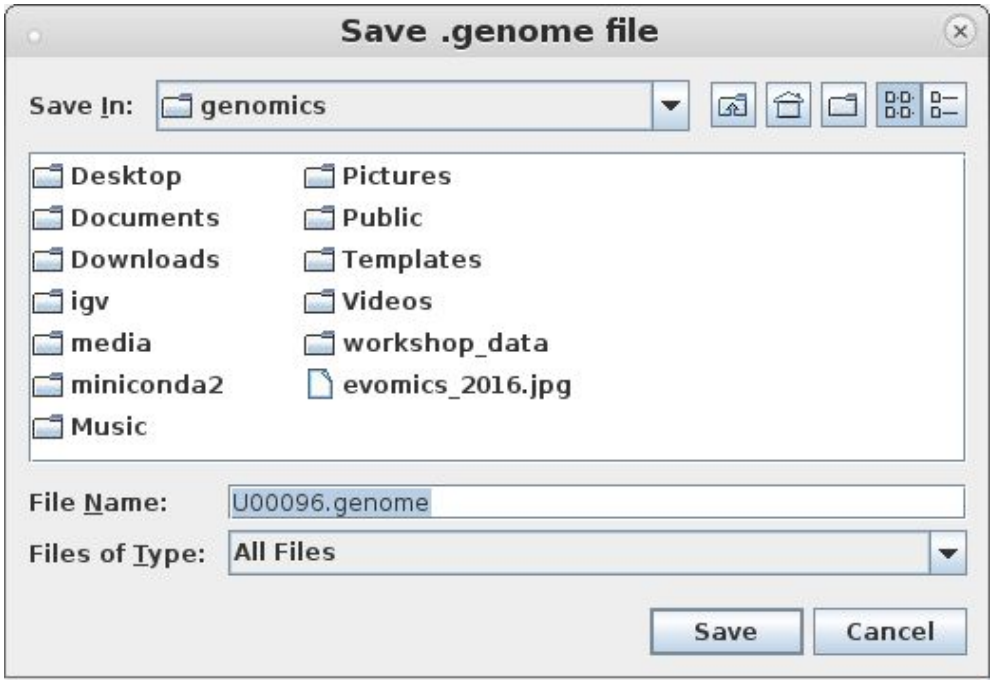
By default IGV does not contain our reference genome. We'll need to import it.

Click on 'Genomes ->Create .genome file...'



Enter the information above and click on ‘OK’ .

IGV will ask where it can save the genome file. Your home directory will be fine.



Click 'Save' again.
Note that the genome and the annotation have now been imported.

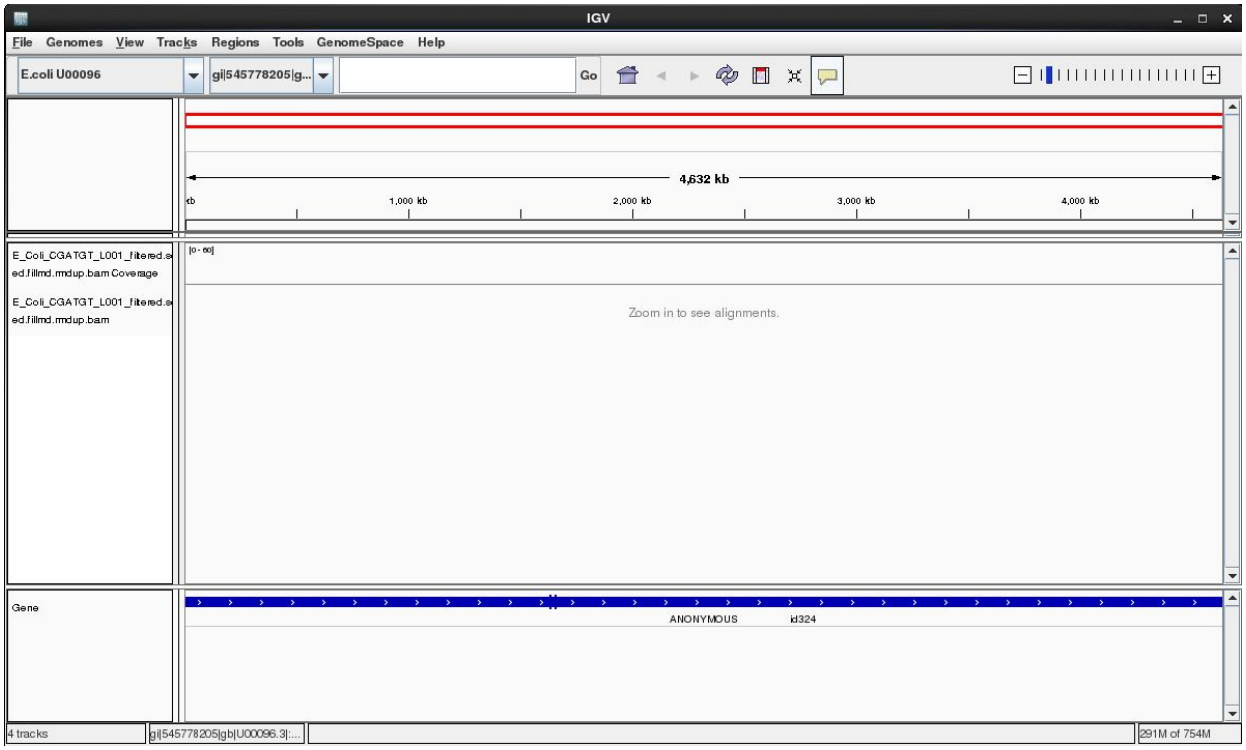
Task 12b: Load the BAM File

Load the alignment file. Note that IGV requires the .bai index file to also be in the same directory.

Select File... and Load From File

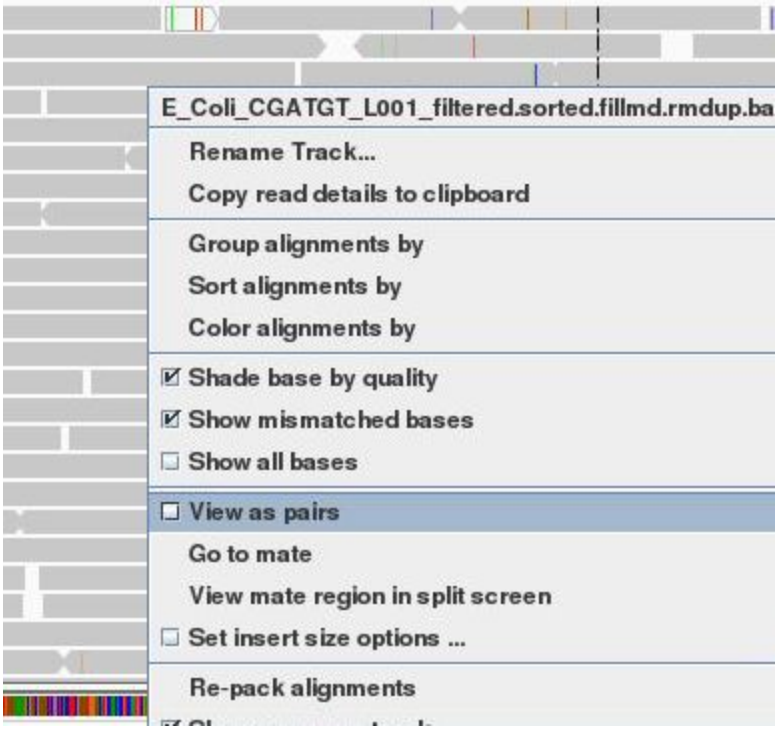
Select the bam file and click open

Once loaded your screen should look similar to the following. Note that you can load more BAM files if you wish to compare different samples or the results of different mapping programs.



Select the chromosome U00096.3 if it is not already selected

Use the +/- keys to zoom in or use the zoom bar at the top right of the screen to zoom into about 1-2kbases as above



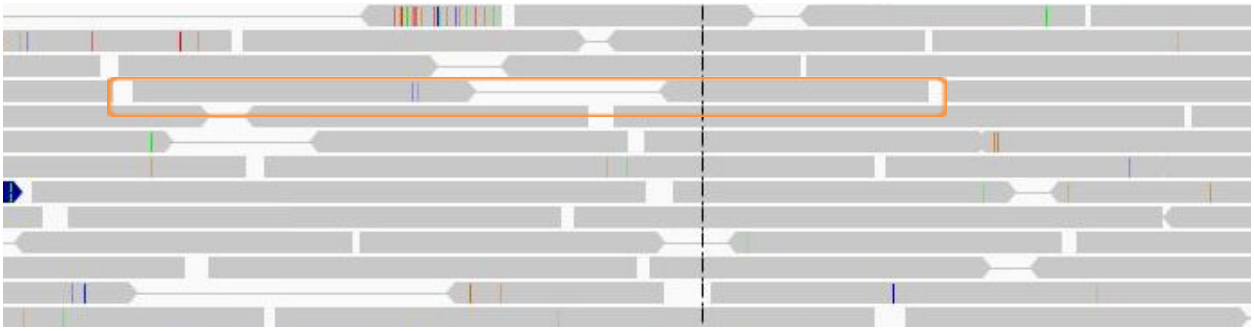
Right click on the main area and select view as pairs

The gray graph at the top of the figure indicates the coverage of the genome:

The more reads mapping to a certain location, the higher the peak on the graph. You'll see a coloured line of blue, green or red in this coverage plot if there are any SNPs (single-nucleotide polymorphisms) present (there are none in the plot). If there are any regions in the genome which are not covered by the reads, you will see these as gaps in the coverage graph. Sometimes these gaps are caused by

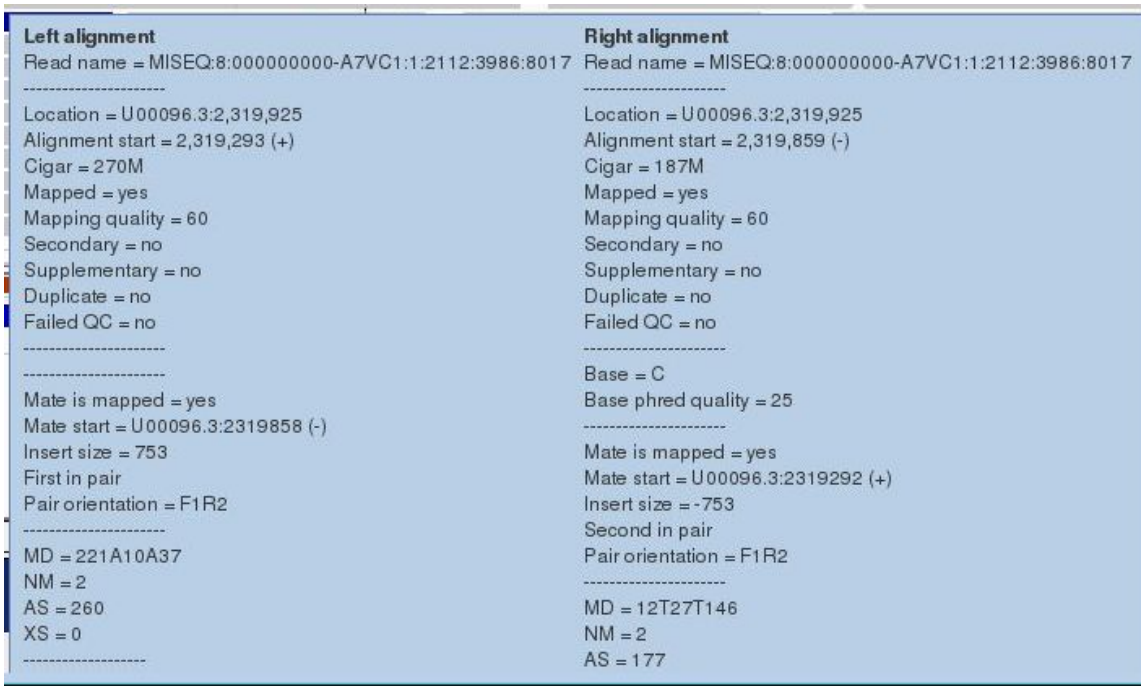
repetitive regions; others are caused by genuine insertions/deletions in your new strain with respect to the reference.

Below the coverage graph is a representation of each read pair as it is mapped to the genome. One pair is highlighted.



This pair consists of 2 reads with a gap (there may be no gap if the reads overlap) Any areas of mismatch either due to inconsistent distances between paired-end reads or due to differences between the reference and the read are highlighted by a colour. The brighter (or less transparent) the colour, the higher the base-calling quality is estimated to be. Differences in a single read are likely to be sequencing errors. Differences consistent in all reads are likely to be mutations.

Hover over a read to get detailed information about the reads' alignment:



Left alignment	**Right alignment**
Read name = MISEQ:8:000000000-A7VC1:1:2112:3986:8017	Read name = MISEQ:8:000000000-A7VC1:1:2112:3986:8017
----------	----------
Location = U00096.3:2,319,925	Location = U00096.3:2,319,925
Alignment start = 2,319,293 (+)	Alignment start = 2,319,859 (-)
Cigar = 270M	Cigar = 187M
Mapped = yes	Mapped = yes
Mapping quality = 60	Mapping quality = 60
Secondary = no	Secondary = no
Supplementary = no	Supplementary = no
Duplicate = no	Duplicate = no
Failed QC = no	Failed QC = no
----------	----------
----------	Base = C
Mate is mapped = yes	Base phred quality = 25
Mate start = U00096.3:2319858 (-)	----------
Insert size = 753	Mate is mapped = yes
First in pair	Mate start = U00096.3:2319292 (+)
Pair orientation = F1R2	Insert size = -753
----------	Second in pair
MD = 221A10A37	Pair orientation = F1R2
NM = 2	----------
AS = 260	MD = 12T27T146
XS = 0	NM = 2
----------	AS = 177

You don't need to understand every value, but compare this to the SAM format to get an idea of what is there.

SNPs and Indels

The following 3 tasks are open-ended. Please take your time with these. Read the examples on the following page if you get stuck.

Task 13: Read about the Alignment Display Format

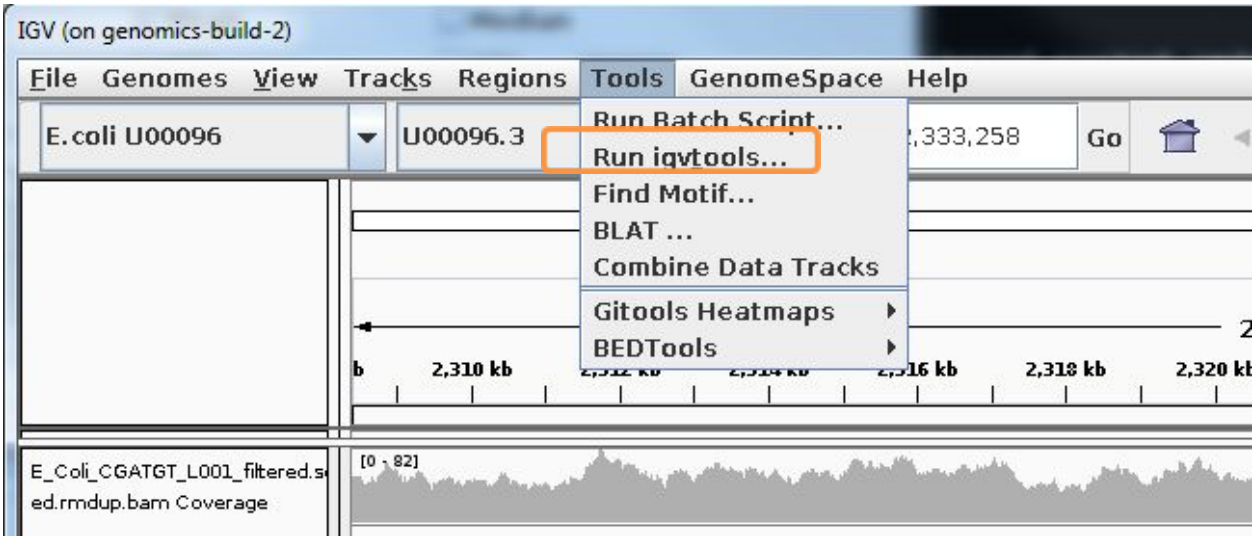
Visit http://www.broadinstitute.org/software/igv/AlignmentData

Task 14: Manually Identify a Region Without any Reads Mapping.

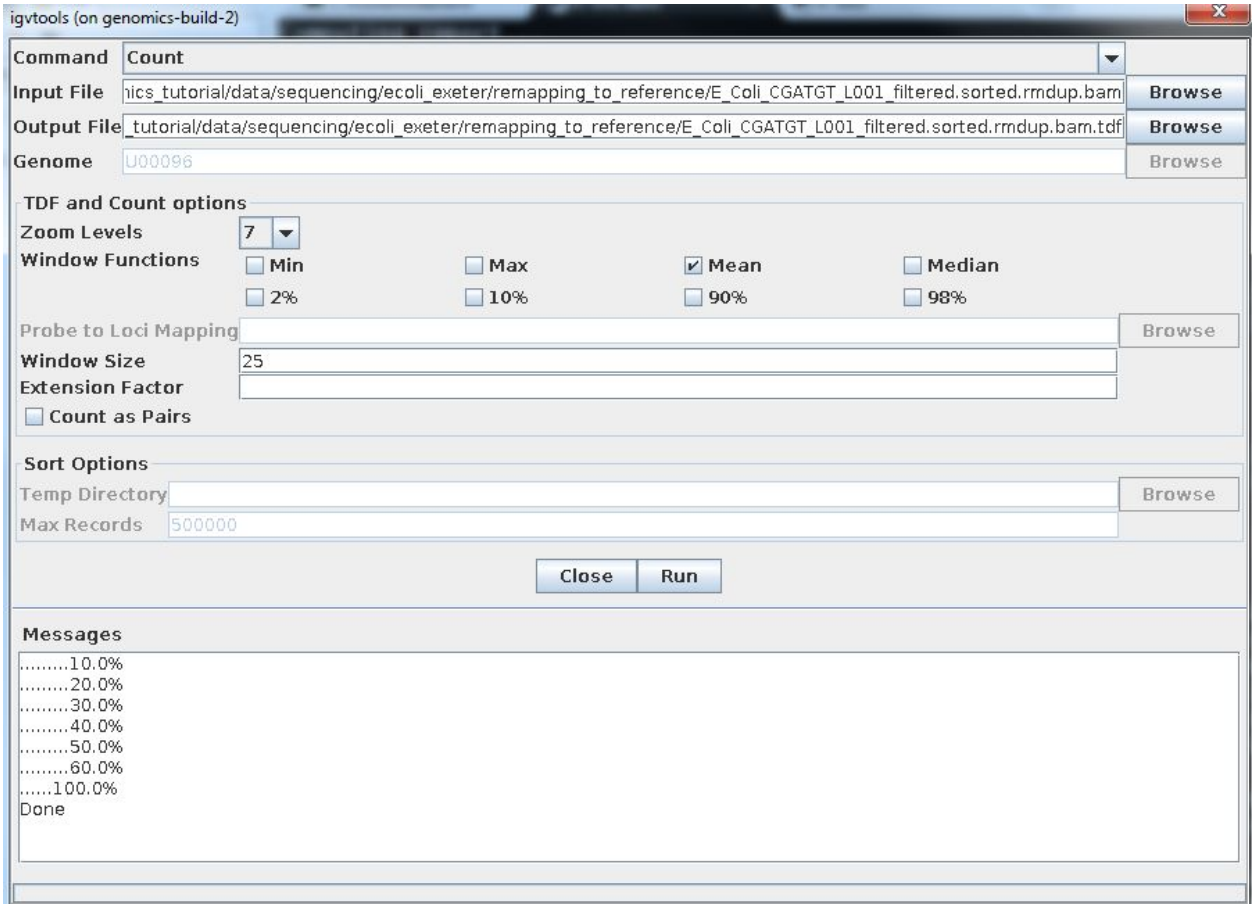
It can be quite difficult to find even with a very small genome. Zoom out as far as you can and still see the reads. Use the coverage plot from QualiMap to try to find it. Are there genes associated?

Because of the way IGV handles BAM files, it will not display coverage information if you zoom out too far. To get coverage information across the entire genome, regardless of how far you are zoomed out, you'll need to create a TDF file which contains a coverage information across windows of X number of bases across the genome. You can do this within IGV:

Select Tools->Run igvtools:



Now load the BAM alignment file in the Input field and click Run:

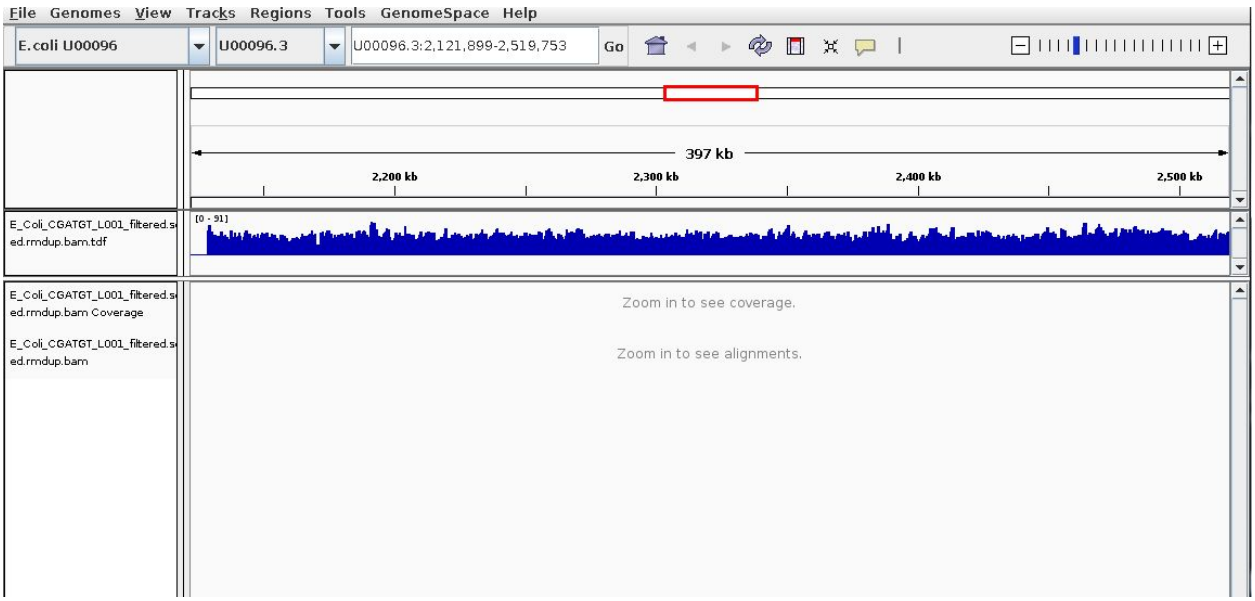


Once completed, close the igvtools window and then you can load this TDF file as by:

Select File -> Load from file…

Load the newly created tdf file

You should then see the extra coverage track which remains visible even after you zoom out.



Task 15: Identify SNPs and Indels Manually

Can you find any SNPs? Which genes (if any) are they in? How reliable do they look? (Hint – look at the number of reads mapping, their orientation - which strand they are on and how bright the base-calls are).

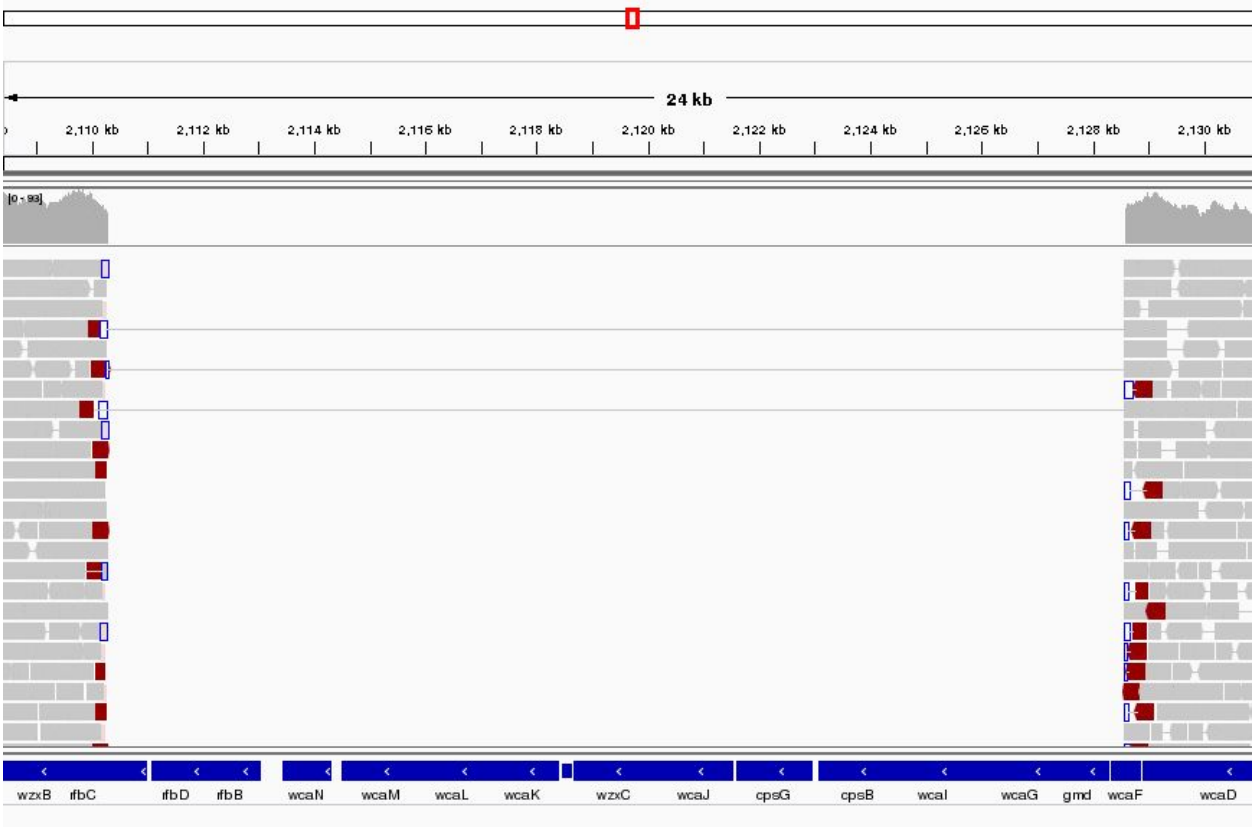
Example: Identifying Variants Manually

Here are some regions where there are differences in the organism sequenced and the reference: Can you interpret what has happened to the genome of our strain? Try to work out what is going on yourself before looking at the comment

Paste each of the genomic locations in this box and click go


```
U00096.3:2,108,392-2,133,153
U00096.3:3,662,049-3,663,291
U00096.3:4,296,249-4,296,510
U00096.3:565,965-566,489
```

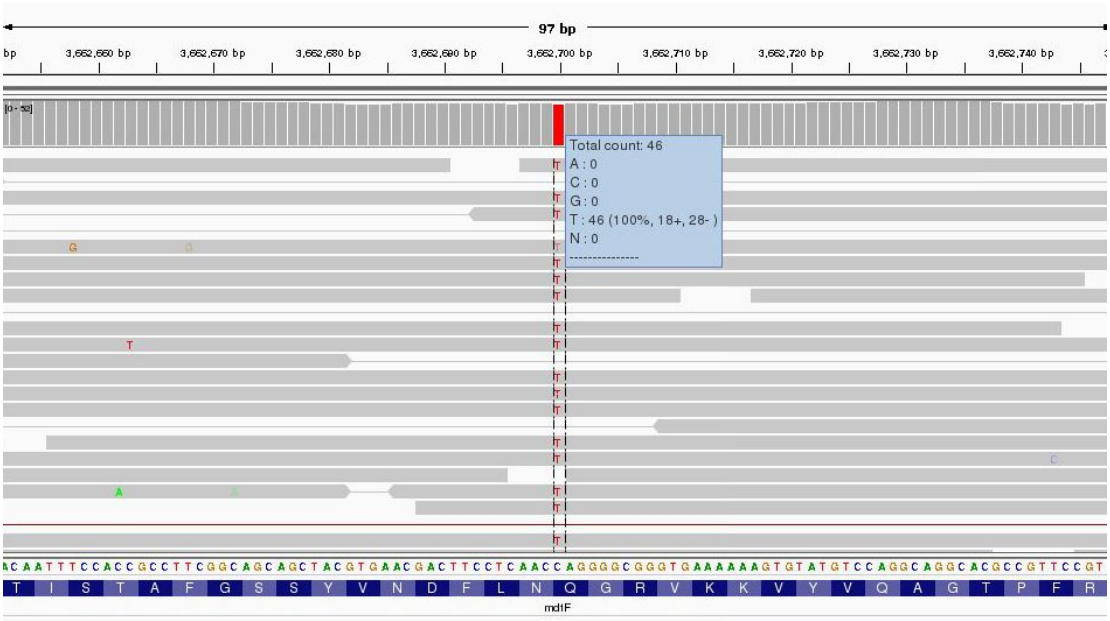
Region U00096.3:2,108,392-2,133,153



This area corresponds to the drop in coverage identified by Qualimap. It looks like a fairly large region of about 17 kbases which was present in the reference and is missing from our sequenced genome. It looks like about 12 genes from the reference strain are not present in our strain - is this real or an artefact?

Well it is pretty conclusive we have coverage of about 60X either side of the deletion and nothing at all within. There are nice clean edges to the start and end of the deletion. We also have paired reads which span the deletion. This is exactly what you would expect if the two regions of coverage were actually joined together.

Region U00096.3:3,662,049-3,663,291

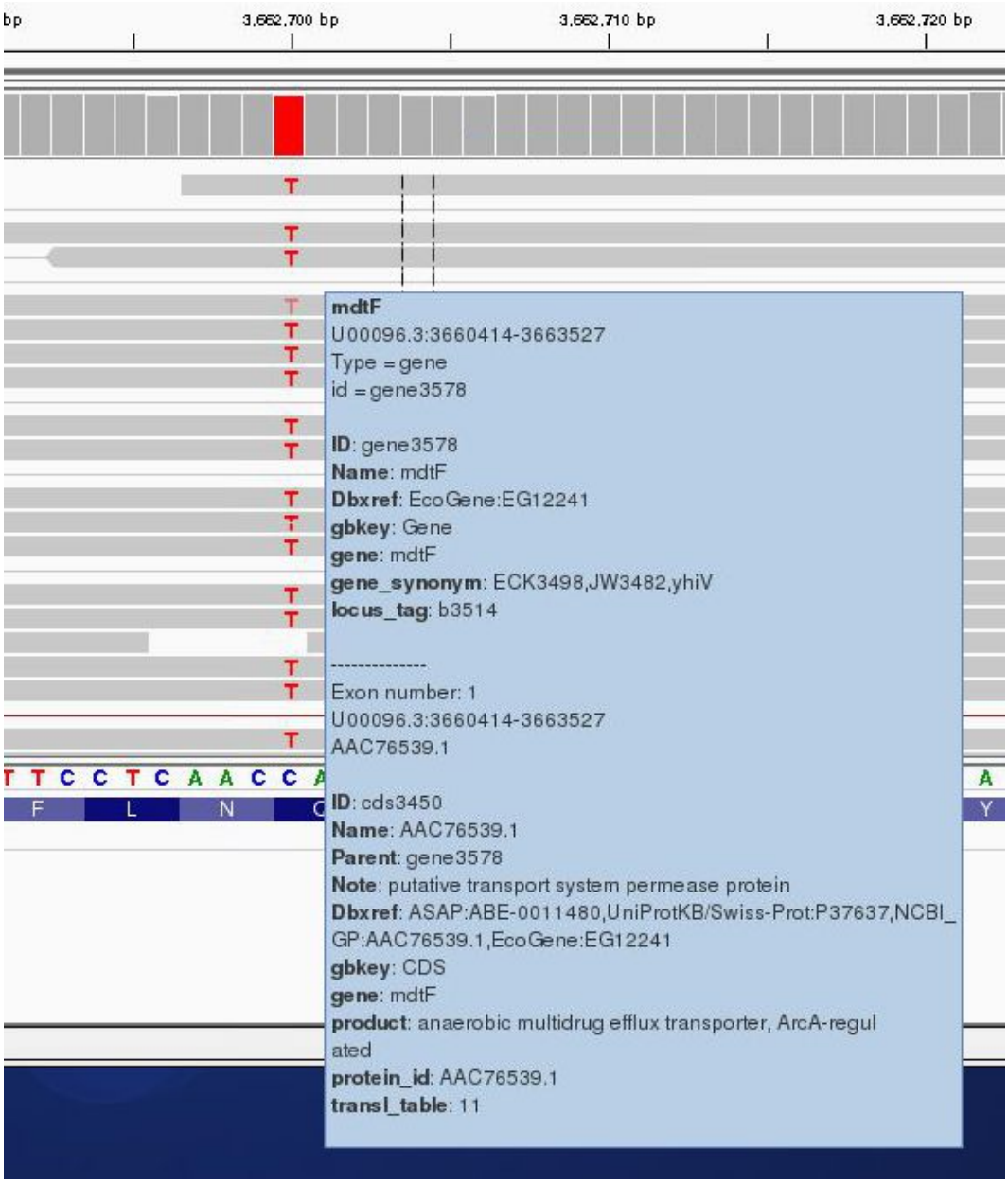


Zoom right in until you can see the reference sequence and protein sequence at the bottom of the display.

The first thing to note is that only discrepancies with respect to the reference are shown. If a read is entirely the same as its reference, it will appear entirely grey. Blue and red blocks indicate the presence of an 'abnormal' distance between paired-end reads. Note that unless this is consistent across most of the reads at a given position, it is not significant.

Here we have a C->T SNP. This changes the codon from CAG->TAG (remember to check what strand the gene is on this one is on the forward strand, if it was on the reverse strand you would have to take the reverse complement of the codon to interpret the amino acid it codes for) and results in a Gln->Stop mutation in the final protein product which is very likely to change the effect of the protein product.

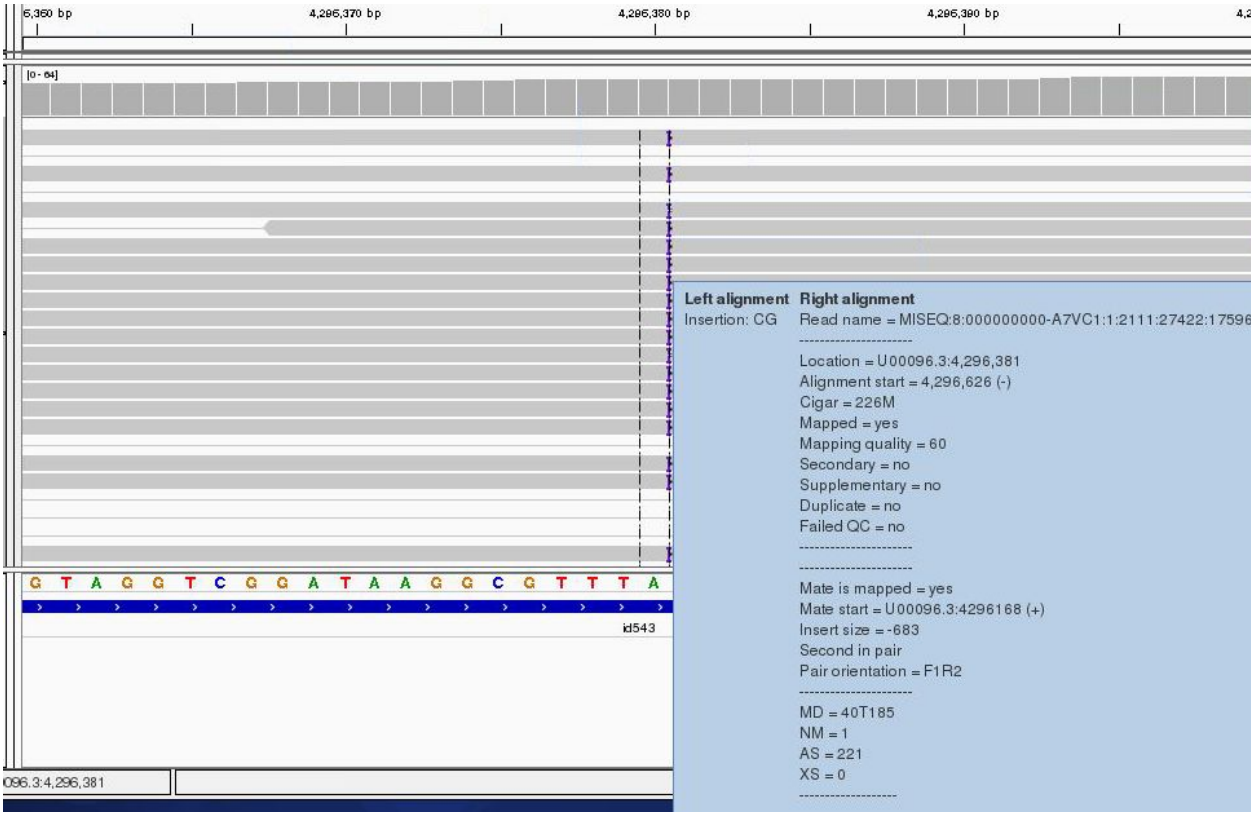
Hover over the gene to get some more information from the annotation... Since it is a drug resistance protein it could be very significant.



One additional check is that the SNPs occur when reading the forward strand. We can check this by looking at the direction of the grey reads, or by hovering over the coverage graph - see previous diagram. We can see that approximately half of the bases reporting the C->T mutation occur in read 1 (forward arrow), and half in read 2 (reverse arrow). This adds confidence to the base-call as it reduces the likelihood of this SNP being the result of a PCR duplication error.

Note that sequencing errors in Illumina data are quite common (look at the odd bases showing up in the screen above). We rely on depth of sequencing to average out these errors. This is why people often mention that a minimum median coverage of 10-20x across the genome is required for accurate SNP-calling with Illumina data. This is not necessarily true for simple organisms such as prokaryotes, but for diploid and polyploid organisms it becomes important because each position may have one, two or many alleles changed.

Regions U00096.3:4,296,332-4,296,428



Much the same guidelines apply for indels as they do for SNPs. Here we have an insertion of two bases CG in our sample compared to the reference. Again, we can see how much confidence we have that the insertion is real by checking that the indel is found on both read 1 and read 2 and on both strands.

The insertion is signified by the presence of a purple bar. Hover your mouse over it to get more details as above

We can hover our mouse over the reference sequence to get details of the gene. We can see that it occurs in a repeat region.

One can research the effect that a SNP or Indel may have by finding the relevant gene at http://www.uniprot.org/ (or google 'mdtF uniprot' in the previous case).

It should be clear from this quick exercise that trying to work out where SNPs and Indels are manually is a fairly tedious process if there are many mutations. As such, the next section will look at how to obtain spreadsheet friendly summary details of these.

Region U00096.3:565,965-566,489

This last region is more complex try to understand what genomic mutation could account for this pattern - discuss with a colleague or an instructor.

Recap: SNP/Indel Identification

1. Only changes from the reference sequence are displayed in IGV
2. Genuine SNPs/Indels should be present on both read 1 and read 2
3. Genuine SNPs/Indels should be present on both strands
4. Genuine SNPs/Indels should be supported by a good (i.e. 20-30x) depth of coverage
5. Very important mutations (i.e. ones relied upon in a paper) should be confirmed via PCR/Sanger sequencing.

Automated Analyses

Viewing alignments is useful when convincing yourself or others that a particular mutation is real rather than an artefact and for getting a feel for short read sequencing datasets. However, if we want to quickly and easily find variants we need to be able to generate lists of variants, in which gene they occur (if any) and what effect they have. We also need to know which (if any) genes are missing (i.e. have zero coverage).

Automated Variant Calling

To call variants we can use a number of packages (e.g. VarScan, GATK). However here, we will show you how to use the bcftools package which comes with samtools. First, we need to generate a pileup file which contains only locations with the variants and pass this to bcftools.

Task 16: Identify SNPs and Indels using Automated Variant Callers

Make sure you are in the directory.
`~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/remapping_to_reference`

Let's use samtools' sister software bcftools to perform SNP/indel variant calling (https://samtools.github.io/bcftools/bcftools.html)

Type bcftools in your terminal:

```
Program: bcftools (Tools for variant calling and manipulating VCFs and BCFs)
Version: 1.9 (using htslib 1.9)

Usage:   bcftools [--version|--version-only] [--help] <command> <argument>

Commands:

 -- Indexing
    index        index VCF/BCF files

 -- VCF/BCF manipulation
    annotate     annotate and edit VCF/BCF files
    concat       concatenate VCF/BCF files from the same set of samples
    convert      convert VCF/BCF files to different formats and back
    isec         intersections of VCF/BCF files
    merge        merge VCF/BCF files files from non-overlapping sample sets
    norm         left-align and normalize indels
    plugin       user-defined plugins
    query        transform VCF/BCF into user-defined formats
    reheader     modify VCF/BCF header, change sample names
    sort         sort VCF/BCF file
    view         VCF/BCF conversion, view, subset and filter VCF/BCF files

 -- VCF/BCF analysis
    call         SNP/indel calling
    consensus    create consensus sequence by applying VCF variants
    cnv          HMM CNV calling
    csq          call variation consequences
    filter       filter VCF/BCF files using fixed thresholds
    gtcheck      check sample concordance, detect sample swaps and contamination
    mpileup      multi-way pileup producing genotype likelihoods
    roh          identify runs of autozygosity (HMM)
    stats        produce VCF/BCF stats

 Most commands accept VCF, bgzipped VCF, and BCF with the file type detected
 automatically even when streaming from a pipe. Indexed VCF and BCF will work
 in all situations. Un-indexed VCF and BCF and streams will work in most but
 not all situations.
```

We will use bcftools mpileup to call SNPs/indels:

```
Usage: bcftools mpileup [options] in1.bam [in2.bam [...]]

Input options:
  -6, --illumina1.3+      quality is in the Illumina-1.3+ encoding
  -A, --count-orphans     do not discard anomalous read pairs
  -b, --bam-list FILE     list of input BAM filenames, one per line
  -B, --no-BAQ            disable BAQ (per-Base Alignment Quality)
  -C, --adjust-MQ INT     adjust mapping quality; recommended:50, disable:0 [0]
  -d, --max-depth INT     max per-file depth; avoids excessive memory usage [250]
  -E, --redo-BAQ          recalculate BAQ on the fly, ignore existing BQs
  -f, --fasta-ref FILE    faidx indexed reference sequence file
      --no-reference      do not require fasta reference file
  -G, --read-groups FILE  select or exclude read groups listed in the file
  -q, --min-MQ INT        skip alignments with mapQ smaller than INT [0]
  -Q, --min-BQ INT        skip bases with baseQ/BAQ smaller than INT [13]
  -r, --regions REG[,...] comma separated list of regions in which pileup is generated
  -R, --regions-file FILE restrict to regions listed in a file
      --ignore-RG         ignore RG tags (one BAM = one sample)
  --rf, --incl-flags STR|INT  required flags: skip reads with mask bits unset []
  --ff, --excl-flags STR|INT  filter flags: skip reads with mask bits set
                                  [UNMAP,SECONDARY,QCFAIL,DUP]
  -s, --samples LIST      comma separated list of samples to include
  -S, --samples-file FILE file of samples to include
  -t, --targets REG[,...] similar to -r but streams rather than index-jumps
  -T, --targets-file FILE similar to -R but streams rather than index-jumps
  -x, --ignore-overlaps   disable read-pair overlap detection

Output options:
  -a, --annotate LIST     optional tags to output; '?' to list []
  -g, --gvcf INT[,...]    group non-variant sites into gVCF blocks according
                          to minimum per-sample DP
      --no-version        do not append version and command line to the header
  -o, --output FILE       write output to FILE [standard output]
  -O, --output-type TYPE  'b' compressed BCF; 'u' uncompressed BCF;
                          'z' compressed VCF; 'v' uncompressed VCF [v]
      --threads INT       number of extra output compression threads [0]

SNP/INDEL genotype likelihoods options:
  -e, --ext-prob INT      Phred-scaled gap extension seq error probability [20]
  -F, --gap-frac FLOAT    minimum fraction of gapped reads [0.002]
  -h, --tandem-qual INT   coefficient for homopolymer errors [100]
  -I, --skip-indels       do not perform indel calling
  -L, --max-idepth INT    maximum per-file depth for INDEL calling [250]
  -m, --min-ireads INT    minimum number gapped reads for indel candidates [1]
  -o, --open-prob INT     Phred-scaled gap open seq error probability [40]
  -p, --per-sample-mF     apply -m and -F per-sample for increased sensitivity
  -P, --platforms STR     comma separated list of platforms for indels [all]
```

If you are running this on datasets with large numbers of samples with limited coverage where recombination is a factor, you can obtain increased sensitivity by passing all the BAM files to the variant caller simultaneously (hence the multiple BAM file options in bcftools). There are also many other options here for variant calling, which you might adjust for variant calling in real life!

Now let's use bcftools mpileup to perform variant calling.

First, let's generate a Variant Call Format (VCF) file (https://samtools.github.io/bcftools/bcftools.html#mpileup)

Type the following:

```
bcftools mpileup -f
~/workshop_materials/genomics_tutorial/data/reference/U00096/U00096.fna
E_Coli_CGATGT_L001_filtered.sorted.fixmate.position.markdup.bam >
var.raw.vcf
```

This may take 10 minutes or so and will generate a Variant Call Format (VCF) file containing the raw unfiltered variant calls for each position in the genome.

Now let's use bcftools call to call variants:

```
bcftools call -c -v --ploidy 1 -O v -o var.called.vcf var.raw.vcf
```

Note that we are asking bcftools to call assuming a ploidy of 1 and to output only the variant sites in VCF format. Using grep we can count how many sites were identified as being variant sites (i.e. sites with a potential mutation). We ask grep not to count lines beginning with a comment (#).

```
grep -v -c "^#" var.called.vcf
```

You should find 320 or so sites.

Now we just need to filter this a bit further to ensure we only retain regions where we have >90% allele frequency

We can do this using vcftools (http://vcftools.sourceforge.net/man_latest.html)

```
vcftools --minDP 10 --min-alleles 2 --max-alleles 2 --non-ref-af 0.9 --vcf
var.called.vcf --recode --recode-INFO-all --out var.called.filt
```

This will create a file called var.called.filt.vcf.recode.vcf. Once complete, view the file using the 'more' command. You should see something similar to: (lines beginning with # are just comment lines explaining the output)

```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
U00096.3        378700  .       A       C       222     .       DP=47;VDB=3.492280e-01;AF1=1;AC1=2;DP4=0,0,20,26;MQ=60;FQ=-165
U00096.3        566173  .       C       G       140     .       DP=74;VDB=1.335471e-01;RPB=-1.366788e+00;AF1=0.5;AC1=1;DP4=22,35,7,9;MQ=60;FQ=143;PV4=0.78,0.051,1,1
U00096.3        566205  .       T       C       152     .       DP=70;VDB=3.660676e-02;RPB=-2.810193e-01;AF1=0.5;AC1=1;DP4=22,31,6,9;MQ=60;FQ=155;PV4=1,1,1,1
U00096.3        566245  .       G       A       133     .       DP=67;VDB=1.726489e-02;RPB=7.739471e-01;AF1=0.5;AC1=1;DP4=22,29,5,9;MQ=60;FQ=136;PV4=0.76,1,1,0.35
U00096.3        566277  .       C       T       55      .       DP=63;VDB=3.921215e-03;RPB=2.597793e-01;AF1=0.5;AC1=1;DP4=25,28,3,6;MQ=60;FQ=58;PV4=0.49,1,1,1
U00096.3        566323  .       C       T       71      .       DP=58;VDB=6.304791e-03;RPB=2.418227e+00;AF1=0.5;AC1=1;DP4=25,23,3,6;MQ=60;FQ=74;PV4=0.47,1,1,1
U00096.3        566326  .       T       C       57      .       DP=57;VDB=5.476300e-03;RPB=2.654789e+00;AF1=0.5;AC1=1;DP4=24,23,3,6;MQ=60;FQ=60;PV4=0.47,1,1,1
U00096.3        566332  .       T       G       26      .       DP=57;VDB=3.998488e-03;RPB=2.444295e+00;AF1=0.5;AC1=1;DP4=25,22,3,7;MQ=60;FQ=29;PV4=0.3,0.32,1,1
U00096.3        566356  .       T       C       71      .       DP=60;VDB=3.343644e-02;RPB=3.626135e+00;AF1=0.5;AC1=1;DP4=25,21,3,7;MQ=60;FQ=74;PV4=0.3,0.49,1,0.13
```

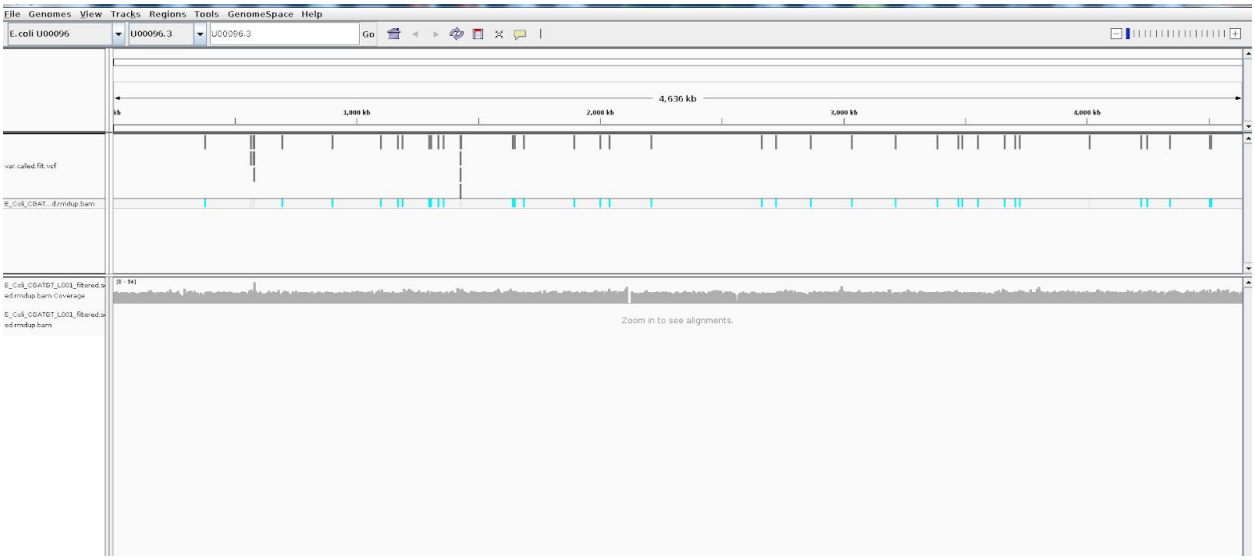
You can see the chromosome, position, reference and alternate allele as well as a quality score for the SNP. This is a VCF file (Variant Call File). This is a standard developed for the 1000 genomes project. The full specification is given at http://samtools.github.io/hts-specs/VCFv4.2.pdf

The lines starting DP and INDEL contain various details concerning the variants. For haploid organisms, most of these details are not necessary.

This forms our definitive list of variants for this sample.

Take a look at some of the variants we just excluded, was it justified? Remember there is no filter that can keep all the correct variants and remove all the dubious!

You can load the VCF file to IGV:



Task 17: Compare the Variants Found using this Method to Those You Found in the Manual Section

Can you see any variants which may have been missed? Often variants within a few bp of indels are filtered out as they could be spurious SNPs thrown up by a poor alignment. This is especially the case if you use non-gapped aligners such as Bowtie.

Quickly Locating Genes which are Missing Compared to the Reference

We can use a command from the BEDTools package (http://bedtools.readthedocs.org/en/latest/) to identify annotated genes which are not covered by reads across their full length.

Type the following on one line:

```
coverageBed -a
E_Coli_CGATGT_L001_filtered.sorted.fixmate.position.markdup.bam
~/workshop_materials/genomics_tutorial/data/reference/U00096/U00096.gff -b
> gene_coverage.txt
```

This should only take a minute or so. The output contains one row per annotated gene - the 13th column contains the proportion of the gene that is covered by reads from our sequencing. 1.00 means the gene is 100% covered and 0.00 means no coverage.

We can use the unix program sort to see which genes are missing. In this case, we are sorting by the 13 column (-k)

If we sort by this column we can see which genes are missing

```
sort -k 13 gene_coverage.txt | more
```

That concludes the first part of the course. You have successfully, QC'd, filtered, remapped and analysed a whole bacterial genome! Well done!

In the next part, we will be looking at how to extract and assemble unmapped reads. This will enable us to look at material which may be present in the strain of interest but not in the reference sequence.

Part 3: Assembly of Unmapped Reads

Introduction

In this section of the workshop we will continue the analysis of a strain of *E.coli*. In the previous section we cleaned our data, checked QC metrics, mapped our data and obtained a list of variants and an overview of any missing regions.

Now, we will examine those reads which did not map to the reference genome. We want to know what these sequences represent. Are they novel genes, plasmids or just contamination?

To do this we will extract unmapped reads, evaluate their quality, prepare them for de novo assembly, assemble them using SPAdes, generate assembly statistics and then produce some annotation via BLAST and RAST.

Extraction and QC of Unmapped Reads

Task 1: Extract the Unmapped Reads

First of all make sure you are in the `~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter` directory
Then create a directory called unmapped_assembly in which we will do our de novo assembly and analysis. Move into the unmapped_assembly directory.

Now we will use the bam2fastq program (http://gsl.hudsonalpha.org/information/software/bam2fastq) to extract from the BAM file just those reads which did NOT map to the reference genome. The bam2fastq program has a number of options, most of which are self-explanatory. Type (all on one line):

```
bam2fastq --no-aligned -o unaligned#.fastq
../remapping_to_reference/E_Coli_CGATGT_L001_filtered.sorted.fixmate.positi
on.markdup.bam
```

The --no-aligned option means only extract reads which did not align. The -o unaligned\# means dump read 1 into a file called unaligned_1.fastq and read 2 into a file unaligned_2.fastq. The program should successfully create two files.

Note that some reads were singletons (i.e. one read mapped to the reference, but the other did not). These will not be included in this analysis.

Task 2

Check that the number of entries in both fastq files is the same. Also check that the last few entries in the read 1 and read 2 files have the same header (i.e. that they have been correctly paired).

Task 3: Evaluate QC of Unmapped Reads

Use the fastqc program to look at the statistics and QC for the `unaligned_1.fastq` and `unaligned_2.fastq` files.

Do these look reasonably good? Remember, some reads will fail to map to the reference because they are poor quality, so the average scores will be lower than the initial fastqc report we did in the remapping workshop. The aim here is to see if it looks as though there are reads of reasonable quality which did not map.

Assuming these reads look ok, we will proceed with preparing them for de novo assembly.

De-novo Assembly

de novo is a Latin expression meaning "from the beginning," "afresh," "anew," "beginning again". When we perform a de novo assembly we try to reconstruct a genome or part of the genome from our reads without making any prior assumptions (in contrast to remapping where we compare our reads to what we think is a close reference sequence).

The advantage is that is that de novo assembly can reveal completely novel results, identify horizontal gene transfer events for example. The disadvantage is that it is difficult to get a good assembly from short reads and it can be prone to misleading results due to contamination and mis-assembly.

Task 4: Learn More About *de novo* Assemblers

To understand more about de-novo assemblers, read the technical note at:
https://www.illumina.com/Documents/products/technotes/technote_denovo_assembly_ecoli.pdf

N.B. You will also learn more in the next section so don't worry if it doesn't all make sense immediately. You should however understand the idea of the k-mer and broadly how the assembly is built up from them.

Task 5: Generate the Assembly

We will be using an assembler called SPAdes (http://cab.spbu.ru/software/spades/). It generally performs pretty well with a variety of genomes. It can also incorporate longer reads produced from PacBio sequencers that we will use later.

One big advantage is that it is not just a pure assembler - it is a suite of programs that prepare the reads you have, assembles them and then refines the assembly.

SPAdes runs the modules that are required for a particular dataset and it produces the assembly with a minimum of preparation and parameter selection - making it very straightforward to produce a decent assembly. As with everything in bioinformatics you should try to assess the results critically and understand the implications for further analysis.

Let's start the assembler because it takes about 20 minutes to run (this might be a nice time to get tea ☺)

```
spades.py -k 21,33,55,77,99,127 --careful -o spades_assembly -1 unaligned_1.fastq -2 unaligned_2.fastq
```

We are telling it to run the SPAdes assembly pipeline with a range of k-mer sizes (-k); specifying --careful tells it to run a mismatch correction algorithm to reduce the number of errors; put the output in the spades_assembly directory and the reads to assemble.

Just because SPAdes does a lot for you does not mean you should not try to understand the process.

Have a read of this:
http://thegenomefactory.blogspot.co.uk/2013/08/how-spades-differs-from-velvet.html
It is a discussion of how SPAdes differs from Velvet another widely used assembler, it explains the overall process nicely:

"

1. Read error correction based on k-mer frequencies using BayesHammer
2. De Bruijn graph assembly at *multiple* k-mer sizes, not just a single fixed one.
3. Merging of different k-mer assemblies (good for varying coverage)
4. Scaffolding of contigs from paired end/mate pair reads
5. Repeat resolution from paired end/mate pair data using rectangle graphs
6. Contig error correction based on aligning the original reads with BWA back to contigs

“

Try to understand the steps in the context of the whole picture:

Can you explain why error correction of reads becomes more important as k-mer length increases?
When the assembly is complete change to the `spades_assembly` directory and look at the output.

Let's take a look at some of the more important content.

params.txt

This contains a summary of the parameters used for assembly - it is useful so you can repeat the exact analysis performed, or can remember you setting when you want to publish the genome.

contigs.fasta

This contains the final results of the assembly in fasta format.

scaffolds.fasta

This contains the final results after scaffolding (which means using paired end information to join contigs together with gaps). In this case the files are identical, probably because the sum of the lengths of our paired reads is not much smaller than our insert size (there are very few large gaps between reads).

assembly_graph.fastg

Contains SPAdes assembly graph in FASTG format - this is a slightly different format that contains more information than fasta - for example it can contain alternative alleles in diploid assemblies. We don't need it here, but see http://fastg.sourceforge.net/FASTG_Spec_v1.00.pdf if you might be working with diploid organisms. You can use the Bandage (http://rrwick.github.io/Bandage/) to view this file.

Task 6: Assessment of the Assembly

We will use QUAST (http://quast.sourceforge.net/) to generate some statistics on the assembly (in the spades_assembly directory).

```
quast.py --output-dir quast scaffolds.fasta
```

This will create a directory called quast and create some statistics on the assembly you produced.

```
cat quast/report.txt
```

```
Assembly                    scaffolds
# contigs (>= 0 bp)         403
# contigs (>= 1000 bp)      14
# contigs (>= 5000 bp)      7
# contigs (>= 10000 bp)     2
# contigs (>= 25000 bp)     1
# contigs (>= 50000 bp)     1
Total length (>= 0 bp)      339986
Total length (>= 1000 bp)   131641
Total length (>= 5000 bp)   116582
Total length (>= 10000 bp)  85899
Total length (>= 25000 bp)  67361
Total length (>= 50000 bp)  67361
# contigs                   279
Largest contig              67361
Total length                286141
GC (%)                      43.27
N50                         816
N75                         550
L50                         27
L75                         143
# N's per 100 kbp          34.25
```

N.B. Your results might differ from the picture here, don't panic!

Try to interpret the information in the light of what we were trying to do. Because we are assembling unaligned reads we are not expecting a whole chromosome to pop out. We are expecting bits of our strain that does not exist in the reference we aligned against; possibly some contamination; various small contigs made up of reads that didn't quite align to our reference like repetitive or highly recombinant regions.

The N50 and L50 measures are very important in a normal assembly and we will visit them later, they are not really relevant to this assembly.

You will notice that we have 1 contig 30-60kb long - what do you think this might be? And 12 other contigs longer than 1kb. We need to find out what this is!

Analysing the *de novo* Assembled Reads

Now that we have assembled the reads and have a feel for how much (or in this case, how little) data we have, we can set about analysing it. By analysing, we mean identifying which genes are present, which organism they are from and whether they form part of the main chromosome or are an independent unit (e.g. plasmid).

We are going to take a 3-prong approach. The first will simply search the nucleotide sequences of the contigs against the NCBI non-redundant database. This will enable us to identify the species to which a given contig matches best (or most closely). The second will call open reading frames within the contigs and search those against the Swissprot database of manually curated (i.e. high quality) annotated protein sequences.

Why not just search the NCBI blast database? Well, remember nearly all of our biological knowledge is based on homology – if two proteins are similar they probably share an evolutionary history and may thus share functional characteristics. Metrics to define whether two sequences are homologous are notoriously difficult to define accurately. If two sequences share 90% sequence identity over their length, you can be pretty sure they are homologous. If they share 2% they probably aren't. But what if they share 30%? This is the notorious twilight zone of 20-30% sequence identity where it is very difficult to judge whether two proteins are homologous based on sequence alone.

To help overcome this searching more subtle signatures may help – this is where Pfam comes in. Pfam is a database which contains protein families identified by particular signatures or patterns in their protein sequence. These signatures are modeled by Hidden Markov Models (HMMs) and used to search query sequences. These can provide a high level annotation where BLAST might otherwise fail. It also has the advantage of being much faster than BLAST.

Task 7: Search Contigs against NCBI non-redundant Database

Firstly we can filter out low coverage and very short contigs using a perl script:

```
filter_low_coverage_contigs.pl < contigs.fasta > contigs.goodcov.fasta
```

The following command executes a nucleotide BLAST search (blastn) of the sequences in the contigs.fa file against the non-redundant database.

As this takes a long time to run the results have been precomputed and are available in ~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/blast_precompute/unmapped_reads/

```
blastn -db ~/workshop_materials/genomics_tutorial/db/blast/nt -query
contigs.goodcov.fasta -evalue 1e-06 -num_threads 2 -show_gis
-num_alignments 10 -num_descriptions 10 -out contigs.fasta.blastn
```

There are a lot of options in this command, let's go through them:

- **-db** is the prepared blast database to search
- -**evalue** apply an e-value (expectation value) cutoff (http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html) cutoff of 1e-06 to limit ourselves to statistically significant hits (i.e. in this case 1 in 1 million likelihood of a hit to a database of this size by a sequence of this length).

- **-num_alignments** and **-num_descriptions** flags tell blastn to only display the top 10 results for each hit,
- -**num_threads** that it should use 2 CPU cores
- **-show_gis** that it should include general identifier (GI) numbers in the output.
- **-out** file in which to place the output.

There is lots of information on running blast from the command line at http://www.ncbi.nlm.nih.gov/books/NBK1763/

N.B. GI (GeneInfo Identifiers) are being phased out by NCBI so future versions of Blast and NCBI databases will not support the –show_gis option and may break some downstream tools such as KronaTools and other databases.

Open the results file:

```
nano contigs.fasta.blastn
```

```
BLASTN 2.2.30+

Reference: Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb
Miller (2000), "A greedy algorithm for aligning DNA sequences", J
Comput Biol 2000; 7(1-2):203-14.

Database: Nucleotide collection (nt)
           29,442,065 sequences; 84,823,117,434 total letters

Query= NODE_9_length_3631_cov_29.6618_ID_17

Length=3631
                                                                      Score     E
Sequences producing significant alignments:                          (Bits)  Value

gi|549811571|gb|CP006698.1|  Escherichia coli C321.deltaA, comple...   6706   0.0
gi|383395315|gb|JQ086376.1|  Enterobacteria phage HK630, complete...   6685   0.0
gi|339305107|gb|JF340119.2|  Synthetic construct clone HO-HIS phy...   6680   0.0
gi|186702979|gb|EU421722.1|  Cloning vector lambdaS2775, complete...   6680   0.0
gi|215104|gb|J02459.1|LAMCG  Enterobacteria phage lambda, complet...   6680   0.0
gi|1066312|gb|U39286.1|CVU39286  Cloning vector TLF97-3, phage la...   6674   0.0
. . .
```

Search for our largest contig - SPAdes names the contigs by increasing size.

To do this in nano, do ctrl + w then search for NODE_1_ and press enter

```
Query= NODE_1_length_67492_cov_565.407_ID_1

Length=67492
                                                                      Score     E
Sequences producing significant alignments:                          (Bits)  Value

gi|664682453|gb|CP008801.1|  Escherichia coli KLY, complete genome    79013   0.0
gi|8918823|dbj|AP001918.1|  Escherichia coli K-12 plasmid F DNA, ...  78976   0.0
gi|619497957|gb|KJ170699.1|  Escherichia coli strain K-12 plasmid...  65330   0.0
```

```
gi|665821556|gb|KJ484626.1|  Escherichia coli plasmid pH2332-166,...  65302   0.0
gi|665821958|gb|KJ484628.1|  Escherichia coli plasmid pH2291-144,...  65213   0.0
gi|28629230|gb|AF550679.1|  Escherichia coli plasmid p1658/97, co...  64591   0.0
gi|4874241|gb|U01159.2|  Escherichia coli F sex factor transfer r...  61474   0.0
gi|665822931|gb|KJ484636.1|  Escherichia coli plasmid pC59-153, c...  41227   0.0
gi|301130432|gb|CP002090.1|  Salmonella enterica subsp. enterica ...  41026   0.0
gi|301130304|gb|CP002089.1|  Salmonella enterica subsp. enterica ...  41026   0.0
```

There are a number of good hits; notice from the contig header line that the average coverage is >500 and the coverage of our genome was around 50 - does this give you a clue to what it is?

Task 8: Obtain Open Reading Frames

The first task is to call open reading frames within the contigs. These are designated by canonical start and stop codons and are usually identified by searching for regions free of stop codons. We will use the EMBOSS package program getorf to call these.

We will use codon table 11 which defines the bacterial codon usage table (https://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi) and state that the sequences we are dealing with are not circular (they are nowhere near long enough!). We will also restrict the ORFs to just those sequences longer than 300 nucleotides (i.e. 100 amino acids). We will store the results in file contigs.orf.fa.

```
getorf -table 11 -circular N -minsize 300 -sequence contigs.goodcov.fasta
-outseq contigs.orf.fasta
```

If we look at the output file we can see that it is a FASTA formatted file containing the name of the contig on which the ORF occurs, followed by an underscore and a number (e.g. _1) to indicate the number of the ORF on that contig. The numbers in square brackets indicate the start and end position of the ORF on the contig (i.e. in nucleotide space). So in this example, the first ORF occurs on NODE 9 and is between position 934 and 1494. The third ORF occurs between positions 2400 and 2047 on the reverse strand. This is a relatively short peptide sequence and is unlikely to be a genuine peptide.

Also note that many ORFs do not start with a Methionine. This is because by default the getorf program calls ORFs between stop codons rather than start and stop codons. Primarily this is to avoid spurious ORFs due to Met residues within a protein sequence and to ensure untranslated regions are captured.

```
>NODE_9_length_3631_cov_29.6618_ID_17_1 [934 - 1494]
TERFEVSEINSQALREAAEQAMHDDWGFDADLFHELVTPSIVLELLDERERNQQYIKRRD
QENEDIALTVGKLRVELETAKSKLNEQREYYEGVISDGSKRIAKLESNEVREDGNQFLVV
RHPGKTPVIKHCTGDLEEFLRQLIEQDPLVTIDIITHRYYGVGGQWVQDAGEYLHMMSDA
GIRIKGE
>NODE_9_length_3631_cov_29.6618_ID_17_2 [2450 - 3529]
RGSEMGRRRSHERRDLPPNLYIRNNGYYCYRDPRTGKEFGLGRDRRIAITEAIQANIELF
SGHKHKPLTARINSDNSVTLHSWLDRYEKILASRGIKQKTLINYMSKIKAIRRGLPDAPL
EDITTKEIAAMLNGYIDEGKAASAKLIRSTLSDAFREAIAEGHITTNHVAATRAAKSEVR
RSRLTADEYLKIYQAAESSPCWLRLAMELAVVTGQRVGDLCEMKWSDIVDGYLYVEQSKT
```

```
GVKIAIPTALHIDALGISMKETLDKCKEILGGETIIASTRREPLSSGTVSRYFMRARKAS
GLSFEGDPPTFHELRSLSARLYEKQISDKFAQHLLGHKSDTMASQYRDDRGREWDKIEIK
>NODE_9_length_3631_cov_29.6618_ID_17_3 [2400 - 2047] (REVERSE SENSE)
FVEQILSSILNRRWEYPAFPNPSTNCFKASWTSLACVPLLKCQVHRKVSAITRKKKPPSG
GLVFFQFFNSNIGYVCMCYLRPYHPVVVAVVDVLRFDNSVEWLSIPFSCDSEVHLSSP
```

Task 9: Search Open Reading Frames against NCBI non-redundant Database

The first thing we can do with these open reading frames is to search them against the NCBI non-redundant database of protein sequences to see what they may match.

Here we will perform a BLAST search using the non-redundant (nr) database, using the blastp program and store the results in contigs.orf.blastp. We'll apply an e-value (expectation value) (http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html) cutoff of 1e-06 to limit ourselves to statistically significant hits (i.e. in this case 1 in 1 million likelihood of a hit to a database of this size by a sequence of this length). The num_alignments and num_descriptions flags tell blastp to only display the top 10 results for each hit, the num_threads tells blastp to use 2 CPU cores and show_gis tells blastp it should include general identifier (GI) numbers in the output.

First reduce the number of orfs so that we have a manageable number - this small perl program that selects 10% of the orfs.

```
reduce_fasta_10x.pl < contigs.orf.fasta > contigs.orf.small.fasta
```

Then you would type (all on one line). HOWEVER this takes several hours therefore the results have been precomputed for you in: ~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/blast_precompute/unmapped_reads/

```
blastp -db ~/workshop_materials/genomics_tutorial/db/blast/nr -query
contigs.orf.small.fasta -evalue 1e-06 -num_threads 2 -show_gis
-num_alignments 10 -num_descriptions 10 -out contigs.orf.blastp
```

Task 10: Review the BLAST Format

Open the results file with nano and search for plasmid in the text. You should find a number of hits to plasmid related proteins - one example is below - can you find any others? (Remember we only checked 10% of the orfs we found). This evidence is not conclusive, but combined with the high coverage, it is starting to look like this contig is a plasmid.

```
Query= NODE_1_length_67492_cov_565.407_ID_1_32 [31455 - 31889]

Length=145
                                                                      Score     E
Sequences producing significant alignments:                          (Bits)  Value

gi|446834068|ref|WP_000911324.1|  MULTISPECIES: pirin                  275   3e-92
gi|446834058|ref|WP_000911314.1|  pirin                                273   1e-91
gi|446834061|ref|WP_000911317.1|  pirin                                271   1e-90
gi|446834059|ref|WP_000911315.1|  pirin                                269   6e-90
```

gi|545289568|ref|WP_021572485.1|	hypothetical protein	269	6e-90
gi|446834062|ref|WP_000911318.1|	MULTISPECIES: pirin	269	6e-90
gi|585223672|ref|WP_024168023.1|	plasmid maintenance protein	268	9e-90
gi|723058272|ref|WP_033552985.1|	plasmid maintenance protein	268	9e-90
gi|446834056|ref|WP_000911312.1|	plasmid maintenance protein	268	1e-89
gi|446834060|ref|WP_000911316.1|	pirin	268	1e-89

>gi|446834068|ref|WP_000911324.1| MULTISPECIES: pirin [Escherichia]
 gi|32470009|ref|NP_862949.1| plasmid maintenance protein [Escherichia coli]
 gi|689926354|ref|YP_009060131.1| PIN domain protein [Escherichia coli]
 gi|691230621|ref|YP_009070585.1| VapC toxin protein [Escherichia coli]
 gi|28629266|gb|AAO49546.1| hypothetical protein [Escherichia coli]
 gi|323184064|gb|EFZ69443.1| PIN domain protein [Escherichia coli OK1357]
 gi|325495739|gb|EGC93600.1| plasmid maintenance protein [Escherichia fergusonii ECD227]
 gi|385154377|gb|EIF16391.1| plasmid maintenance protein [Escherichia coli O32:H37 str. P4]

Additional Checks

Task 11: Check that the Contigs do not Appear in the Reference Sequence

In theory, the unmapped reads used to generate the contigs should not assemble into something which will map against the genome. However, it is always possible (especially with more complex genomes), that this might happen. To double-check move back to the folder containing the contigs.goodcov.fasta:

```
blastn -subject
~/workshop_materials/genomics_tutorial/data/reference/U00096/U00096.fna
-query contigs.goodcov.fasta | more
```

Here we use the BLAST+ package in a different mode to compare two sequences against each other. Unlike the previous examples where we have searched against a database of sequences, here we are doing a simple search of the contigs against the reference genome we are using. Scroll down a little…

```
Query= NODE_17_length_917_cov_10.3076_ID_33

Length=917

Subject= gi|545778205|gb|U00096.3| Escherichia coli str. K-12 substr. MG1655,
complete genome

Length=4641652


 Score =   193 bits (104),  Expect = 3e-49
 Identities = 186/227 (82%), Gaps = 0/227 (0%)
 Strand=Plus/Plus

Query  68       ACGGCATCCACGAAGGCGACAGAGGCTGCGGGAAGTGCGGTATCAGCATCGCAGAGCAAA  127
                ||||||||||||||||||||||||||| || ||||||    | ||| | |||||||||
Sbjct  1430285  ACGGCATCCACGAAGGCGACAGAGGCTGCTGGCAGTGCGACGGCGGCAGCTCAGAGCAAA  1430344
```

You can see that some of the contigs that have been assembled hit the reference sequence. In the record above the e-value is 3e-49 which is massively significant; however, the e-value is calculated as the chance of a hit this close against a random sequence of the same size. Since our subject sequence is now very small and we know it is related to our strain it is not surprising that there are some hits. We are concerned about whole contigs that map closely to the reference genome.

Part 4 *De novo* Assembly Using Short Reads

Introduction

In this section of the workshop we will continue the analysis of a strain of *E.coli*. In the previous section we extracted those reads which did not map to the reference genome and assembled them. However, it is often necessary to be able to perform a de novo assembly of a genome. In this case, rather than doing any remapping, we will start with the filtered reads we obtained in part 3 of the workshop.

To do this, we will use the program SPAdes again to try to get the best possible assembly for a given genome. We will then generate assembly statistics and then produce some annotation via Pfam and BLAST.

Task 1: Start the Assembly

The assembly takes a while so the results have been **pre-computed** for you and are available in the directory:

~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/denovo_assembly

If you were to run the command it would be as follows:

```
spades.py -o denovo_assembly_rerun -1
E_Coli_CGATGT_L001_R1_001.filtered.fastq -2
E_Coli_CGATGT_L001_R2_001.filtered.fastq
```

This will create a directory called `denovo_assembly_rerun` to hold the results.

Assembly Theory

We are using SPAdes (http://cab.spbu.ru/software/spades/) to perform our assembly. It is a de Bruijn graph based assembler, similar to other short read assemblers like velvet (https://www.ebi.ac.uk/~zerbino/velvet/). The advantage of SPAdes is that it does a lot of error correction and checking before and after the assembly which improves the final result. A downside of SPAdes is that it was designed for assembling reads from a single cell and although it does a good job with DNA prepared from a community it can leave in some low coverage sequences which are likely to be artifacts.

You can read more about the comparison here
http://thegenomefactory.blogspot.co.uk/2013/08/how-spades-differs-from-velvet.html

SPAdes is also very easy to use - apart from telling it where your input files are the only parameter that you might want to choose is the length of k-mer.

K-mer length. Rather than store all reads individually which would be unfeasible for Illumina type datasets, de Bruijn assemblers convert each read to a series of k-mers and store each k-mer once, along with information about how often it occurs and which other k-mers it links to. A short k-mer length (e.g. 21) reduces the chance that data will be missed from an assembly (e.g. due to reads being shorter than the k-mer length or sequencing errors in the k-mer), but can result in shorter contigs as repeat regions cannot be resolved.

When using the Velvet assembler it is necessary to try a large combination of parameters to ensure that you obtain the 'best' possible assembly for a given dataset. There is even a program called VelvetOptimiser which does it for you. However, what 'best' actually means in the context of genome assembly is ill-defined. For a genomic assembly you want to try to obtain the lowest number of contigs, with the longest length, with the fewest errors. However, although numbers of contigs and longest lengths are easy to evaluate, it is extremely difficult to know what is or isn't an error when sequencing a genome for the first time.

SPAdes allows you to choose more than one k-mer length - it then performs an assembly for each k-mer and merges the result - trying to get the best of both worlds. It actually has some pre-calculated k-mer settings based on the length of reads you have, so you don't even have to choose that.

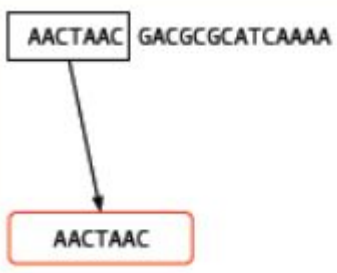
Let's look at the assembly process in more detail:

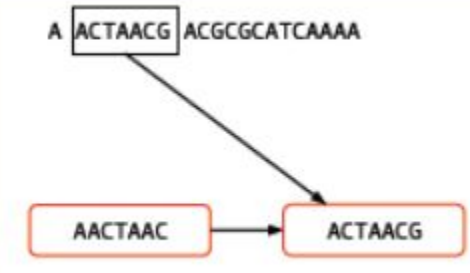
Description of k-mers:

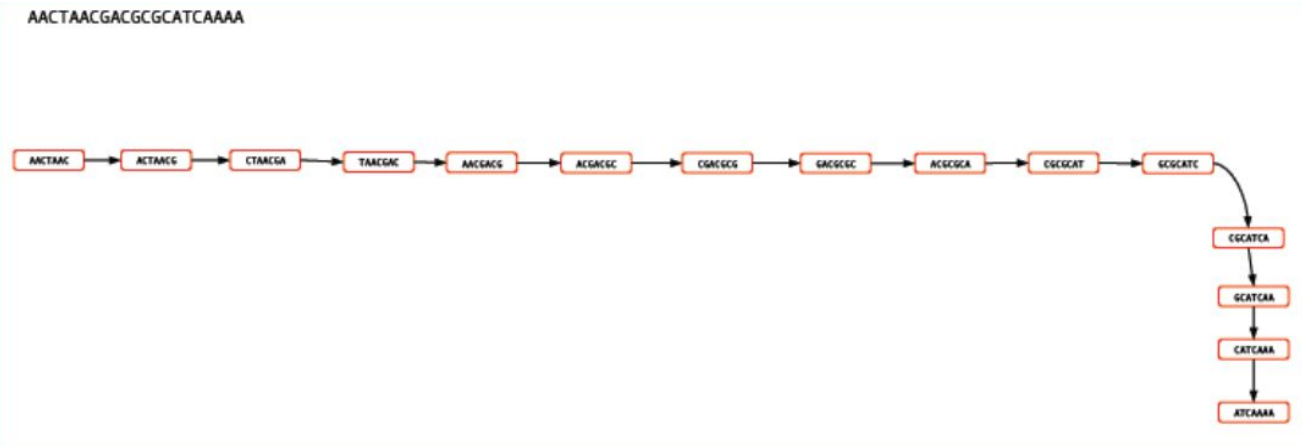
What are they? Let's say you have a single read:

```
AACTAACGACGCGCATCAAAA
```

The set of k-mers obtained from this read with length 6 (i.e. 6-mers) would be obtained by taking the first six bases, then moving the window along one base, taking the next 6 bases and so-on until the end of the read. E.g:

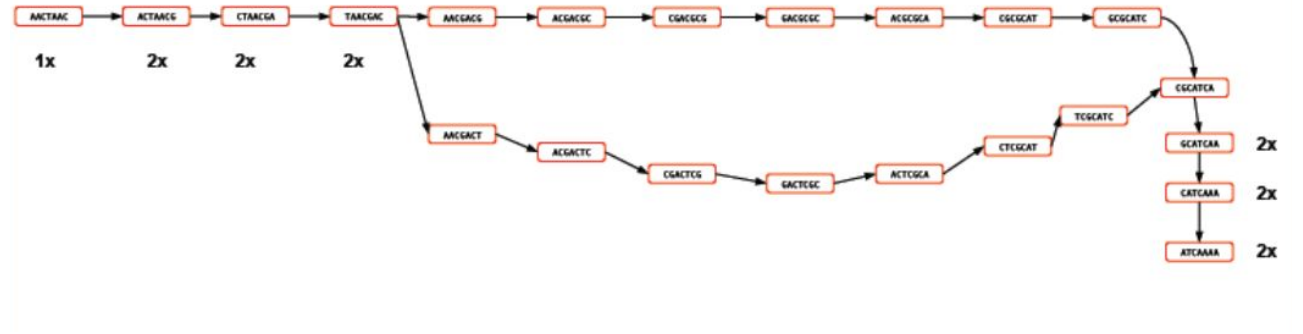




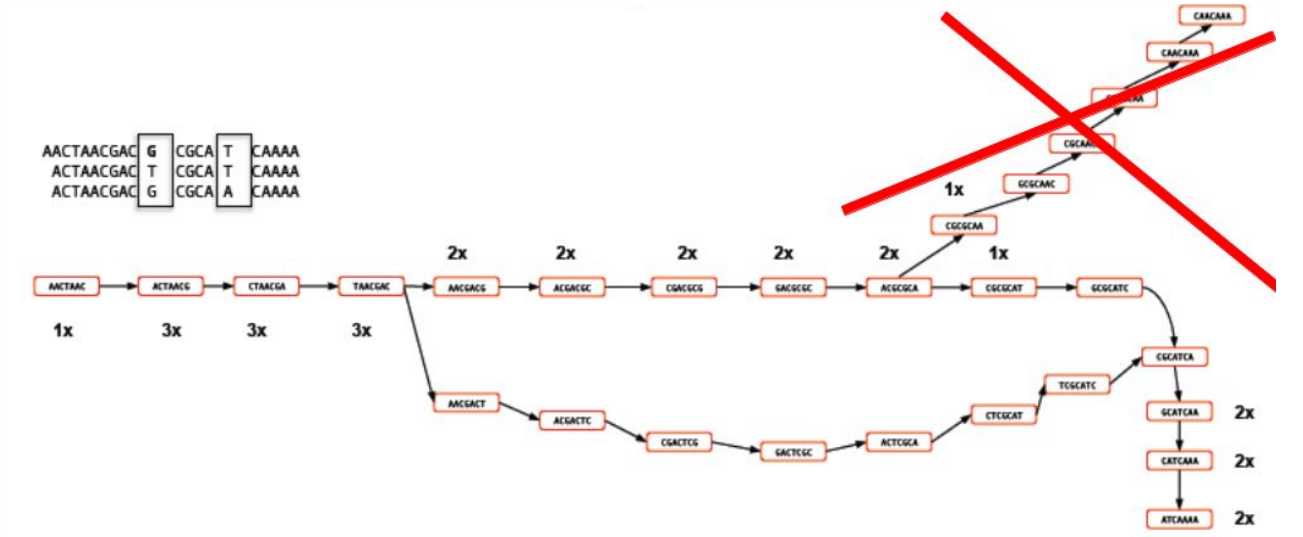


You may well ask, “So what? How does that help”? For a single read, it really doesn't help. However let's say that you have another read which is identical except for a single base:

```
AACTAACGAC G CGCATCAAAA
ACTAACGAC T CGCATCAAAA
```



Rather than represent both reads separately, we need only store the k-mers which differ and the number of times they occur. Note the 'bubble' like structure which occurs when a single base-change occurs. This kind of representation of reads is called a 'k-mer graph' (sometimes inaccurately referred to as a de-bruijn graph).



Now let's see what happens when we add in a third read. This is identical to the first read except for a change at another location. This results in an extra dead-end being added to the path.

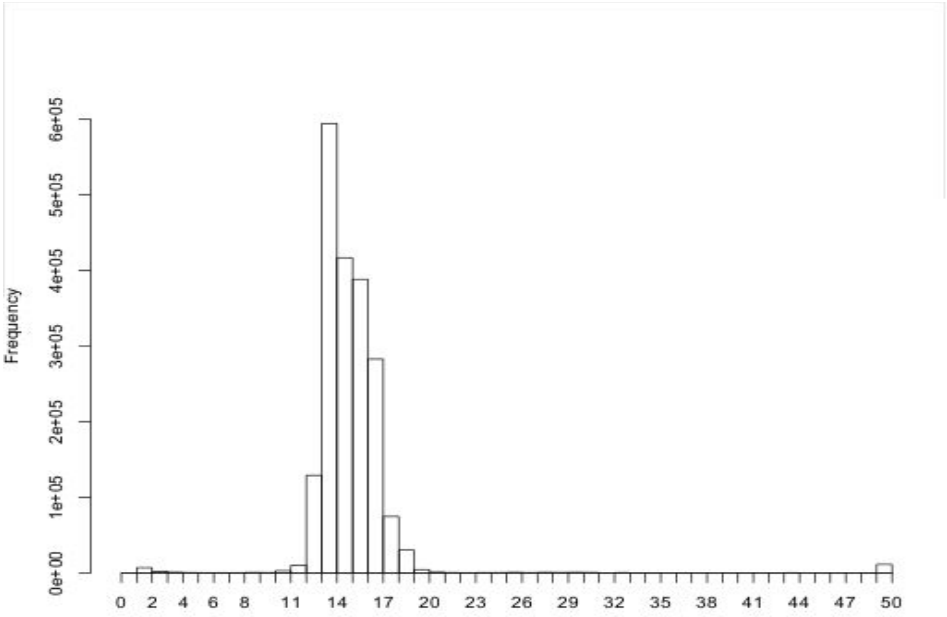
The job of any k-mer based assembler is to find a path through the k-mer graph which correctly represents the genome sequence.
Images courtesy of Mario Caccamo

Description of coverage cutoff:

In the figure above, you can see that the coverage of various k-mers varies between 1x and 3x. The question is which parts of the graph can be trimmed or removed so that we avoid any errors. As the graph stands, we could output three different contigs as there are three possible paths through the graph. However, we might wish to apply a coverage cutoff and remove the top right part of the graph because it has only 1x coverage and is more likely to be an error than a genuine variant.

In a real graph you would have millions of k-mers and thousands of possible paths to deal with. The best way to estimate the coverage cutoff in such cases is to look at the frequency plot of contig (node) coverage, weighted by length. In the example below you can see that contigs with a coverage below 7x or 8x occur very infrequently. As such it is probably a good idea to exclude those contigs which have coverage less than this – they are likely to be errors.

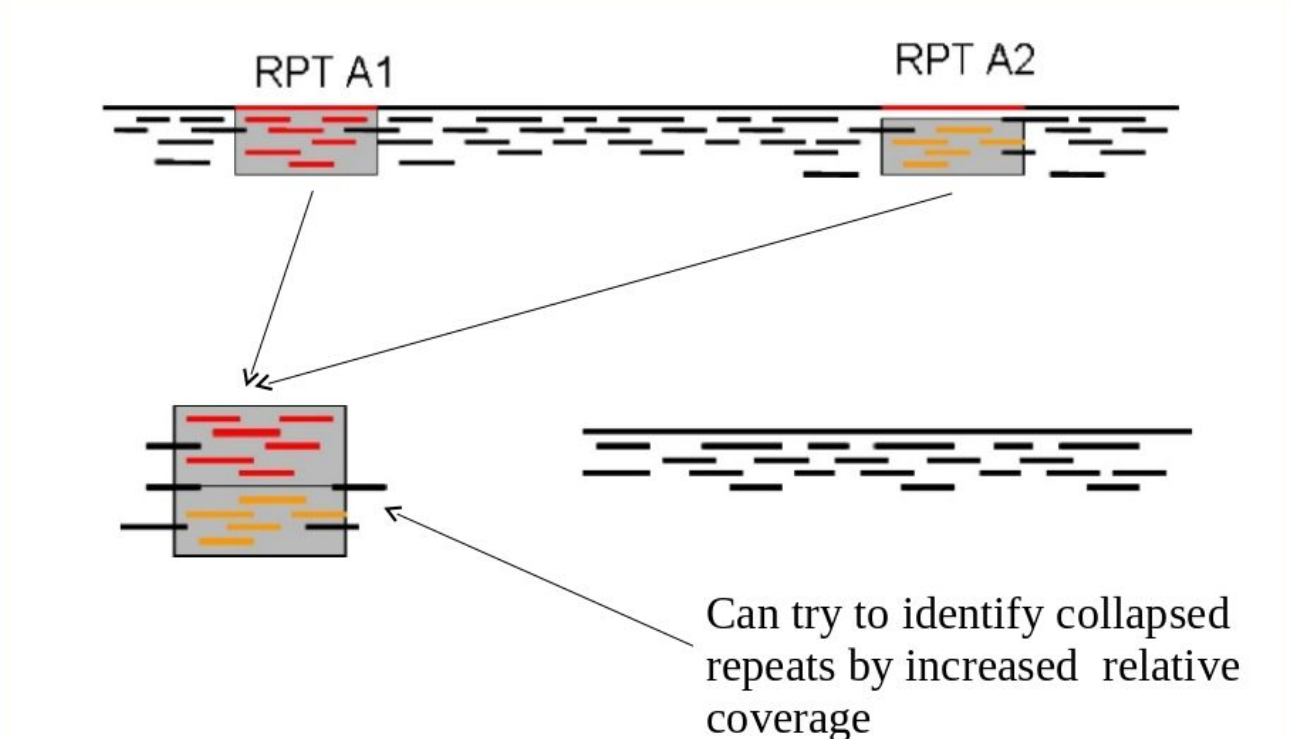
Description of expected coverage:



In the example below you can see a stretch of DNA with many reads mapping to it. There are two repetitive regions (A1 and A2) which have identical sequence. If we try to assemble the reads without any knowledge of the true DNA sequence, we will end up with an assembly that is split into two or more contigs rather than one.

One contig will contain all the reads which did not fall into A1 and A2. The other will contain reads from both A1 and A2. As such the coverage of the repetitive contig will be twice as high as that of the non-repetitive contig.

If we had 5 repeats we would expect 5x more coverage relative to the non-repetitive contig. As such, provided we know what level of coverage we expect for a given set of data, we can use this information to try and resolve the number of repeats we expect.



A commonly used metric to describe the effectiveness of the assembly is called N50 - see http://en.wikipedia.org/wiki/N50_statistic for details.

Task 2: Checking the Assembly

Change into the `denovo_assembly` directory:

```
cd denovo_assembly
```

Firstly we can filter out low coverage and very short contigs using a perl script:

```
filter_low_coverage_contigs.pl < scaffolds.fasta > scaffolds.goodcov.fasta
```

We will use QUAST again (http://quast.sourceforge.net/) to generate some statistics on the assembly.

```
quast.py --output-dir quast scaffolds.goodcov.fasta
```

This will create a directory called quast and create some statistics on the assembly you produced.

```
cat quast/report.txt
```

```
All statistics are based on contigs of size >= 5

Assembly                    scaffolds.goodcov
# contigs (>= 0 bp)         81
# contigs (>= 1000 bp)      67
# contigs (>= 5000 bp)      49
# contigs (>= 10000 bp)     46
# contigs (>= 25000 bp)     42
# contigs (>= 50000 bp)     29
Total length (>= 0 bp)      4689514
Total length (>= 1000 bp)   4679794
Total length (>= 5000 bp)   4642981
Total length (>= 10000 bp)  4623085
Total length (>= 25000 bp)  4560613
Total length (>= 50000 bp)  4090836
# contigs                   81
Largest contig              293215
Total length                4689514
GC (%)                      50.72
N50                         136627
N75                         95318
L50                         12
L75                         21
# N's per 100 kbp          0.00
```

You can see that there are 81 contigs in the assembly - so it is still far from complete.
The N50 is 136K and the N75 is 95K so most of the assembly is in quite large contigs.

This is fairly normal for a short read assembly - don't expect complete chromosomes.

A good check at this point is to map the original reads back to the contigs.fasta file and check that all positions are covered by reads. Amazingly it is actually possible for de-novo assemblers to generate contigs to which the original reads will not map.

Task 3: Map Reads Back to Assembly

Here we will use BWA again to index the scaffolds.fasta file and remap the reads. This is almost identical to the procedure we followed during the alignment section, the only difference is that instead of aligning to the reference genome, we are aligning to our newly created reference.

Make sure you are in the following directory:
~/workshop_materials/genomics_tutorial/data/sequencing/ecoli_exeter/denovo_assembly/

Let's create a subdirectory to keep our work separate

```
mkdir remapping_to_assembly
cd remapping_to_assembly
cp ../scaffolds.fasta .
```

Let's start by indexing the contigs.fasta file. Type:

```
bwa index scaffolds.fasta
```

Once complete we can start to align the reads back to the contigs. Type (all on one line):

```
bwa mem -t 2 scaffolds.fasta ../../E_Coli_CGATGT_L001_R1_001.filtered.fastq
../../E_Coli_CGATGT_L001_R2_001.filtered.fastq >
E_Coli_CGATGT_L001_filtered.sam
```

Once complete we can convert the SAM file to a BAM file:

```
samtools view -bS E_Coli_CGATGT_L001_filtered.sam >
E_Coli_CGATGT_L001_filtered.bam
```

And then we can sort the BAM file:

```
samtools sort -o E_Coli_CGATGT_L001_filtered.sorted.bam
E_Coli_CGATGT_L001_filtered.bam
```

Once completed, we can index the BAM file:

```
samtools index E_Coli_CGATGT_L001_filtered.sorted.bam
```

We can then (at last!) obtain some basic summary statistics using the samtools flagstat command:

```
samtools flagstat E_Coli_CGATGT_L001_filtered.sorted.bam
```

```
1269338 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
622 + 0 supplementary
0 + 0 duplicates
1266061 + 0 mapped (99.74% : N/A)
1268716 + 0 paired in sequencing
634358 + 0 read1
634358 + 0 read2
1252446 + 0 properly paired (98.72% : N/A)
1264352 + 0 with itself and mate mapped
1087 + 0 singletons (0.09% : N/A)
8962 + 0 with mate mapped to a different chr
7791 + 0 with mate mapped to a different chr (mapQ>=5)
```

We can see here that very few of the reads do not map back to the contigs. Importantly 99% of reads are properly paired which gives us some indication that there are not too many mis-assemblies.

Run qualimap to get some more detailed information (and some images)

```
qualimap bamqc -outdir bamqc -bam E_Coli_CGATGT_L001_filtered.sorted.bam
```

```
firefox bamqc/qualimapReport.html
```

In the Chromosome stats section:

Chromosome stats

Name	Length	Mapped bases	Mean coverage	Standard deviation
NODE_1_length_293215_cov_26.7248_ID_1	293215	15962208	54.4386	11.386
NODE_2_length_235405_cov_25.9929_ID_3	235405	12493267	53.0714	10.9025
NODE_3_length_229124_cov_26.8329_ID_5	229124	11966638	52.2278	10.4106
NODE_4_length_227801_cov_26.3369_ID_7	227801	11934749	52.3911	12.4527
NODE_5_length_207566_cov_25.5821_ID_9	207566	10796431	52.0144	16.3361
NODE_6_length_203932_cov_24.002_ID_11	203932	10099128	49.522	9.0934
NODE_7_length_194067_cov_27.4679_ID_13	194067	10435540	53.7729	11.2855
NODE_8_length_185369_cov_26.5954_ID_15	185369	9974734	53.8102	10.7684
NODE_9_length_184921_cov_26.3007_ID_17	184921	9891106	53.4883	10.9106

The larger of our contigs have a mean coverage of around 50 - which is what we would expect from our original alignment.

NODE_25_length_67492_cov_567.168_ID_49	67492	78055078	1,156.51	225.09

There is one contig which has the size of 67492 - this is exactly the same as the contig we found in the unmapped reads - that is pretty good indication that it is a separate sequence (remember we suspected a plasmid) and not integrated into the chromosome. Also, look at its coverage!

Let's double check that by blasting these contigs against the unmapped assembly contigs from part 4:

```
blastn -subject ../scaffolds.goodcov.fasta -query
../../unmapped_assembly/spades_assembly/scaffolds.fasta >
check_plasmid.blastn
```

View the file using more or a text editor:

```
more check_plasmid.blastn
```

and about 30% of the way down the file you should find: (hint you can use nano Ctrl + w)

```
Query= NODE_1_length_67492_cov_601.94_ID_2528

Length=67492

Subject= NODE_25_length_67492_cov_567.168_ID_49

Length=67492

 Score = 1.246e+05 bits (67474),  Expect = 0.0
 Identities = 67486/67492 (99%), Gaps = 0/67492 (0%)
 Strand=Plus/Plus
```

This shows us that this contig exactly almost matches that in the unmapped assembly, strongly supporting that this is a plasmid sequence and not integrated into the chromosomes.

Task 4: View Assembly in IGV

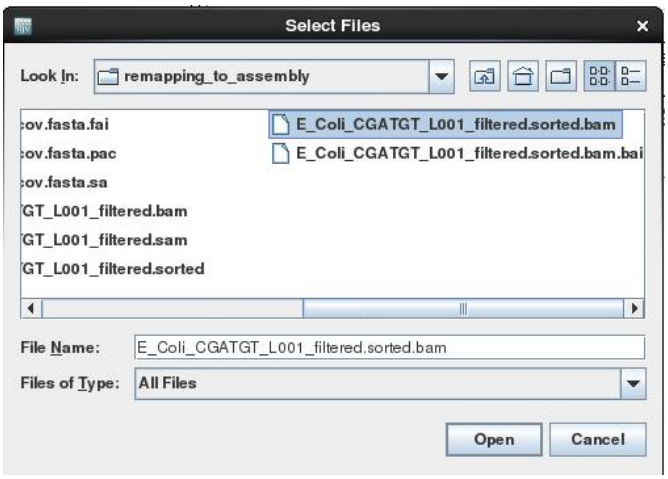
Load up IGV from the console or desktop

```
igv.sh
```

Click Genomes -> Load Genome from File….

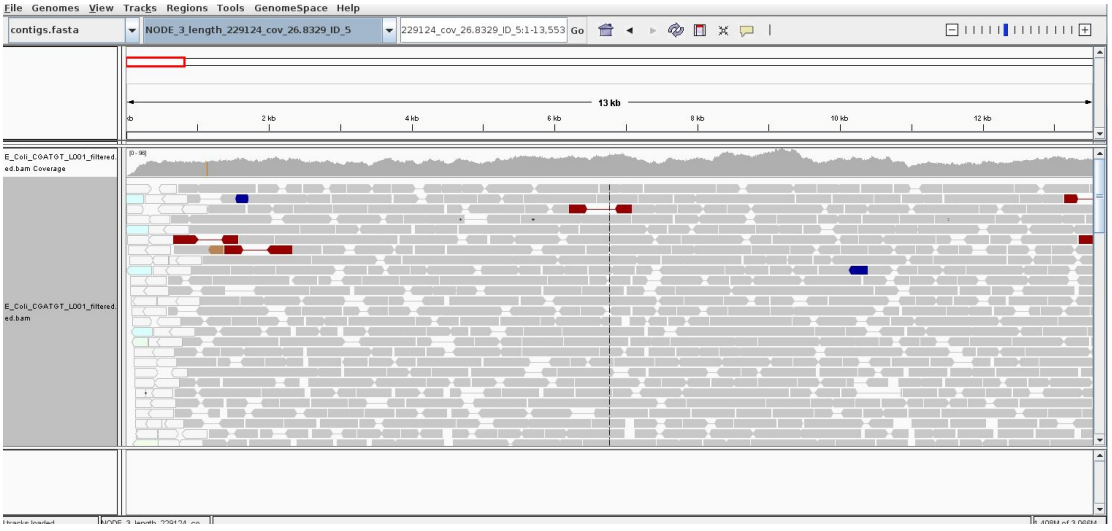
We are going to import the contigs we have assembled as the reference. Unlike the reference genome though, we have no annotation available. Make sure you select the `scaffolds.goodcov.fasta` file for the complete de novo assembly (not the unmapped reads assembly).

Once loaded, click on File->Load From File… select the `E_Coli_CGATGT_L001_filtered.sorted.bam` file. Again, make sure you load the file in the **`remapping_to_assembly`** directory.



Once loaded, explore some of the contigs in IGV.
See if you can find anything unusual in any of the contigs.

Here is one to get you started.
Select **NODE_3**…



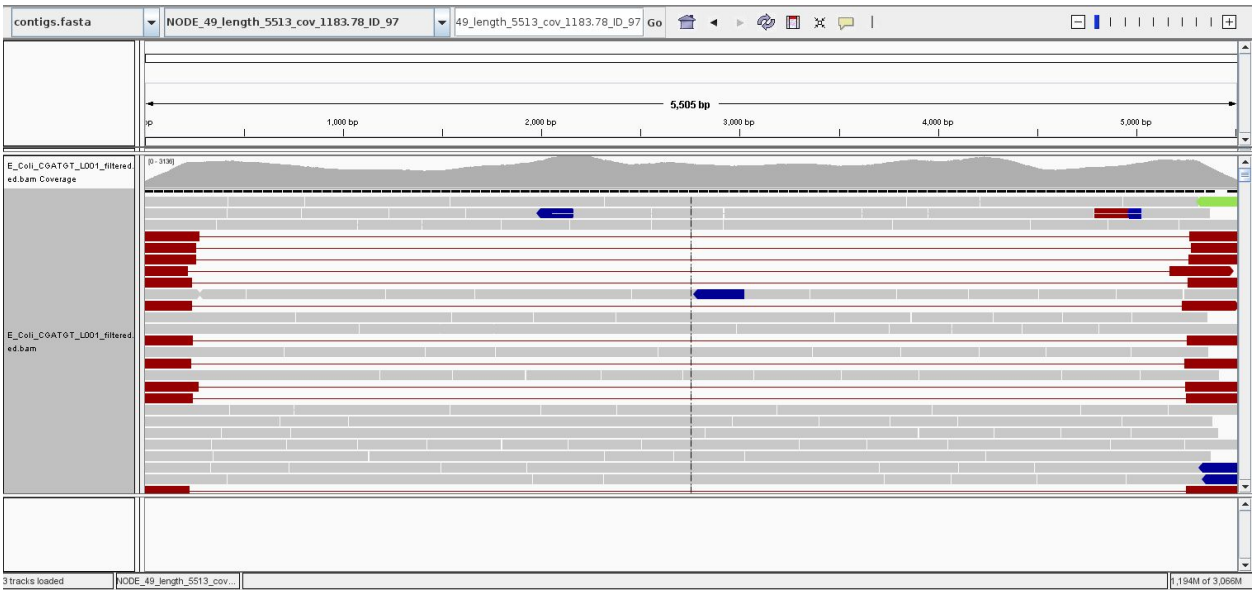
Why does the contig start and end in repetitive sequence (indicated by the white reads)? You may need to zoom in to see the details. Think about what an assembler will do if it cannot uniquely assign reads.

If an assembler cannot resolve these repetitive regions with paired-end reads or coverage information, it will generally be unable to assemble any further sequence for that contig. Therefore, it is quite common to see contigs which start and end in sequence which is repeated elsewhere.

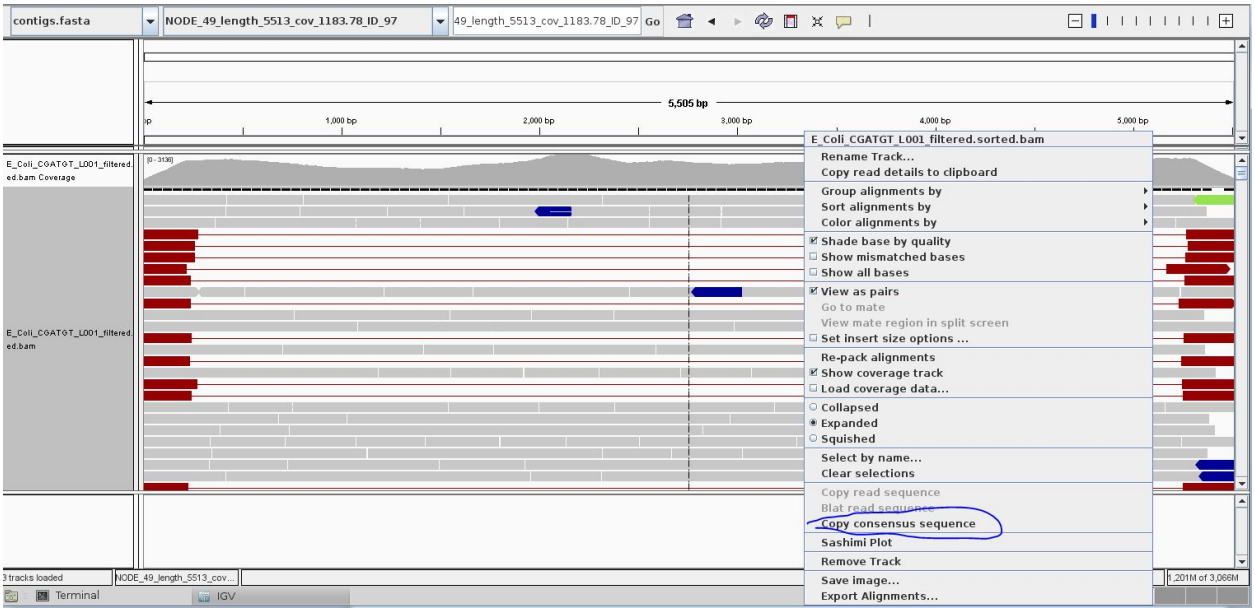
Here is another:

Select NODE_49…..

Right click on the reads and select view as pairs:



What do you think is going on here? Try blasting the contig sequence using BlastX at http://blast.ncbi.nlm.nih.gov/Blast.cgi to identify which genes the contig contains. To obtain the sequence you can right click and select 'Copy consensus sequence':



You can also do the same for individual reads, but you need to un-select ‘View as pairs’ before right clicking on a read. You may lose track of the paired-end reads and find it easier to copy the read name before un-selecting ‘View as pairs’ and then pasting it into the ‘Select by name…’ search box.

You should find that the contig contains at least two phage genes. There appear to be at least two phages present, one which seems to be the full contig, the other with the red read-pairs seems to be missing the sequence in the middle of the contig.

Annotation of *de novo* Assembled Contigs

We will now annotate the contigs using BLAST and Pfam as with the unmapped contigs.

Task 5: Obtain Open Reading Frames

As before, we’ll call open reading frames within the de-novo assembly. Again, we will use codon table 11 which defines the bacterial codon usage table (http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi) and state that the sequences we are dealing with are not circular. We will also restrict the ORFs to just those sequences longer than 300 nucleotides (i.e. 100 amino acids). We will store the results in file contigs.orf.fasta.

Make sure you are in the denovo_assembly/ directory:

```
getorf -table 11  -circular N -minsize 300 -sequence
scaffolds.goodcov.fasta -outseq scaffolds.orf.fasta
```

Hybrid *de novo* Assembly

You will have seen that even with good coverage and a relatively long (300bp) paired end Illumina dataset - the assembly we get is still fairly fragmented. Our *E.coli* example assembles into 78 contigs and the largest contig is around 10% of the genome size.

Why is this?

One possible reason would be that regions of the original genome were not sequenced, or sequenced at too low coverage to assemble correctly. Regions of the genome will occur with different frequencies in the library that was sequenced - You can see this in the variation of coverage when you did the alignment. This can be due to inherent biases in the preparation and the random nature of the process.

However, as coverage increases, the chances of not sequencing a particular region of the genome reduces, and the most significant factor becomes the resolution of repeats within the assembly process. If two regions contain the same or very similar sequences the assembler cannot reliably detect that they are actually two or more distinct sequences and incorrectly 'collapses' the repeat into

a single sequence. The assembler is now effectively missing a sequence and therefore breaks in the assembly occur.

One resolution to this is to use a sequencing technology like PacBio which can produce longer reads - the reads are then long enough to include the repeated sequence, plus some unique sequence, and the problem can then be resolved.

An approach becoming more and more popular (now, a standard for genome assembly) is to combine technologies. For example: high quality Illumina sequencing to get the accuracy of reads combined with low quality PacBio sequencing to enable the repeats to be spanned and correctly resolved.

Our exercise will be to use Illumina and PacBio datasets to assemble a species of pseudomonas. These are subsets of data used in "Evaluation and validation of de novo and hybrid assembly techniques to derive high-quality genome sequences" Utturkar et al., 2014. (http://www.ncbi.nlm.nih.gov/pubmed/24930142). This paper also contains a good explanation of the process and different approaches that are available.

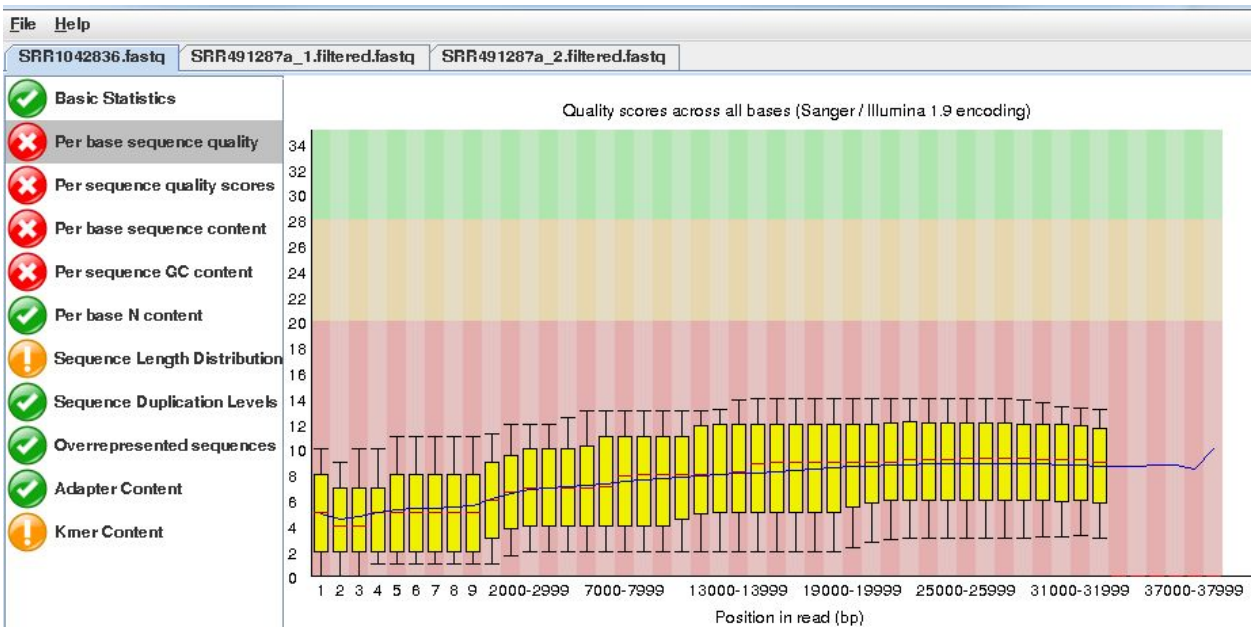
Task 6: QC the Data

It is always important to check and understand the quality of the data you are working with: Change to the directory and run `fastqc`:

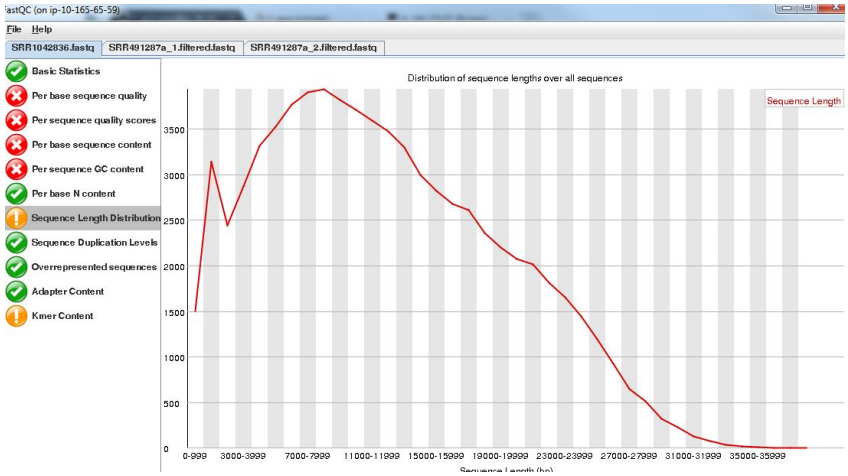
```
cd ~/workshop_materials/genomics_tutorial/data/sequencing/pseudomonas_gm41
```

```
fastqc
```

Open the files `SRR1042836a.fastq SRR491287a_1.fastq SRR491287a_2.fastq` and look at the reports generated.



Note that the quality of the PacBio reads (`SRR1042836a.fastq`) is much lower than the Illumina reads with a greater than 1 chance in 10 of there being a mistake for most reads.



However, importantly, the length of the PacBio reads is much longer.

Trim the Illumina reads using the program fastq-mcf:

```
fastq-mcf ../../reference/adaptors/adaptors.fasta SRR491287a_1.fastq
SRR491287a_2.fastq -o SRR491287a_1.filtered.fastq -o
SRR491287a_2.filtered.fastq -q 20 -p 10 -u -x 0.01
```

You can check the number of filtered reads using `grep -c` and the quality if trimmed reads with fastqc if you want.

For this exercise we want the long reads from PacBio even though they are low quality. We are relying on the assembler to use them appropriately.

Task 7: Illumina Only Assembly

Firstly let's construct an assembly using only the available Illumina data.
Make sure you are in the directory:
`~/workshop_meterials/genomics_tutorial/data/sequencing/pseudomonas_gm41`

The next command will take some time so the data has been precomputed and is available in illumina_assembly/

```
spades.py --threads 2 --careful -o illumina_only_assembly -1
SRR491287a_1.filtered.fastq -2 SRR491287a_2.filtered.fastq
```

Change to the directory illumina_only_assembly

Filter out low coverage and very short contigs using a perl script:

```
filter_low_coverage_contigs.pl < scaffolds.fasta > scaffolds.goodcov.fasta
```

Let's look at the metrics for the assembly.

```
quast.py --output-dir quast scaffolds.goodcov.fasta
```

```
cat quast/report.txt
```

```
Assembly                    scaffolds.goodcov
# contigs (>= 0 bp)         149
# contigs (>= 1000 bp)      134
# contigs (>= 5000 bp)      113
# contigs (>= 10000 bp)     103
# contigs (>= 25000 bp)     74
# contigs (>= 50000 bp)     46
Total length (>= 0 bp)      6632341
Total length (>= 1000 bp)   6621581
Total length (>= 5000 bp)   6569639
Total length (>= 10000 bp)  6499338
Total length (>= 25000 bp)  6001845
Total length (>= 50000 bp)  4940758
# contigs                   149
Largest contig              241647
Total length                6632341
GC (%)                      59.00
N50                         87433
N75                         49685
L50                         24
L75                         47
# N's per 100 kbp          12.14
```

(Your results may be slightly different. This is because spades uses a random seed that changes every time)

Task 8: Create Hybrid Assembly

Now will execute the same command, but this time include the longer PacBio reads to see the effect it has on our assembly.

Change back into the directory

Run (This may take some time so the data has been precomputed and is available in `hybrid_assembly/` if you are impatient!):

```
spades.py --threads 2 --careful -o hybrid_assembly --pacbio
SRR1042836a.fastq -1 SRR491287a_1.filtered.fastq -2
SRR491287a_2.filtered.fastq
```

Change to the directory hybrid_assembly

Filter out low coverage and very short contigs using a perl script:

```
filter_low_coverage_contigs.pl < scaffolds.fasta > scaffolds.goodcov.fasta
```

Let's look at the metrics for the assembly - this time we will compare it with the illumina only assembly:

```
quast.py --output-dir quast scaffolds.goodcov.fasta
../illumina_only_assembly/scaffolds.goodcov.fasta
```

```
cat quast/report.txt
```

Assembly	hybrid_assembly_scaffolds.goodcov	illumina_only_assembly_scaffolds.goodcov
# contigs (>= 0 bp)	85	149
# contigs (>= 1000 bp)	75	134
# contigs (>= 5000 bp)	69	113
# contigs (>= 10000 bp)	65	103
# contigs (>= 25000 bp)	53	74
# contigs (>= 50000 bp)	47	46
Total length (>= 0 bp)	6667288	6632341
Total length (>= 1000 bp)	6660729	6621581
Total length (>= 5000 bp)	6642787	6569639
Total length (>= 10000 bp)	6614698	6499338
Total length (>= 25000 bp)	6425214	6001845
Total length (>= 50000 bp)	6178935	4940758
# contigs	85	149
Largest contig	536255	241647
Total length	6667288	6632341
GC (%)	59.00	59.00
N50	151622	87433
N75	89848	49685
L50	15	24
L75	30	47
# N's per 100 kbp	4.29	12.14

You can also explore the interactive html report:

```
firefox quast/report.html
```

It seems that using the longer reads has improved the completeness of our assembly - reducing the number of contigs.

Task 9: Align Reads Back to Reference

Let's realign our original reads back to the assembly and see what we have - refer to previous notes if you are unsure of the steps.

Start in the hybrid assembly directory
`~/workshop_materials/genomics_tutorial/data/sequencing/pseudomonas_gm41/hybrid_assembly`

```
mkdir remapping_to_assembly
```

```
cd remapping_to_assembly
```

```
cp ../scaffolds.fasta .
```

```
bwa index scaffolds.fasta
```

First remap the Illumina reads. Type all on one line:

```
bwa mem -t 2 scaffolds.fasta ../../SRR491287a_1.filtered.fastq
../../SRR491287a_2.filtered.fastq > gm41.illumina.sam
```

Convert the sam to bam:

```
samtools view -bS gm41.illumina.sam > gm41.illumina.bam
```

Sort the bam file by genomic coordinate:

```
samtools sort –o gm41.illumina.sorted.bam gm41.illumina.bam
```

Index the bam file:

```
samtools index gm41.illumina.sorted.bam
```

Collect mapping stats:

```
samtools flagstat gm41.illumina.sorted.bam > mappingstats.illumina.txt
```

We can also map the PacBio reads, but for this we will use Heng Li's new and improved aligner minimap2 (https://github.com/lh3/minimap2#map-long-genomic). At the moment, this aligner is better for long reads, but for short genomic data (Illumina), Heng Li still recommends bwa. See this post by Heng Li for more information (http://lh3.github.io/2018/04/02/minimap2-and-the-future-of-bwa).

```
minimap2 -ax map-pb scaffolds.fasta ../../SRR1042836a.fastq >
gm41.pacbio.sam
```

And then the same as before (convert to bam, sort, index, mapping stats)

```
samtools view -bS gm41.pacbio.sam > gm41.pacbio.bam
```

```
samtools sort -o gm41.pacbio.sorted.bam  gm41.pacbio.bam
```

```
samtools index gm41.pacbio.sorted.bam
```

```
samtools flagstat gm41.pacbio.sorted.bam > mappingstats.pacbio.txt
```

```
21903 + 0 in total (QC-passed reads + QC-failed reads)
308 + 0 secondary
9095 + 0 supplementary
0 + 0 duplicates
17777 + 0 mapped (81.16% : N/A)
0 + 0 paired in sequencing
0 + 0 read1
0 + 0 read2
0 + 0 properly paired (N/A : N/A)
0 + 0 with itself and mate mapped
0 + 0 singletons (N/A : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

You will notice that not such a high proportion of PacBio reads map back to the assembly.

Now start igv:
`igv.sh`

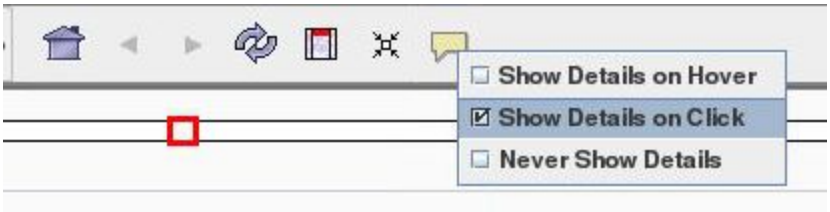
Load your assembled genome -

Click on genome - load from file

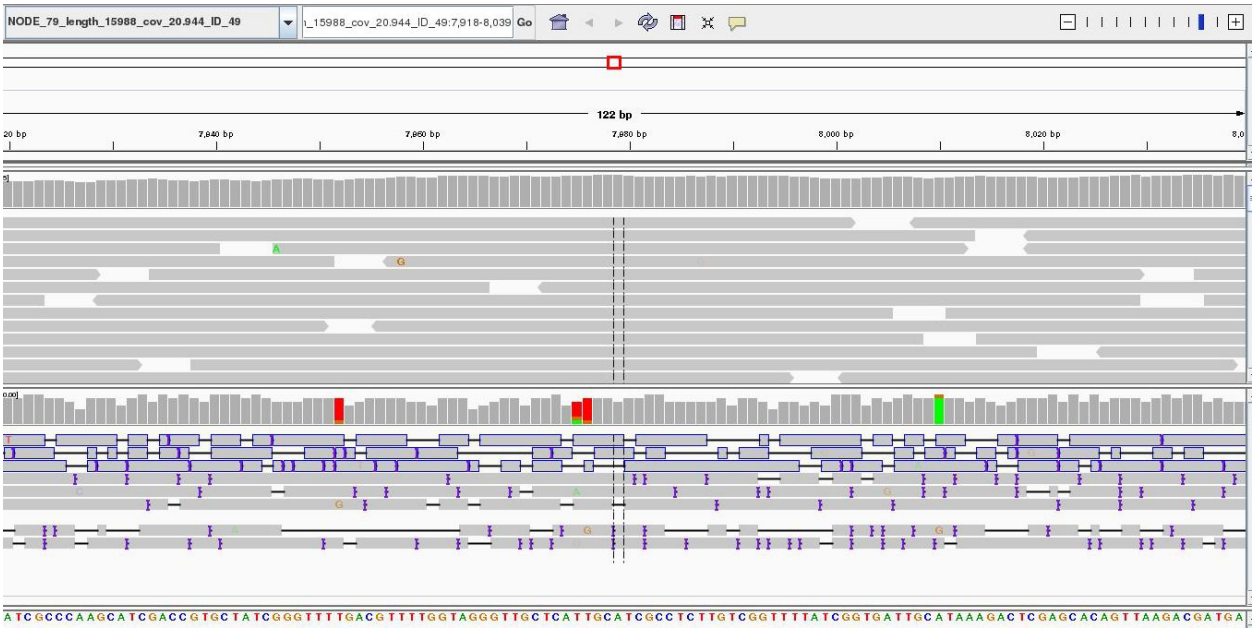
Make sure you get the assembly from the `hybrid_assembly` (igv remembers the previous directory which may contain similar files.)

Now load your 2 alignment files:
click on load from File and then select `gm41.pacbio.sorted.bam` and `gm41.illumina.sorted.bam`

On the toolbar select - "Show Details on Click"



Find a region that has decent coverage of both reads and zoom in.
(Region shown here: NODE_79_length_15988_cov_20.944_ID_49:7,963-8,084)



You can see that the PacBio reads are much longer, but the error rate particularly insertions and deletions is much higher than for the Illumina reads.

Explore a few other contigs to see if you can find something that looks like an error or mis-assembly. Remember the assembly process is difficult and far from perfect.

Summary

You have seen that de-novo assembly of short reads is a challenging problem. Even for small genomes, the resulting assembly is fragmented into contigs and far from complete.

Incorporating longer reads to produce a hybrid assembly can be used to reduce the fragmentation of the genome. We have only used a single (perhaps the simplest) technique to incorporate long reads. You can read more about hybrid assembly techniques here: http://www.ncbi.nlm.nih.gov/pubmed/24930142

Concluding Remarks

Well done! If you have reached this far, you deserve a good pat on the back and a cup of tea (or a whisky). You have completed some of the most common tasks in genomics. You can use the same machine and the same scripts to perform an analysis of any dataset!

If you need to transfer data to/from the instance a tutorial can be found at http://www.siteground.com/tutorials/ssh/ssh_winscp.htm or will be covered in the last session of this Workshop!