

2014 Workshop on Genomics

Part 1: Short read genomics: Remapping

Instructors:

- Konrad Paszkiewicz k.h.paszkiewicz@exeter.ac.uk

Objectives:

By the end of the workshop you will be expected to:

- Understand how short reads are generated
- 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/Map file) using BWA
- Convert the SAM file to BAM format (Binary Alignment/Map 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

Background:

- This course assumes that you have successfully completed the Unix workshop or have equivalent knowledge.

1.1 Introduction

Welcome to the Genomics workshop. Generating reams 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 5 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.

The five parts are:

1. Introduction to Illumina sequencing-by-synthesis
2. Remapping a strain of *E.coli* to a reference sequence
3. Assembly of unmapped reads
4. Complete *de-novo* assembly of all reads
5. Repeating parts 3-5 on two other strains and comparing them

For this first workshop we will assume little background knowledge, save a basic familiarity with the Linux operating system and the Amazon 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.

In the second part 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. In part 5, you will be asked to repeat the steps in workshops 1, 2 and 3 on 2 other data sets and to compare the results.

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

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

It means, type the highlighted text into your terminal.

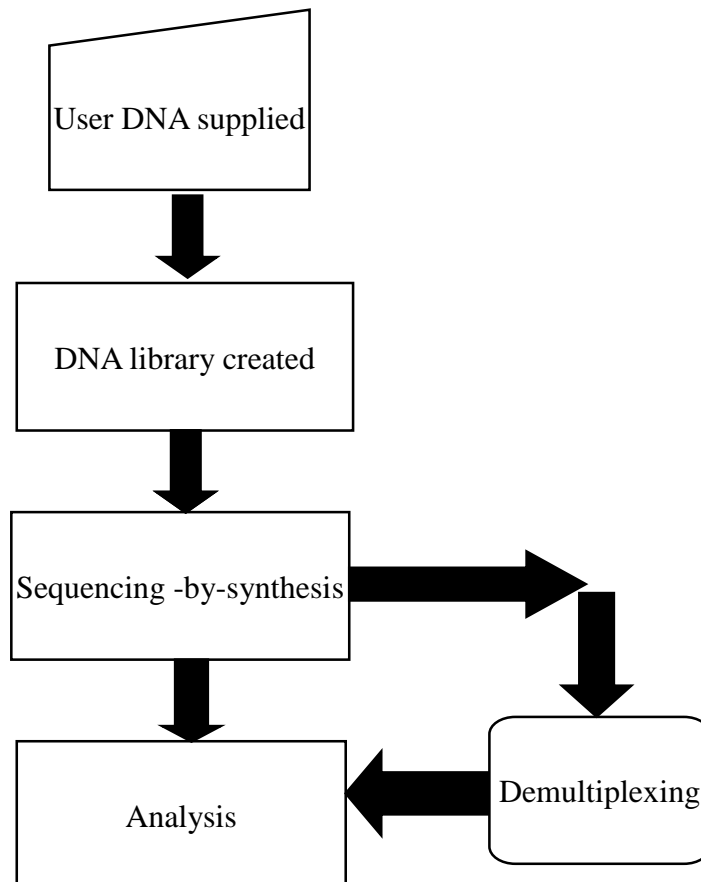
1.2 Principles of Illumina-based sequencing:

There are several second generation (i.e. non-Sanger) sequencers currently on the market. These include the Life 5500 (formerly known as the ABI SOLiD), the Roche 454 GS FLX and 454 Junior, Ion Torrent, and the Illumina HiSeq and MiSeq systems. All of these systems 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) sequencer, is able to detect fluorescence from a single molecule of DNA. However, the machine weighs 2 tons, produces 1/10000th of the data of an Illumina run and has a 5-10% error rate. Despite these issues, the system is capable of producing reads up to 25kb from a single DNA molecule. As such the data produced by this system is useful for sequencing entire genes or closing gaps in existing assemblies.

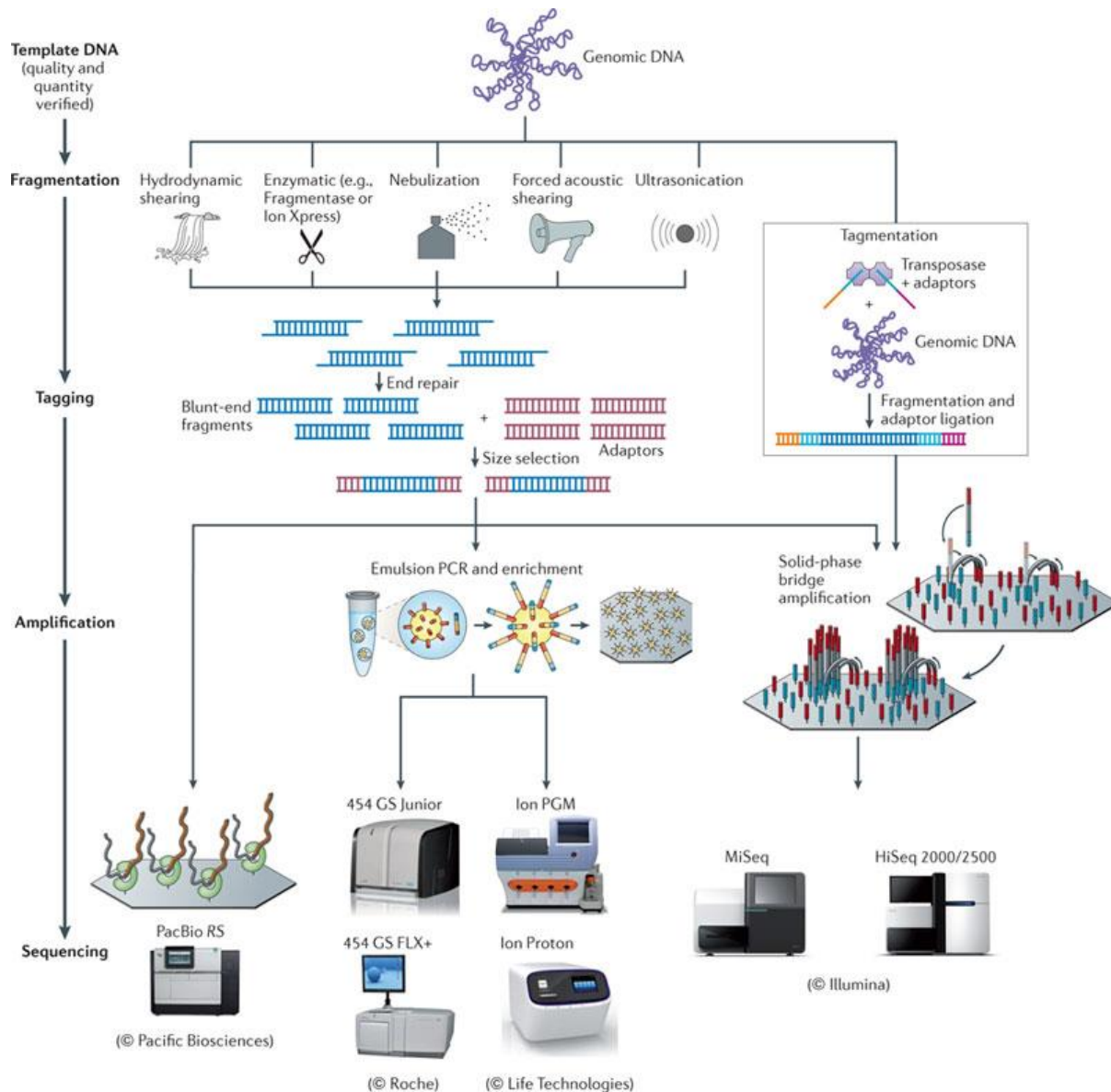
We will only look at the Illumina sequencing pipeline here, but the basic principles apply to all other sequencers. If you would like further details on other platforms then I recommend reading *Mardis ER. Next-generation DNA sequencing methods. Annual Reviews Genomics Hum Genet 2008; 9 :387-402* Other papers are listed at <http://biosciences.ex.ac.uk/facilities/sequencing/usefulresources/>

A typical sequencing run would begin with the user supplying 1-10 ug of genomic DNA to a facility along with quality control information in the form of an Agilent Bioanalyser trace or gel image and quantification information. The following flowchart illustrates the basic workflow.



1.3 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 6-8bp 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.



Nature Reviews | Microbiology

High-throughput bacterial genome sequencing: an embarrassment of choice, a world of opportunity

Nicholas J. Loman, Chrystala Constantinidou, Jacqueline Z. M. Chan, Mihail Halachev, Martin Sergeant, Charles W. Penn, Esther R. Robinson & Mark J. Pallen *Nature Reviews Microbiology* **10**, 599-606 (September 2012)

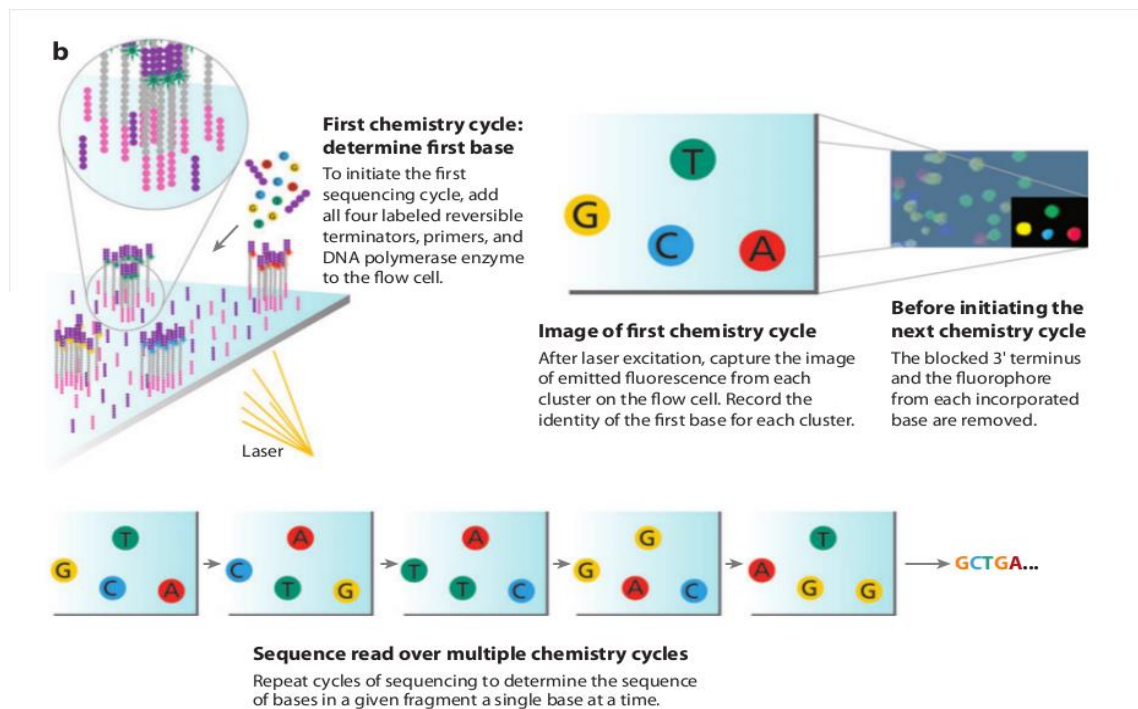
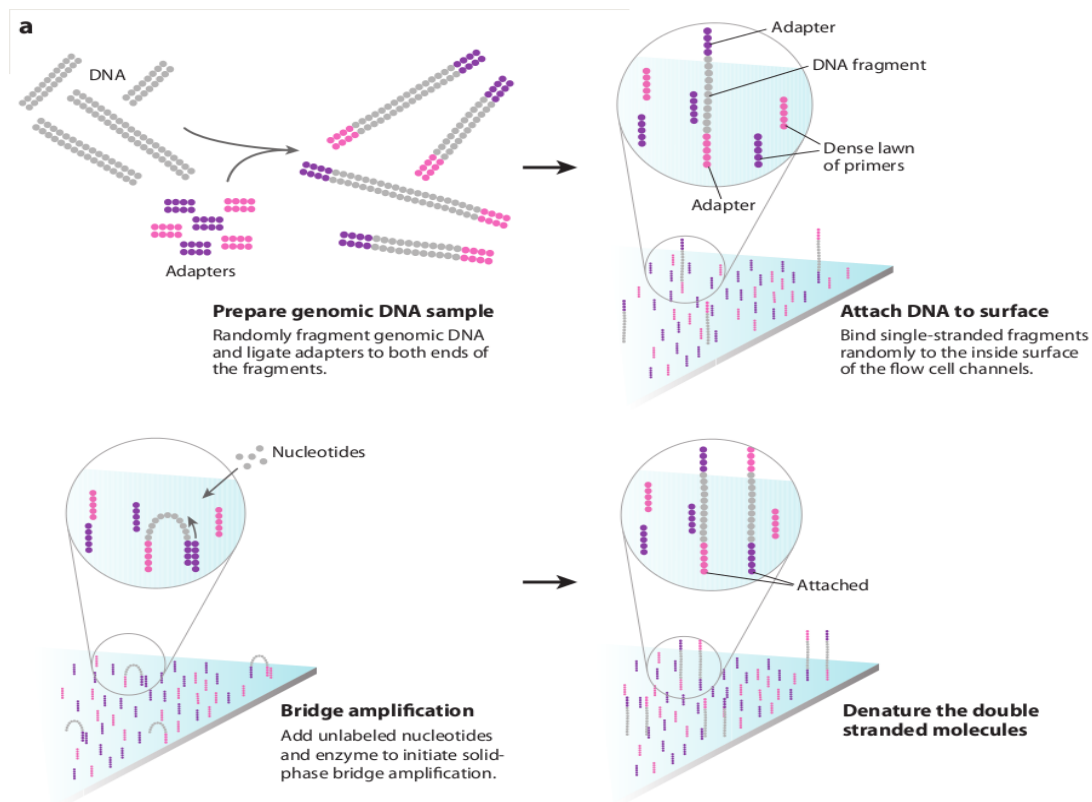
1.4 Sequencing

(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 2000/2500 or MiSeq starts with an Illumina-specific adapter library and takes place on the oligo-derivatized surface of a flow cell, and is either performed on the instrument itself or 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 mm². 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 figure on the following page summarises the process:



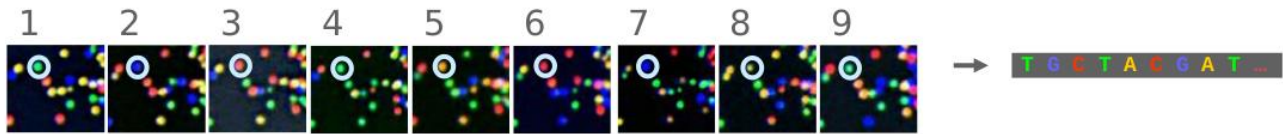
The Illumina sequencing-by-synthesis approach: Cluster strands created by bridge amplification are primed and all four fluorescently labeled, 3'-OH blocked nucleotides are added to the flow cell with DNA polymerase. The cluster strands are extended by one nucleotide. Following the incorporation step, 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 effect 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

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. This is in contrast to the 454 system where the position of each cluster is defined by steel plate with picolitre sized holes in which the reaction takes place.

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!).



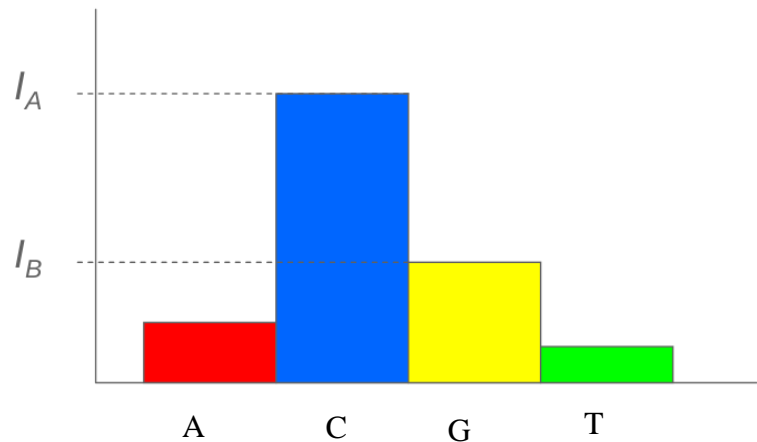
The above figure illustrates the principles of base-calling from cycles 1 to 9. If we focus on the highlighted cluster, one can observe that the colour (wavelength) of light observed at each cycle changes along with the brightness (intensity). This is due to the incorporation of complementary ddNTPs 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:

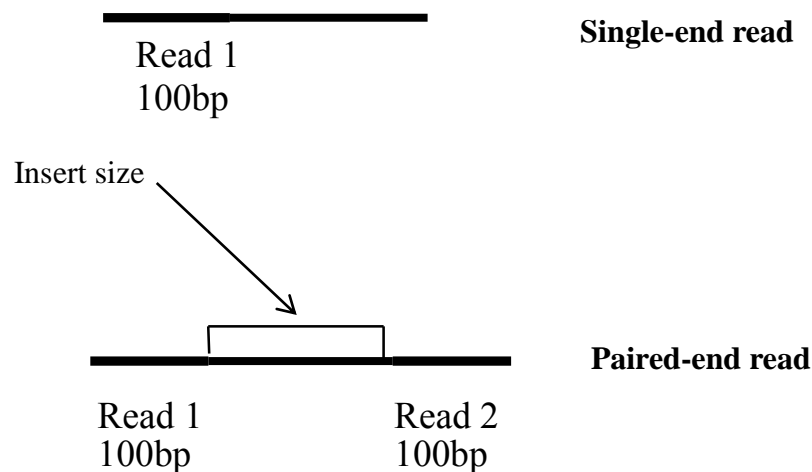
$$C = \frac{I_A}{I_A + I_B}$$



1.5 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 HiSeq) 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 I.e:



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 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.

Repetitive DNA

Unique DNA



Paired read maps uniquely



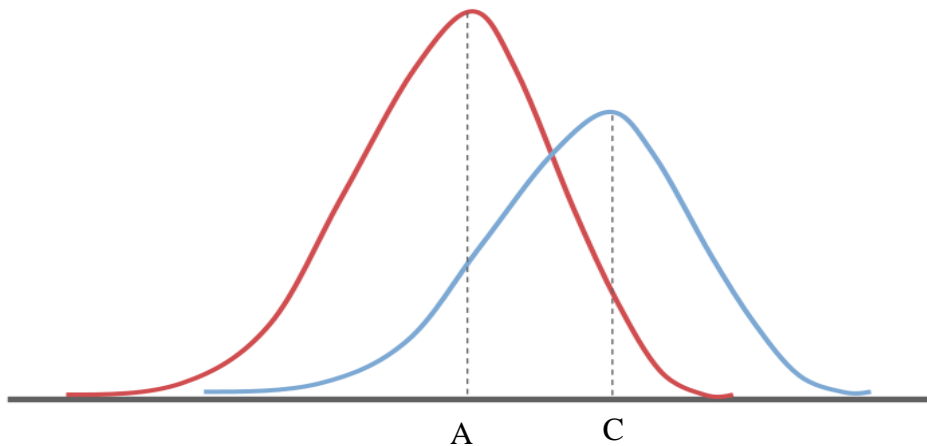
Single read maps to multiple positions

1.6 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:

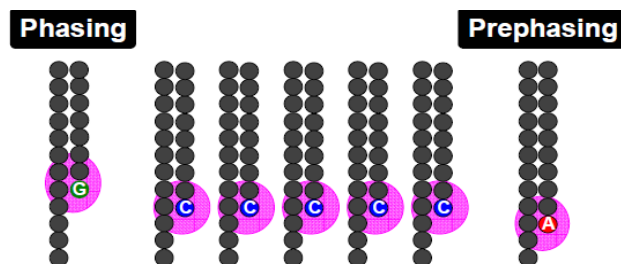
When reading an A base, a small amount of C will also be measured due to frequency overlap and vice-versa. Similarly with G and T bases. Additionally, from the figure below, it should be clear that the extent to which the dyes fluoresce differs. As such it is necessary to normalize the intensities. This 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. E.g. 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 your sample will be put through several cycles of PCR before sequencing. 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 homopolymeric stretch of sequence. This much more of a problem on the 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

1.7 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. Most quality scores are calculated using the Phred scale. 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 454, SOLiD and Illumina runs you should see quality scores between Q20 and Q40. Note that these are 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 bias 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 should be filtered out.

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.

2014 Workshop on Genomics

Part 2: Short read genomics: Remapping

Instructors:

- Konrad Paszkiewicz k.h.paszkiewicz@exeter.ac.uk

Objectives:

By the end of the workshop you will be expected to:

- 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

2.1 Introduction

You'll need to be within your Amazon instance to use the workshop resources here.

In this section of the workshop we will be analysing a (hypothetical) strain of *E. coli* which is involved in urinary tract infections and causes internal hemorrhaging. The strain we start with is treatable with standard antibiotics. We want to obtain a list of single nucleotide polymorphisms (SNPs), insertions/deletions (Indels) and any genes which have been deleted.

In later sections you will repeat this analysis on your own using two other strains which have developed resistance. The final section will ask you to compare the results from all three datasets.

2.2 Quality control

In this section of the workshop we will be learning about evaluating the quality of an Illumina sequencing run. The process described here can be used with any FASTQ formatted file from any platform (e.g 454, Illumina, Ion Torrent, PacBio etc).

2nd (and 3rd) generation sequencers produce vast quantities of data. A single Illumina HiSeq lane will produce over 10Gb 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 over-represented 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 over-represented 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:

Most 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 454, SoLID and Illumina runs you should see quality scores between Q20 and Q40. Note that these are 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.

FASTQ format:

A FASTQ entry consists of 4 lines

```
@D3P26HQ1:110:d0eh1acxx:8:1101:1116:2122 1:N:0:  
AGGTGTCTCTACAACCAAGCTACAACAGAGCAATGGGCTATCTGGTGGGATTTAAAGGGGTGAAAATGCATCCCCCTAAAATNAAAGTGGTTTT  
+  
ADDADCFHHHDHGHIII<GIICH4FGCIHIEGFHGHGIIIGDHFDFG?DEHH>FGIG=E@GGADDDCCCC@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

Typical FASTQ formatted file:

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.

	Range	Offset	Type	Range
Sanger standard				
fastq-sanger	33-126	33	PHRED	0 to 93
Solexa/early Illumina				
fastq-solexa	59-126	64	Solexa	-5 to 62
Illumina 1.3+				
fastq-illumina	64-126	64	PHRED	0 to 62

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.

2.2.1 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 (<http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/>). FastQC offers a graphical visualisation of QC metrics, but *does not* have the ability to filter data.

Task 1:

In your instance, from your home directory change into the `genomics_tutorial/strain1` directory. E.g.:

```
cd genomics_tutorial/strain1
```

Note that the FASTQ files are contained within a sub-directory called `illumina_reads`. Change into this directory by typing:

```
cd illumina_reads
```

```
ubuntu@ip-10-212-185-189: ~/genomics_tutorial/strain1/illumina_reads
File Edit View Search Terminal Help
ubuntu@ip-10-212-185-189:~$ ls
configure_freenx.sh Desktop genomics_tutorial install perl_tutorial_materials
ubuntu@ip-10-212-185-189:~$ cd genomics_tutorial/
ubuntu@ip-10-212-185-189:~/genomics_tutorial$ ls
reference_sequence strain1 strain2 strain3
ubuntu@ip-10-212-185-189:~/genomics_tutorial$ cd strain1/
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1$ ls
illumina_reads
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1$ cd illumina_reads/
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1/illumina_reads$ ls
strain1_read1.fastq strain1_read2.fastq
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1/illumina_reads$
```

Note that this is a paired-end run. As such there are two files, one for read 1 (`strain1_read1.fastq`) and the other for the reverse read 2 (`strain1_read2.fastq`). Reads from the same pair can be identified because they have the same header. Many programs require that the read 1 and read 2 files have the reads in the same order. To view the first few reads we can use the 'more' command:

```
more strain1_read1.fastq
```

```
@CP000243.1-6332150-1
cgtctttggtgtacaggacggtcacgatgatgtggtgctg
+
C5CCCCB4C6BC##?C:II#&#BCBC)CCCB<C1C<C#??
@CP000243.1-6332148-1
ttcaccgcatgatcatcaccggatcgccgggatcatg
+
CBI&BCCCABIBACC;CBIIC@:ICI@ICAB9C@;#:CCB
@CP000243.1-6332146-1
cgatgccgaatctattctgcactttgcgagattggcttg
+
BC8C@AC7>CCFC2@C6AICIACB#C##CBB#7BB@C#AI
@CP000243.1-6332144-1
tggacggcgcttcaatgattattggctatggcgcagatct
+
CCCIC0&CCC=B?:C<CI2IAICBC@II?CCC@AC ,BCC<
@CP000243.1-6332142-1
ccgcaataccgaccataagaagatttatgcgaaacgcctt
+
```

Press 'q' to exit more. We can do the same for read 2 and confirm that the headers are the same.

more strain1_read2.fastq

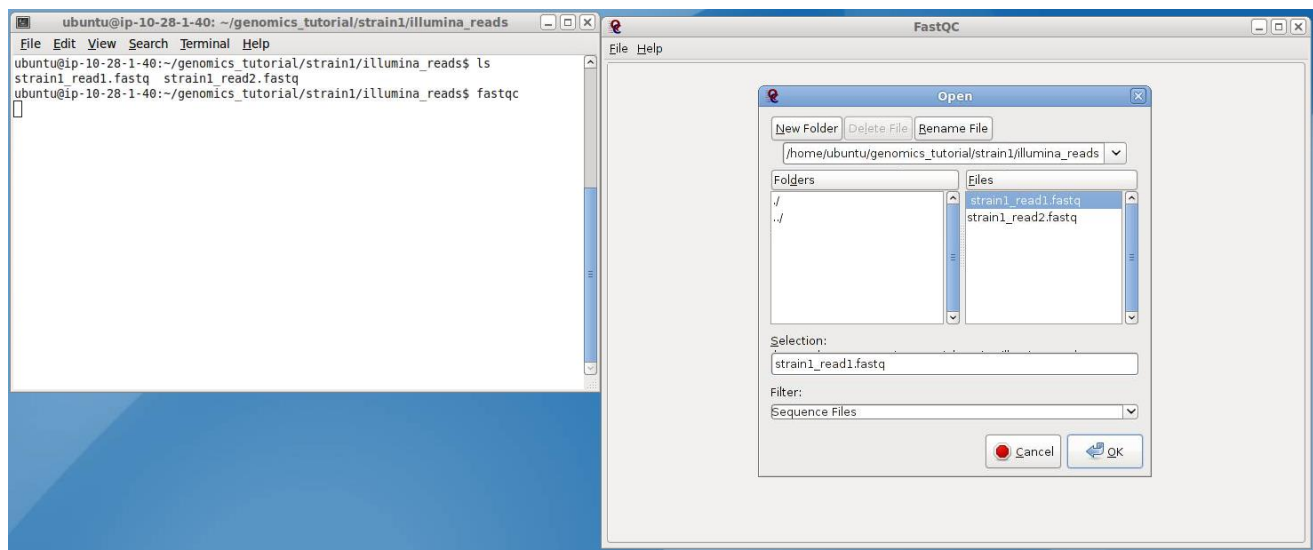
```
@CP000243.1-6332150-2
gtgcgggagagtcccacaacacgcaggactcttttgcttt
+
3E&1G=HH;HH>H?#?5+&HEA+ABFE?F,CHHE#C$IB:
@CP000243.1-6332148-2
aataaactcgttgaactgtacaacaagcataggtcgtga
+
?CGB@BC##BIFH1CB '*ABAHH9>+:AH#%<BHH' AB+A
@CP000243.1-6332146-2
acaacacgcagtgccggacgcgtgacatagcgccccagca
+
G.GC70A>??6>CGCFAEG9>><@HFGBH=BB?@>AAHH#
@CP000243.1-6332144-2
tgcaacgagcagcttgctagtaaagacaaggagcaacctt
+
BHBG,@IB?*BBI;%HBIFB BBH+A#B?FG(D?HGHH?6
@CP000243.1-6332142-2
aatatccagctggtgtggttaatttaagcacacattgtcaa
+
```

Again, press 'q' to exit more.

Now, let's start the fastqc program. Type:

fastqc

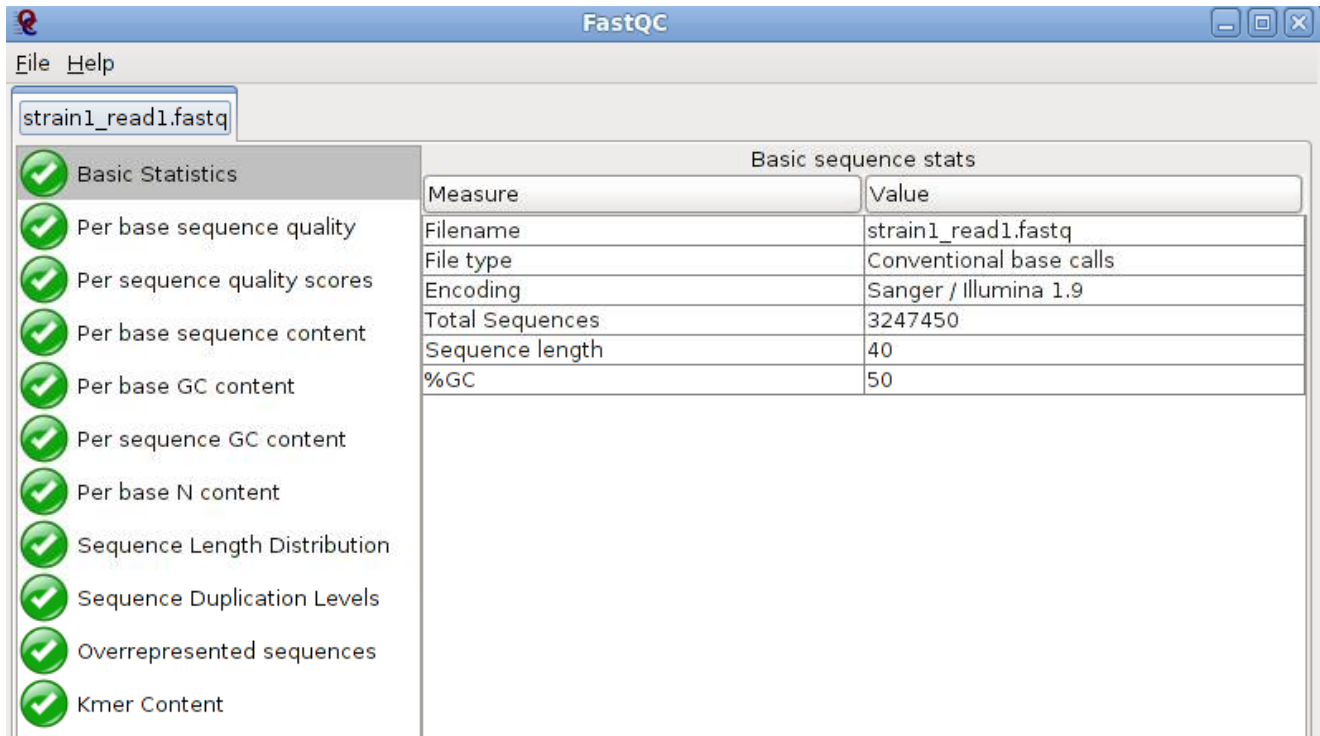
This should open the fastqc program. Load the strain1_read1.fastq file in the genomics_tutorial/strain1/illumina_reads directory.



After a few minutes the program should finish analysing the FASTQ file.

The fastqc program performs a number of tests which determines whether or 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 and '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.

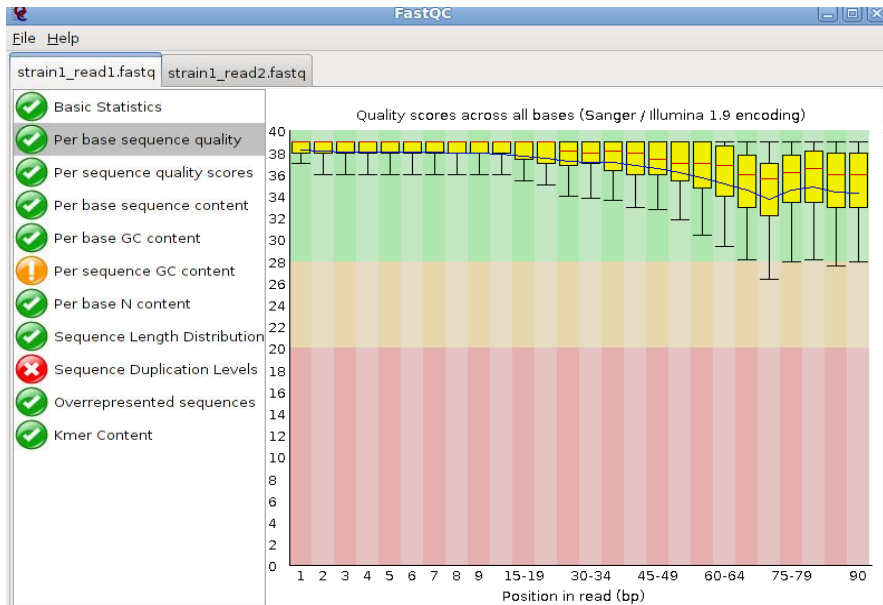
The bottom line is that you need to be aware of what your library is and whether what fastqc is reporting makes sense for that type of library.



In this case we have an *E.coli* library so a percentage GC content of 50% is consistent with our expectations.

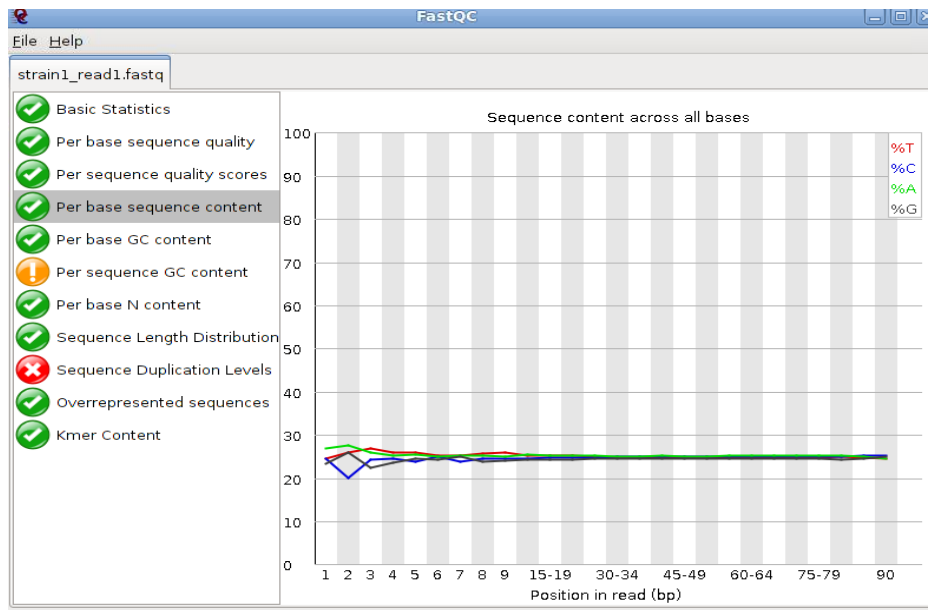
Quality scores:

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.



Per base sequence content:

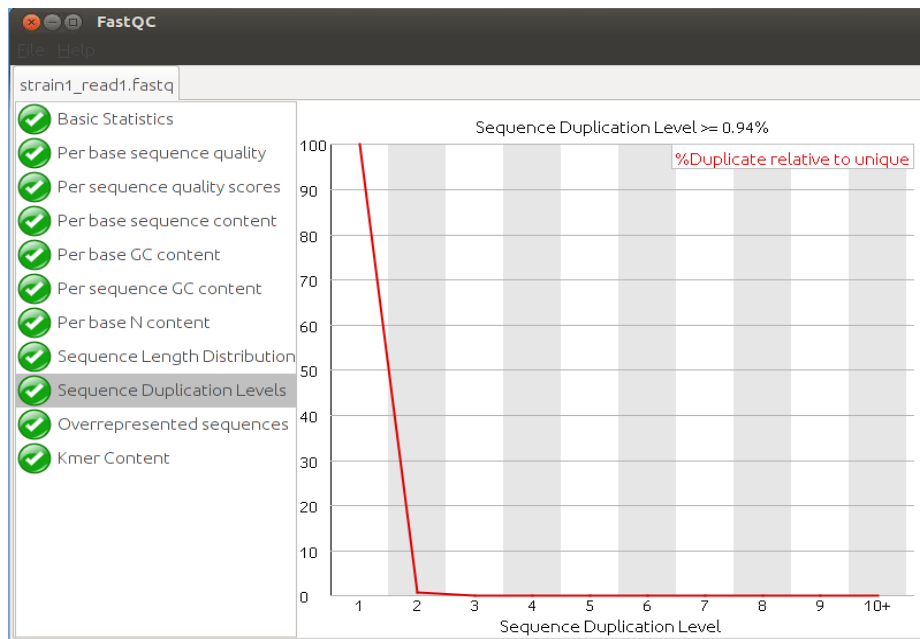
For a completely randomly generated library with a GC content of 50% one expects that at any given position within a read there will be a 25% chance of finding an A,C,T or G base. Here we can see that our library satisfies these criteria, although there appears to be some minor bias at the beginning 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://biosciences.exeter.ac.uk/facilities/sequencing/dataguide/qualitycontrol/> 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.



Task 2:

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

Note that the number of reads reported in both files is identical. This is because if one read fails to pass the Illumina chastity filter, its partner is automatically excluded too.

Overall, both read 1 and read 2 can be regarded as 'good' data-sets.

2.2.2 Quality control – filtering of Illumina data

In this section we will be actually be filtering the data to ensure any low quality reads are removed and that any sequences containing adaptor sequences are either trimmed or removed altogether. To do this we will use the fastq-mcf program from the ea-utils package (available at <http://code.google.com/p/ea-utils/>). This package is remarkably fast and ensures that after filtering both read 1 and read 2 files are in the correct order.

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

Make sure you are in the ~/genomics_tutorial/strain1/illumina_reads directory. We will execute the fastq-mcf program which performs both adaptor sequence trimming and low quality bases. To remove adaptor sequences, we need to supply the adaptor sequences to the program. A list of the most common qadaptors used is given in the file ~/software/ea-utils/adaptors.fasta:

```
ubuntu@linux:~$ more ~/software/ea-utils/adaptors.fasta
>Nextera_enrichment
CTGTCTCTTATACACATCT
>TruSeq_Read1
AGATCGGAAGAGCACACGTCTGAACTCCAGTCA
>TruSeq_Read2
AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT
>Nextera_mate_pair_Read1
CTGTCTCTTATACACATCT
>Nextera_mate_pair_Read2
AGATGTGTATAAGAGACAG
>PolyA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Task 3:

To run the fastq-mcf program, type the following (all on one line):

```
fastq-mcf ~/software/ea-utils/adaptors.fasta strain1_read1.fastq strain1_read2.fastq -o strain1_read1.filtered.fastq -o strain1_read2.filtered.fastq -C 1000000 -q 20 -p 10 -u -x 0.01
```

The input options are as follows:

~/software/ea-utils/adaptors.fasta is the location of the adaptor file

-o indicates the output files to be created

-C indicates a number of reads to use for subsampling (this is set to speed up the process)

-q quality threshold causing base removal (any read with a Q score less will be removed)

-p maximum adaptor difference percentage (allows for mismatches in the adaptor which may be due to sequencing errors)

-u enable Illumina PF filtering (see below for what this means)

-x bad read percentage causing cycle removal

After a few minutes the filtering should be complete and you should see something similar to:

```
ubuntu@ip-10-239-138-237:~/genomics_tutorial/strain1/illumina_reads$ fastq-mcf ~/software/ea-utils/adaptors.fasta strain1_read1.fastq strain1_read2.fastq -o strain1_read1.filtered.fastq -o strain1_read2.filtered.fastq -C 1000000 -q 20 -p 10 -u -x 0.01
Scale used: 2.2
Filtering Illumina reads on purity field
Phred: 33
Threshold used: 2501 out of 1000000
Files: 2
Total reads: 3247450
Too short after clip: 0
Trimmed 708987 reads (strain1_read1.fastq) by an average of 1.22 bases on quality < 20
Trimmed 1037979 reads (strain1_read2.fastq) by an average of 1.36 bases on quality < 20
```

Illumina sequencing has a filtering step known as a “Chastity Filter”. Dependent upon how this is set up by each individual sequencing service, the output file may contain both reads which pass and fail this chastity filter. Filtering Illumina reads on purity field removes all the reads which failed this filter.

The output shows us that a total of 3247450 reads were inputted. 708987 reads were trimmed from the read 1 file and a slightly higher number (1037979 reads) were trimmed from the read 2 file. Generally, read 1 has fewer reads trimmed than read 2 as read 2 tends to be of slightly lower quality than read 1.

The filtered reads should be present in files strain1_read1.filtered.fastq and strain1_read2.filtered.fastq

Task 4:

Check the quality scores and sequence distribution in the fastqc program for the two filtered fastq files. You should notice very little change (since comparatively few reads were filtered).

Note that although in this case filtering did very little, most runs are not as 'clean' as this so you may find your filtering removes many more reads in the real-world.

Task 5:

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.

```
tail strain1_read1.filtered.fastq
```

```
tail strain1_read2.filtered.fastq
```

```
ubuntu@domU-12-31-39-16-C1-55:~/genomics_tutorial/strain1/illumina_reads$ tail strain1_read1.filtered.fastq
+
BBABBBBBBABABBABABABBAABBABBBBBBABBGHHHG
@CP000243:5:9999:500:40/1
TCCGGCTTTATCCCCTGATTCTGTGGATAACCGTATTACC
+
DEDDDEDEDDDEDDDEDDDEDEDEDEDEDEDEBBAAB
@CP000243:6:10000:423:40/1
GACGCCATTCTGCGTCAGAGCAGACTCAATCTTGCAATA
+
DCCDDCCDCDDCCDDDDCCDDCCDDCCDDCCDCHHHHH
ubuntu@domU-12-31-39-16-C1-55:~/genomics_tutorial/strain1/illumina_reads$ tail strain1_read2.filtered.fastq
+
FGGGGFGGGFFFGGGFFFGGGFFGGF; <<<<
@CP000243:5:9999:500:40/2
TGTCTGCAAGTTCGAATTACCAACAAGCACTGCTTTTT
+
IJIJIJIJIJJIIIIJIJIJJIIIIJIJJIIIIJJII87888
@CP000243:6:10000:423:40/2
TGCGCTTATCCGGTAACTATCGTCTTGAGTCCAACCCGG
+
CCDCCDDCCDDCCDDCCDDCCDDCCDDCCDDCCDBBBAB
ubuntu@domU-12-31-39-16-C1-55:~/genomics_tutorial/strain1/illumina_reads$
```

Task 6:

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 "@CP000243" strain1_read1.filtered.fastq
```

Do the same for the strain1_read2.filtered.fastq file.

2.3 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 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.

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 that indicates 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 100bp read is likely to contain at least one error.

Let's look at 2 potential SNPs which are in fact artefacts.

1. 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.

2. PCR duplication:

The highlighted region red on the left shows where there appears to be a variant (either due to sequencing of a diploid genome or non-clonal samples). A C-base is present in the reference and half the reads, whilst an A-base is present in the other reads.



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 no. This 'SNP' is in fact due to PCR duplication. I.e. the same fragment of DNA has been replicated many times more than the average and happens to contain an error. We can filter out such reads during after alignment to the reference (see later).

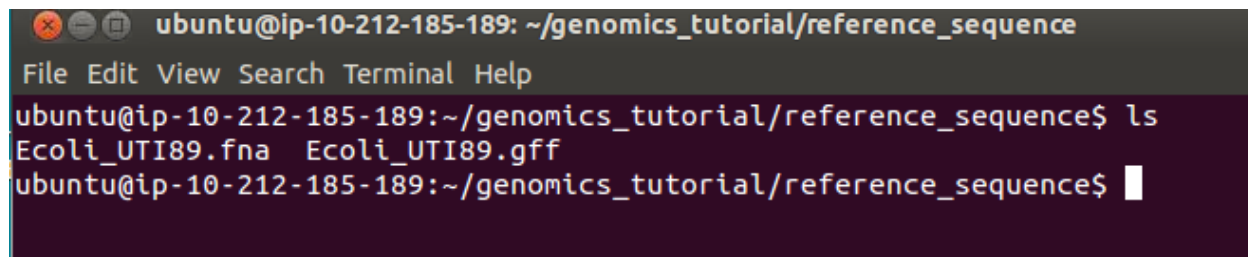
2.3.1: Indexing a reference genome:

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.

Task 7: Generating an index file from the reference sequence

Change directory to the reference_sequence directory:

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


A terminal window with a dark background. The title bar shows 'ubuntu@ip-10-212-185-189: ~/genomics_tutorial/reference_sequence'. The menu bar includes 'File Edit View Search Terminal Help'. The prompt is 'ubuntu@ip-10-212-185-189:~/genomics_tutorial/reference_sequence\$'. The command 'ls' has been executed, showing two files: 'Ecoli_UTI89.fna' and 'Ecoli_UTI89.gff'. The prompt is now 'ubuntu@ip-10-212-185-189:~/genomics_tutorial/reference_sequence\$' with a cursor.

In this directory we have 2 files. Ecoli_UTI89.fna is a FASTA file which contains the reference genome sequence. The Ecoli_UTI89.gff file contains the annotation for this genome. We will use this later.

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

```
bwa
```

This should yield something like:

A terminal window with a dark background. The title bar shows 'ubuntu@domU-12-31-39-16-C1-55:~/genomics_tutorial\$'. The command 'bwa' has been executed. The output is: 'Program: bwa (alignment via Burrows-Wheeler transformation)', 'Version: 0.7.5a-r405', 'Contact: Heng Li <lh3@sanger.ac.uk>', 'Usage: bwa <command> [options]', and a list of commands: 'index', 'mem', 'fastmap', 'pmerge', 'aln', 'samse', 'sampe', 'bwasw', 'fa2pac', 'pac2bwt', 'pac2bwtgen', 'bwtupdate', 'bwt2sa'. A note follows: '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.' The prompt is now 'ubuntu@domU-12-31-39-16-C1-55:~/genomics_tutorial\$'.

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

If we type:

bwa index

We can see more options for the bwa index command:

```
Usage:  bwa index [-a bwtsv|is] [-c] <in.fasta>

Options: -a STR      BWT construction algorithm: bwtsv or is [auto]
         -p STR      prefix of the index [same as fasta name]
         -6          index files named as <in.fasta>.64.* instead of <in.fasta>.*

Warning: '-a bwtsv' does not work for short genomes, while '-a is' and
         '-a div' do not work not for long genomes. Please choose '-a'
         according to the length of the genome.
```

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 bwtsv. For bacterial genomes the default algorithm will work fine.

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

bwa index Ecoli_UTI89.fna

```
ubuntu@ip-10-212-185-189:~/genomics_tutorial/reference_sequence$ bwa index Ecoli_UTI89.fna
[bwa_index] fail to open file 'bwa' : No such file or directory
ubuntu@ip-10-212-185-189:~/genomics_tutorial/reference_sequence$ bwa index Ecoli_UTI89.fna
[bwa_index] Pack FASTA... 0.06 sec
[bwa_index] Construct BWT for the packed sequence...
[bwa_index] 4.50 seconds elapse.
[bwa_index] Update BWT... 0.05 sec
[bwa_index] Pack forward-only FASTA... 0.04 sec
[bwa_index] Construct SA from BWT and Occ... 0.95 sec
[main] Version: 0.7.5a-r405
[main] CMD: bwa index Ecoli_UTI89.fna
[main] Real time: 7.156 sec; CPU: 5.612 sec
ubuntu@ip-10-212-185-189:~/genomics_tutorial/reference_sequence$
```

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

2.3.2: 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 ~/genomics_tutorial/strain1/ directory.

cd ~/genomics_tutorial/strain1/

Now let's create a directory to store our alignment (just to be tidy):

mkdir remapping_to_reference

And now lets change into that directory:

cd remapping_to_reference

```
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1$ ls
illumina_reads
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1$ mkdir remapping_to_reference
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1$ ls
illumina_reads  remapping_to_reference
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1$ cd remapping_to_reference/
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1/remapping_to_reference$ ls
ubuntu@ip-10-212-185-189:~/genomics_tutorial/strain1/remapping_to_reference$ █
```

We can now 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]
  -c INT      skip seeds with more than INT occurrences [10000]
  -S          skip mate rescue
  -P          skip pairing; mate rescue performed unless -S also in use
  -A INT      score for a sequence match [1]
  -B INT      penalty for a mismatch [4]
  -O INT      gap open penalty [6]
  -E INT      gap extension penalty; a gap of size k cost {-O} + {-E}*k [1]
  -L INT      penalty for clipping [5]
  -U INT      penalty for an unpaired read pair [17]

Input/output options:

  -p          first query file consists of interleaved paired-end sequences
  -R STR      read group header line such as '@RG\tID:foo\tSM:bar' [null]

  -v INT      verbose level: 1=error, 2=warning, 3=message, 4+=debugging [3]
  -T INT      minimum score to output [30]
  -a          output all alignments for SE or unpaired PE
  -C          append FASTA/FASTQ comment to SAM output
  -M          mark shorter split hits as secondary (for Picard/GATK compatibility)

Note: Please read the man page for detailed description of the command line and options.
```

We can see that we need to provide BWA with a FASTQ file containing the raw reads (denoted by <in1.fq>) to align to a reference file (unhelpfully this is 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 19 bp of the sequence vs the reference), the number of processors the program should use (-t – our machine has 8 processors). As standard bwa mem will output the results to the screen (so-called standard-out) so in order to store the output in a file we will need to redirect it using the ‘>’ operator.

N.B. Typically the BWA mem algorithm is most effective for longer reads than we are using in this tutorial (i.e. >70bp), however for simplicity we want to show you how BWA mem works in practice despite having shorter reads. This is because most Illumina reads being produced are at least 100bp. If you do have shorter read data (i.e. <70bp) you should use BWA aln rather than BWA mem. See <http://bitsandbugs.org/2013/11/20/validation-of-bwa-mem/> for more details.

Note that our reference sequence is in ~/genomics_tutorial/reference_sequence/Ecoli_UTI89.fna

Our filtered reads for read 1 are in
~/genomics_tutorial/strain1/illumina_reads/strain1_read1.filtered.fastq

Our filtered reads for read 2 are in

~/genomics_tutorial/strain1/illumina_reads/strain1_read2.filtered.fastq

So to align read 1 using 8 processors and output to file alignment.sam type (all on one line):

Task 8: Aligning reads to the reference genome

```
bwa mem -t 8 /home/ubuntu/genomics_tutorial/reference_sequence/Ecoli_UTI89.fna  
/home/ubuntu/genomics_tutorial/strain1/illumina_reads/strain1_read1.filtered.fastq  
/home/ubuntu/genomics_tutorial/strain1/illumina_reads/strain1_read2.filtered.fastq >  
alignment.sam
```

This will take around 5 minutes.

```
[M::main_mem] read 441774 sequences (17523121 bp)...  
[M::mem_pestat] # candidate unique pairs for (FF, FR, RF, RR): (0, 121991, 1, 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: (421, 448, 475)  
[M::mem_pestat] low and high boundaries for computing mean and std.dev: (313, 583)  
[M::mem_pestat] mean and std.dev: (448.17, 39.89)  
[M::mem_pestat] low and high boundaries for proper pairs: (259, 637)  
[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::worker2@2] performed mate-SW for 10541 reads  
[M::worker2@3] performed mate-SW for 10770 reads  
[M::worker2@1] performed mate-SW for 10463 reads  
[M::worker2@6] performed mate-SW for 10687 reads  
[M::worker2@0] performed mate-SW for 10467 reads  
[M::worker2@4] performed mate-SW for 10608 reads  
[M::worker2@7] performed mate-SW for 10722 reads  
[M::worker2@5] performed mate-SW for 10540 reads  
[main] Version: 0.7.5a-r405  
[main] CMD: bwa mem -t 8 /home/ubuntu/genomics_tutorial/reference_sequence/Ecoli_UTI89.fna /home/ubuntu/genomics_tutorial/strain1/illumina_reads  
/strain1_read1.filtered.fastq /home/ubuntu/genomics_tutorial/strain1/illumina_reads/strain1_read2.filtered.fastq  
[main] Real time: 293.744 sec; CPU: 267.529 sec  
ubuntu@ip-10-233-31-155:~/genomics_tutorial/strain1/remapping_to_references$
```


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> [region1 [...]]

Options:

- b output BAM
- h print header for the SAM output
- H print header only (no alignments)
- S input is SAM
- u uncompressed BAM output (force -b)
- l fast compression (force -b)
- x output FLAG in HEX (samtools-C specific)
- X output FLAG in string (samtools-C specific)
- c print only the count of matching records
- L FILE output alignments overlapping the input BED FILE [null]
- t FILE list of reference names and lengths (force -S) [null]
- T FILE reference sequence file (force -S) [null]
- o FILE output file name [stdout]
- R FILE list of read groups to be outputted [null]
- f INT required flag, 0 for unset [0]
- F INT filtering flag, 0 for unset [0]
- q INT minimum mapping quality [0]
- l STR only output reads in library STR [null]
- r STR only output reads in read group STR [null]
- s FLOAT fraction of templates to subsample; integer part as seed [-1]
- ? longer help

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.

Note that our reference sequence is in ~/genomics_tutorial/reference_sequence/Ecoli_UTI89.fna

Type (all on one line):

```
samtools view -bS -T  
~/genomics_tutorial/reference_sequence/Ecoli_UTI89.fna alignment.sam >  
alignment.bam
```

This should take around 5 minutes.

Listing the contents of the directory shows us that alignment.bam is now present as well as alignment.sam.

```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$ ls
alignment.bam alignment.sam
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$
```

Task 10: Remove suspected PCR duplicates

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:

On the command-line type:

```
samtools rmdup alignment.bam alignments.rmdup.bam
```

This will take around 3-4 minutes. Again, once complete, list the directory.

```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$ samtools
ls rmdup alignment.bam alignments.rmdup.bam
[bam_rmdup_core] processing reference CP000243.1...
[bam_rmdup_core] 0 / 13 = 0.0000 in library ' '
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$ ls
alignment.bam alignment.sam alignments.rmdup.bam
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$
```

Task 11: 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 (don't ask me why this isn't done automatically...). We can do this using the samtools sort command.

```
samtools sort alignment.rmdup.bam alignment.rmdup.sorted
```

This will take another 5 minutes or so. Once complete, list the directory contents (again using the ls command). Note that the output file is called alignment.sorted, but actually the program appends .bam to the end of the file (see below).

```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$ samtools
ls sort alignment.rmdup.bam alignment.rmdup.sorted
[bam_sort_core] merging from 2 files...
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$ ls
alignment.bam alignment.rmdup.bam alignment.rmdup.sorted.bam alignment.sam
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$
```

Task 13: 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. We'll use the samtools index command to do this.

Type:

```
samtools index alignment.rmdup.sorted.bam
```

```
ubuntu@ip-10-33-137-138:~/genomics_tutorial/strain1/remapping_to_reference$ samtools index alignment.rmdup.sorted.bam
ubuntu@ip-10-33-137-138:~/genomics_tutorial/strain1/remapping_to_reference$ ls
alignment.bam alignment.rmdup.bam alignment.rmdup.sorted.bam alignment.rmdup.sorted.bam.bai alignment.sam
ubuntu@ip-10-33-137-138:~/genomics_tutorial/strain1/remapping_to_reference$
```

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

Task 13: Obtain mapping statistics

Finally we can obtain some summary statistics.

```
samtools flagstat alignment.rmdup.sorted.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.

```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$ more mappingstats.txt
6494900 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 duplicates
6109686 + 0 mapped (94.07%:-nan%)
6494900 + 0 paired in sequencing
3247450 + 0 read1
3247450 + 0 read2
6032416 + 0 properly paired (92.88%:-nan%)
6032420 + 0 with itself and mate mapped
77266 + 0 singletons (1.19%:-nan%)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$
```

So here we can see we have 6494900 reads in total, none of which failed QC. 94.07% of reads mapped to the reference genome and 92.88% mapped with the expected 500-600bp distance between them. 1.19% reads could not have their read-pair mapped. Some of these will be reads which span the circular chromosome (aligners do not tend to handle circular chromosomes).

0 reads have mapped to a different chromosome than their pair. If there were any such reads they would likely be due to repetitive sequences (e.g. phage insertion sites) or an insertion of plasmid or phage DNA into the main chromosome. If you were dealing with a eukaryotic organism this could be evidence of gene fusion or inter-chromosomal re-arrangements which are known to occur in cancer.

Task 14: Cleanup

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

Type (all on one line):

```
rm alignment.bam alignment.sam alignment.rmdup.bam
```

You should just have the alignment.rmdup.sorted.bam, the index file and the mappingstats.txt file left after this.

```
ubuntu@ip-10-33-137-138:~/genomics_tutorial/strain1/remapping_to_reference$ ls
alignment.rmdup.sorted.bam alignment.rmdup.sorted.bam.bai mappingstats.txt
ubuntu@ip-10-33-137-138:~/genomics_tutorial/strain1/remapping_to_reference$
```

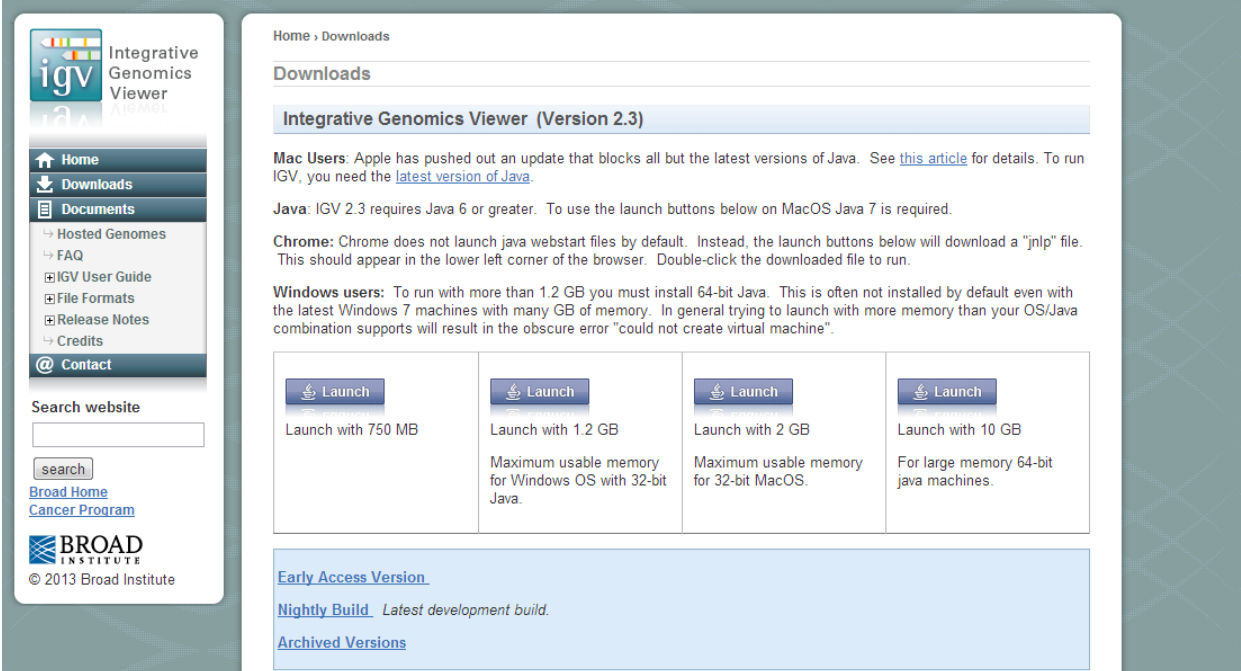
Well done! You have now mapped, filtered and sorted your first whole genome data-set!
Let's take a look at it!

Task 15: Load the Integrative Genomics Viewer

Within your amazon instance (make sure you don't accidentally load firefox on your desktop PC), load the Firefox web browser by double clicking the icon on the desktop. Go to <http://www.broadinstitute.org/software/igv/download>

You may find that you need to register to access the download page. Please do so if you need to.

Once you have registered you should reach the downloads page (similar to the screenshot below). Click on the 'Launch with 2GB' option. If Firefox asks you whether or not to Open or Save the file, click on 'Open'. For larger genomes you may need to use the 10Gb version (though you'll need a machine with at least this much RAM).



Home » Downloads

Downloads





Integrative Genomics Viewer (Version 2.3)

Mac Users: Apple has pushed out an update that blocks all but the latest versions of Java. See [this article](#) for details. To run IGV, you need the [latest version of Java](#).

Java: IGV 2.3 requires Java 6 or greater. To use the launch buttons below on MacOS Java 7 is required.

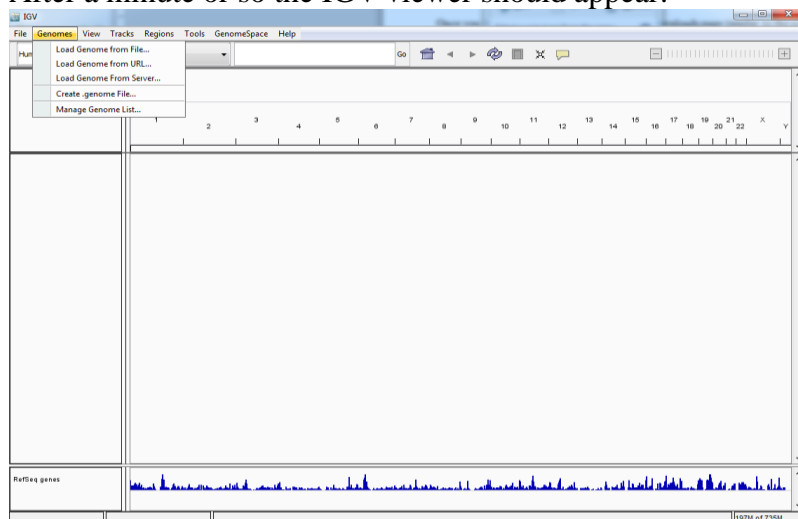
Chrome: Chrome does not launch java webstart files by default. Instead, the launch buttons below will download a "jnlp" file. This should appear in the lower left corner of the browser. Double-click the downloaded file to run.

Windows users: To run with more than 1.2 GB you must install 64-bit Java. This is often not installed by default even with the latest Windows 7 machines with many GB of memory. In general trying to launch with more memory than your OS/Java combination supports will result in the obscure error "could not create virtual machine".

 Launch with 750 MB	 Launch with 1.2 GB Maximum usable memory for Windows OS with 32-bit Java.	 Launch with 2 GB Maximum usable memory for 32-bit MacOS.	 Launch with 10 GB For large memory 64-bit java machines.
---	---	--	--

[Early Access Version](#)
[Nightly Build](#). Latest development build.
[Archived Versions](#)

After a minute or so the IGV viewer should appear:



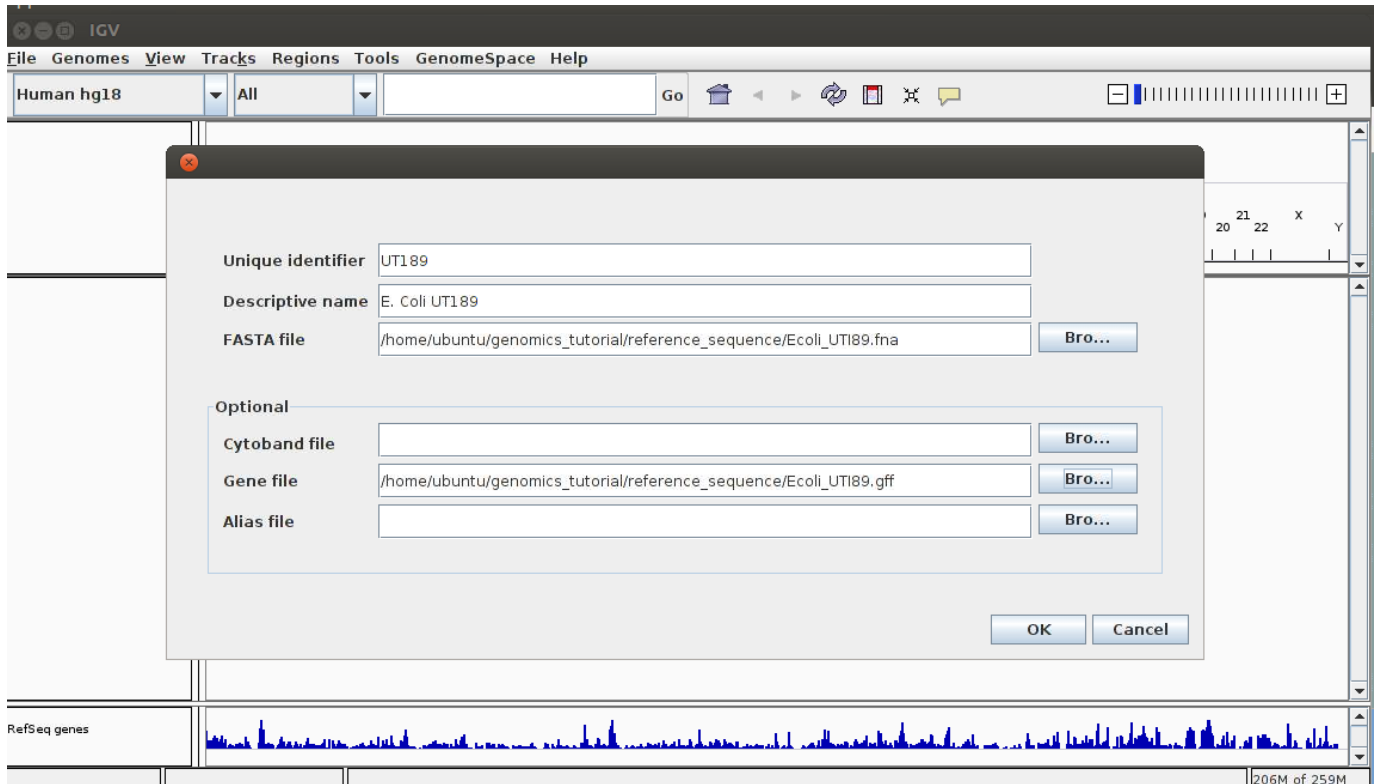
NOTE: alternatively, launch IGV by typing `igv.sh` in a separate terminal window.

Task 16: Import the *E.coli* UTI89 reference genome to IGV

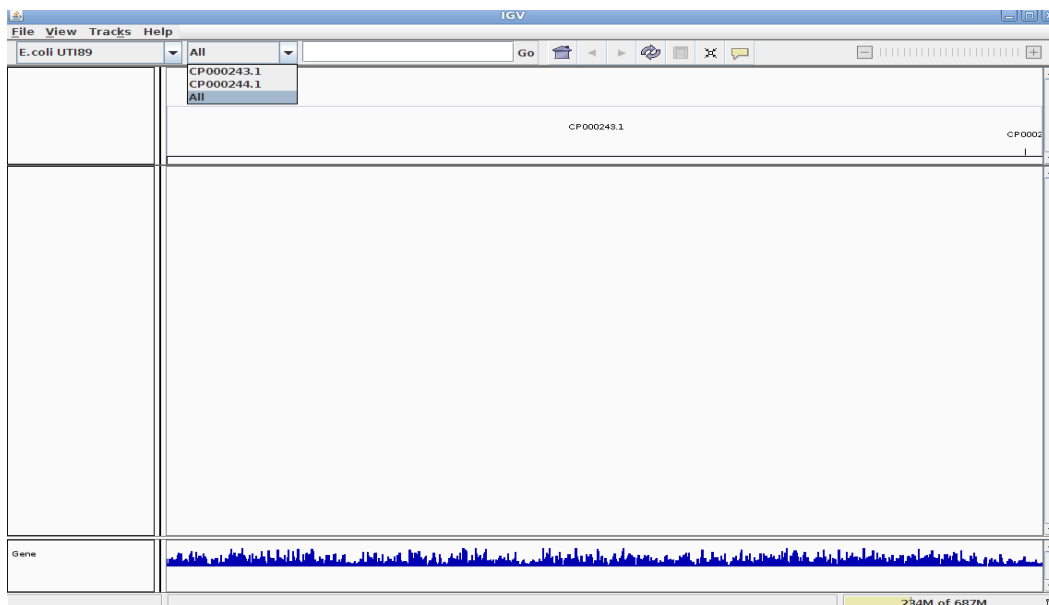
By default IGV does not contain our reference genome. We'll need to import it.

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

Enter the following details into and click on 'Save'.



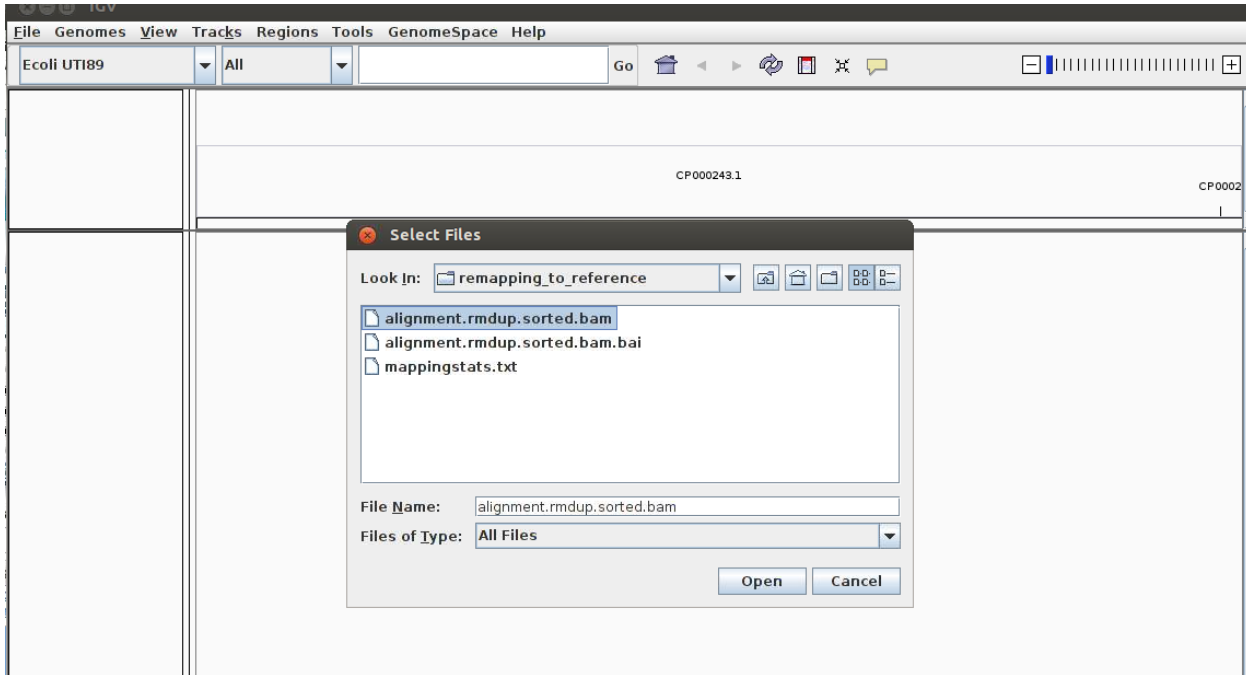
IGV will ask where it can save the genome file. Your home directory will be fine. Click 'Save' again.



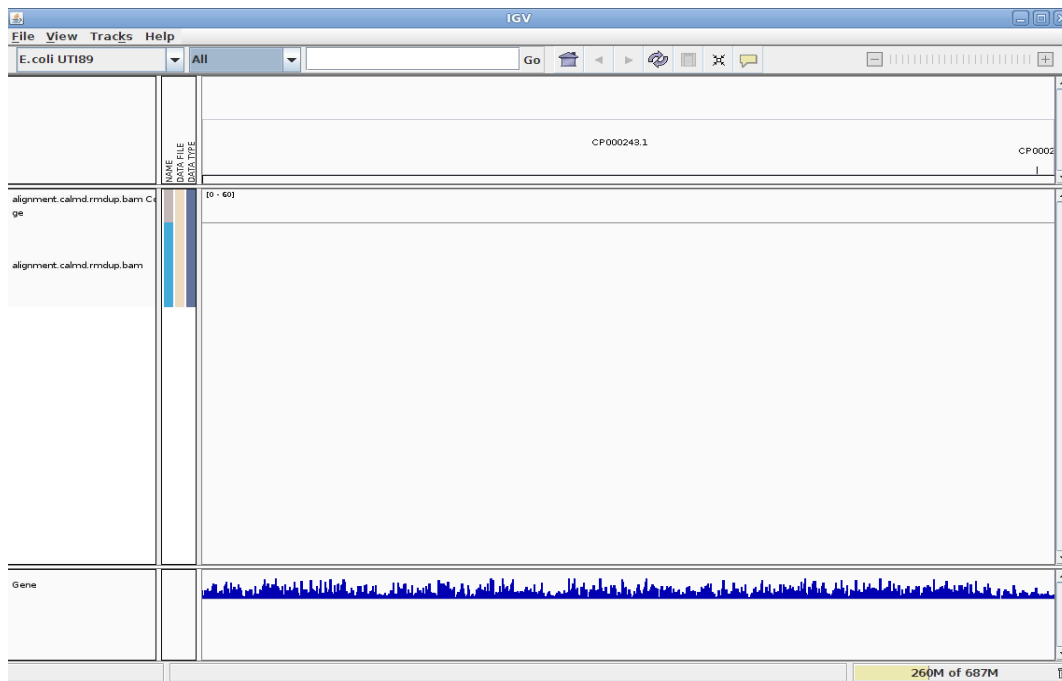
Note that the genome has now been imported. Two chromosomes are listed.

Task 17: Load the BAM file

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

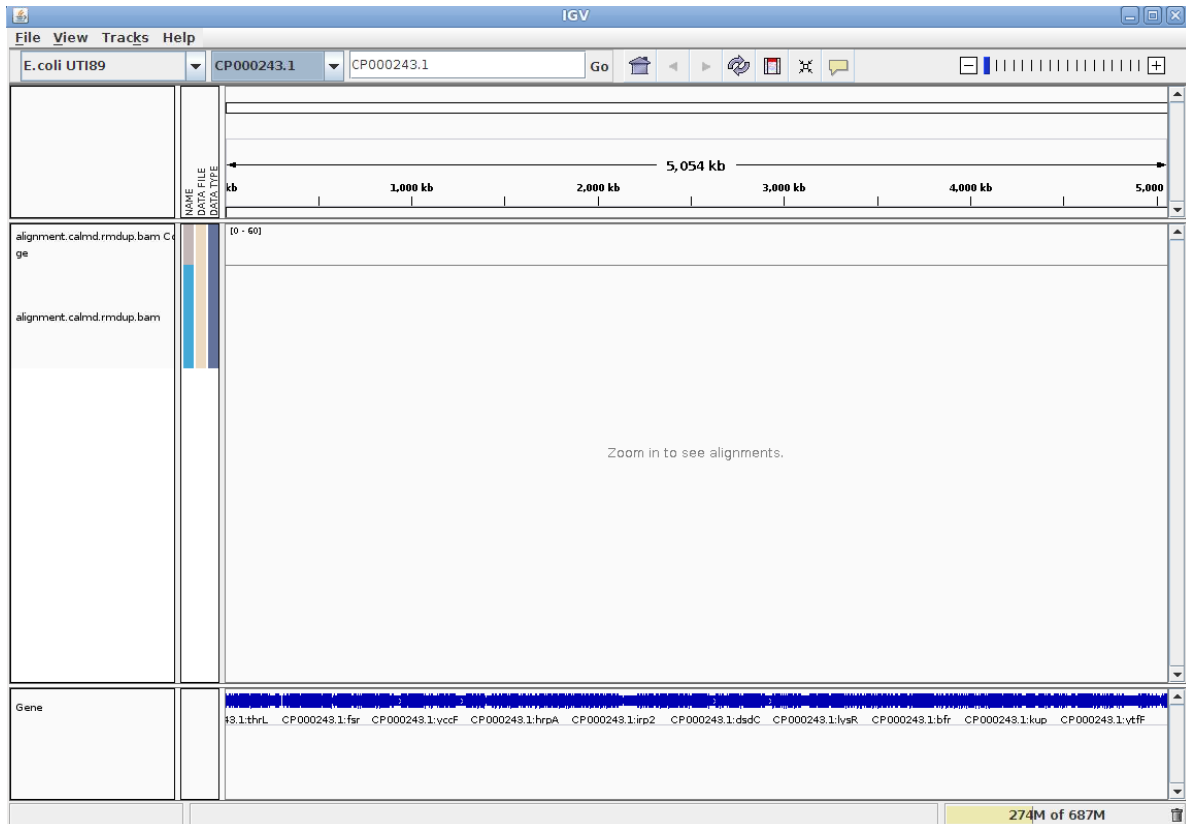


Once loaded your screen should look similar to the following. Note that you can load BAM files if you wish to compare different samples or the results of different mapping programs.

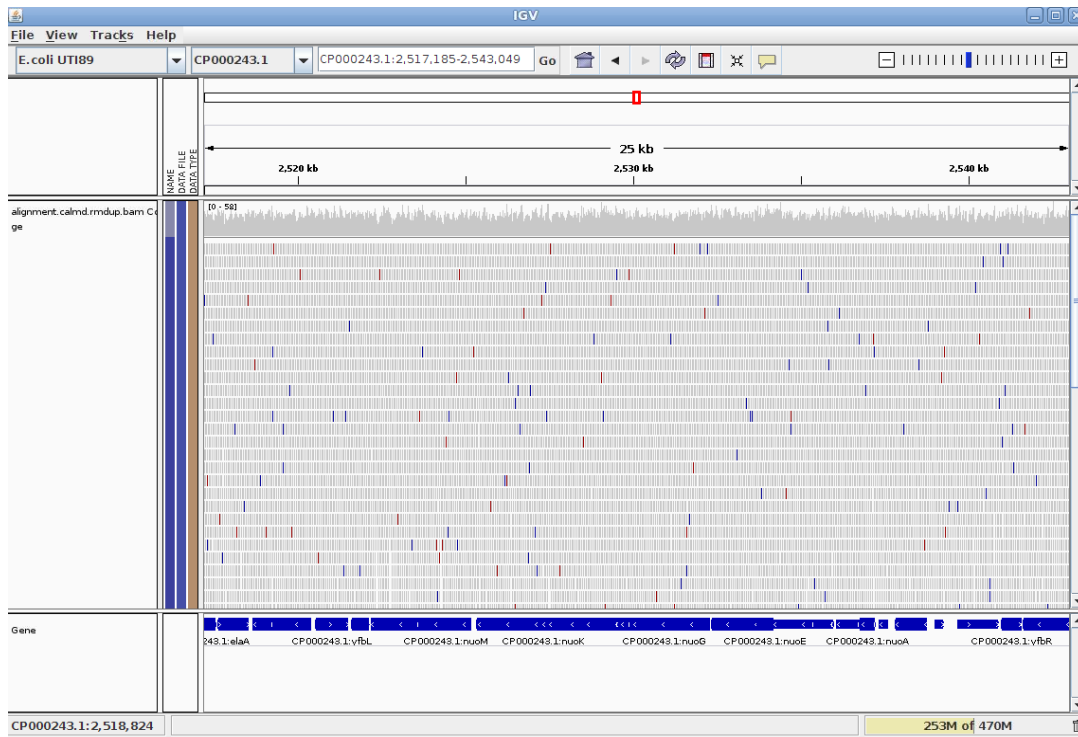


Note that at the moment, you cannot see any reads. We first need to select a chromosome and zoom in. To save memory, IGV will only display reads at a certain magnification (you can alter this in the View -> Preferences -> Alignments menu).

Select the CP000243.1 chromosome.



Either use the the +/- keys to zoom in or use the zoom bar at the top right of the screen. At the bottom you can see a representation of the genes that have been annotated for the reference strain. The following is a snapshot of the genome at position 25171850-2543949 – a 25kb window on the genome.



The gray graph at the top of the figure indicates the coverage of the genome by the grey reads below. 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 below). 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.

Above the coverage graph is a representation of each read as it is mapped to the genome. Any areas of mismatch either due to inconsistent distances between paired-end reads or due to mutations are highlighted by a colour. If there is a consistent column of colour, this is indicative of a SNP or Indel. The brighter the colour, the higher the base-calling quality is estimated to be.

2.5: 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 18: Read about the alignment display format

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

Task 19: Manually Identify a region without reads mapping

What is this region? Is there a gene close-by? What sort of genes seem to result in areas of low/zero coverage? Why do you think this is? (think about repetitive sequences, what does BWA do if a read maps to multiple locations?)

Task 20: Identify SNPs and Indels manually

Zoom into the region at the start of chromosome CP000243.1. 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 and how bright the base-calls are).

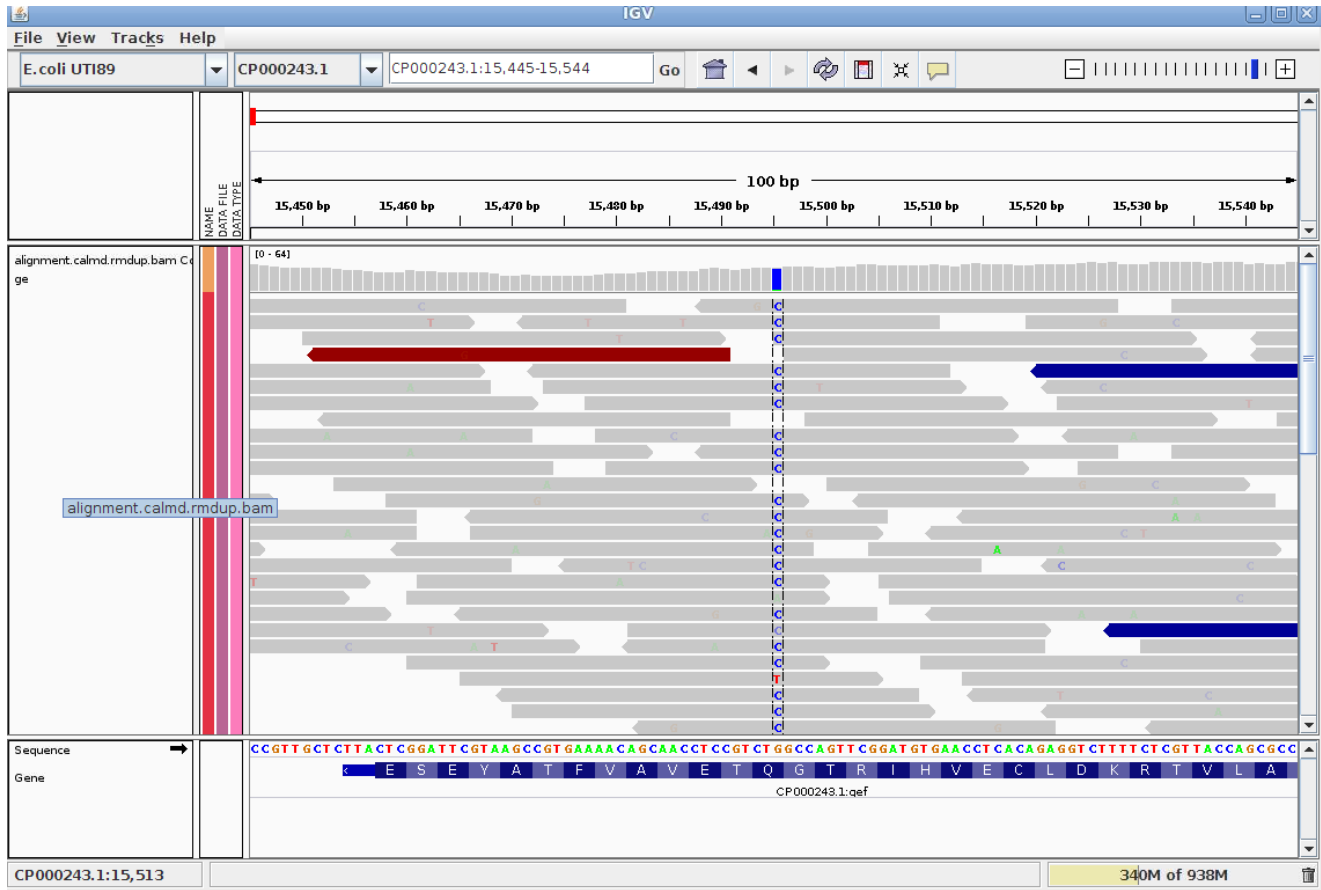
Zoom in to maximum magnification at the site of the SNP. Can you determine whether a SNP results in a synonymous (i.e. silent) or non-synonymous change in the amino acid? Can you use PDB (<http://www.rcsb.org/pdb/home/home.do>) or other resources to determine whether or not this occurs in a catalytic site or other functionally crucial region? (Note this may not always be possible).

What effect do you think this would have on the cell?

Example: Identifying SNPs manually

To help you there is an example pasted below for position 15,450-15,540bp on chromosome CP000243.1.

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 G->C SNP. This changes the codon from CAG->GAG (remember this gene is on the reverse strand as indicated by the arrows) and results in a Gln->Thu mutation in the final protein product. As this is a cell-killing gene, this could be a very important change. Note that there is a single T base listed in one of the reads towards the bottom of the screen. Based on the frequency of C-bases vs T-bases at this position we can say that the T-base is likely to be a sequencing error and ignore it.

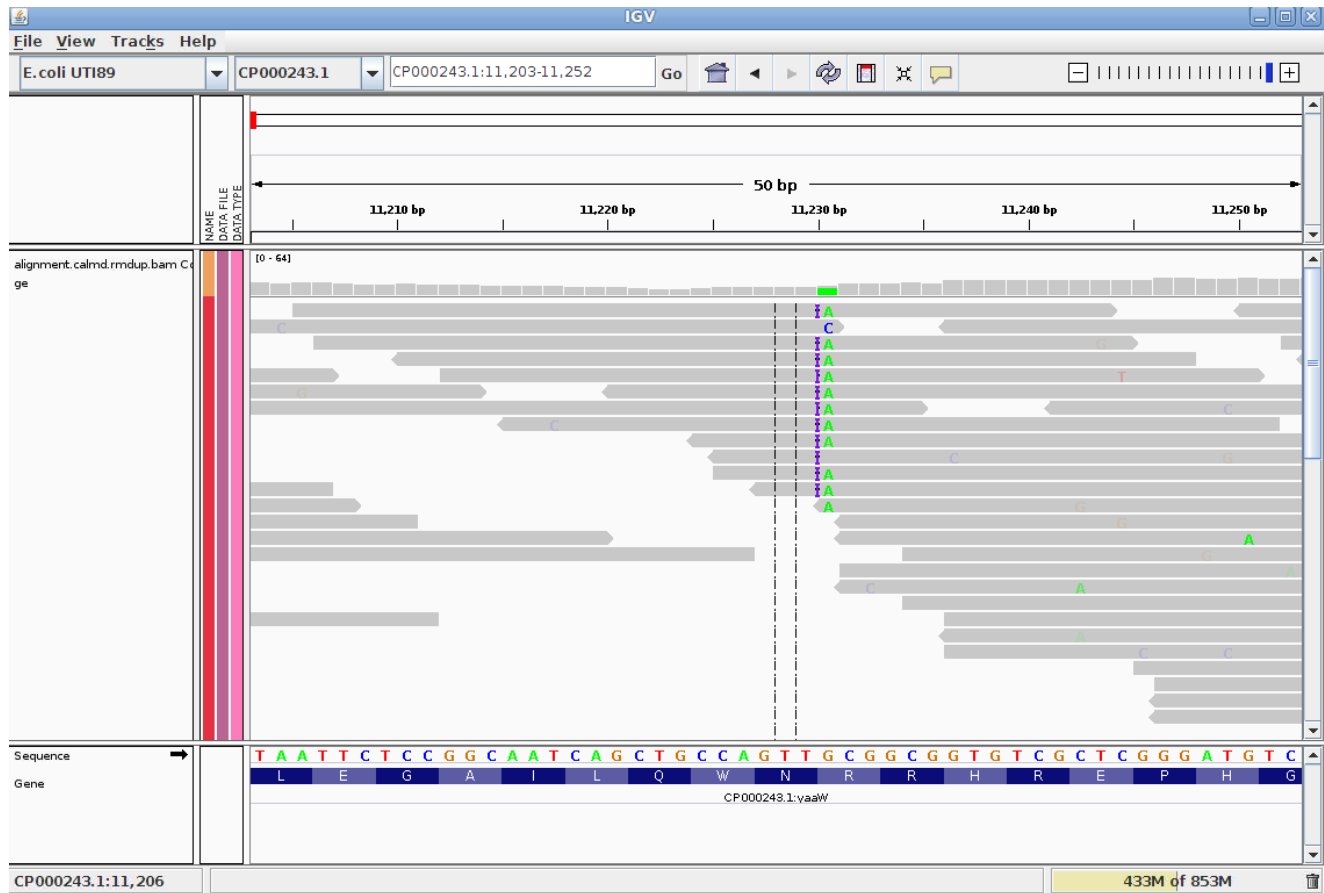
One additional check is that the SNPs occur on both read 1 and read 2. We can check this by looking at the direction of the grey reads. We can see that approximately half of the bases reporting the G->C mutation occur in read 1 (forward arrow), and half in read 2 (reverse arrow). This adds confidence 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 20-30x 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.

Example: Identifying Indels manually

To help you there is an example pasted below for position 11,200-11,250bp on chromosome CP000243.1.



Much the same guidelines apply for indels as they do for SNPs. Here we have an A->G SNP which is preceded by the insertion of a C base in our sample compared to the reference. Again, we can see how much confidence we have that the insertion is real by the brightness of the colours and by ensuring that the indel is found on both read 1 and read 2.

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

We can hover our mouse over the reference sequence to get details of the gene. We can see that it is a cytochrome C chaperone. Given that this indel changes the reading frame of the protein it could have very significant effects.

One can research the effect the effect of a SNP or Indel may have by finding the relevant gene at <http://www.uniprot.org> (or google 'yaaW uniprot' in this 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 spread-sheet friendly summary details of these.

Recap: SNP/Indel identification

1. Only changes from the reference sequence are displayed in IGV
2. Genuine SNPs/Indels should be present on reads in both the forward and reverse direction (otherwise it could be a PCR artifact)
3. Genuine SNPs/Indels should be supported by a good (i.e. 20-30x) depth of coverage (so that we're sure it's not just noise).
4. Very important mutations (e.g. ones relied upon in a paper) should be confirmed via PCR/Sanger sequencing.

2.6: Automated analyses

Viewing alignments is useful when convincing yourself or others that a particular mutation is real rather than artefactual 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).

2.6.1: Automated variant calling

To call variants we can use a number of packages (e.g. VarScan, GATK, FreeBayes). However here, we will show you how to use the bcftools package which comes with samtools. Note, that this is a simplified method because we only have a single sample. Towards the end of this tutorial we will look at how to call SNPs when we have multiple samples. By calling SNPs in multiple samples at the same time we can leverage data across different samples (e.g. one sample may have low coverage in a particular region which would normally prevent variant calling, but by looking at the sample in the context of the other samples we can increase our statistical power enabling us to call variants).

First we need to generate a pileup file which contains only locations with the variants and pass this to bcftools.

Task 21: Identify SNPs and Indels using automated variant callers

Make sure you are in the ~/genomics_tutorial/strain1/remapping_to_reference directory. Type the following:

```
samtools mpileup
```

You should see a screen similar to the following

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

Input options:

- 6 assume the quality is in the Illumina-1.3+ encoding
- A count anomalous read pairs
- B disable BAQ computation
- b FILE list of input BAM files [null]
- C INT parameter for adjusting mapQ; 0 to disable [0]
- d INT max per-BAM depth to avoid excessive memory usage [250]
- E extended BAQ for higher sensitivity but lower specificity
- f FILE faidx indexed reference sequence file [null]
- G FILE exclude read groups listed in FILE [null]
- l FILE list of positions (chr pos) or regions (BED) [null]
- M INT cap mapping quality at INT [60]
- r STR region in which pileup is generated [null]
- R ignore RG tags
- q INT skip alignments with mapQ smaller than INT [0]
- Q INT skip bases with baseQ/BAQ smaller than INT [13]

Output options:

- D output per-sample DP in BCF (require -g/-u)
- g generate BCF output (genotype likelihoods)
- O output base positions on reads (disabled by -g/-u)
- s output mapping quality (disabled by -g/-u)
- S output per-sample strand bias P-value in BCF (require -g/-u)
- u generate uncompress BCF output

SNP/INDEL genotype likelihoods options (effective with ``-g'` or ``-u'`):

- e INT Phred-scaled gap extension seq error probability [20]
- F FLOAT minimum fraction of gapped reads for candidates [0.002]
- h INT coefficient for homopolymer errors [100]
- I do not perform indel calling
- L INT max per-sample depth for INDEL calling [250]
- m INT minimum gapped reads for indel candidates [1]
- o INT Phred-scaled gap open sequencing error probability [40]
- P STR comma separated list of platforms for indels [all]

Notes: Assuming diploid individuals.

If you are running this on your own datasets, please make sure you set the `-d` parameter if you have high coverage (i.e. > 200x mean coverage) per sample.

As the samtools mpileup command outputs an unfriendly output, we will pass it directly to the bcftools view command using the linux pipe (`|`). Type the following (don't forget the `-` after the `-bvcg`):

```
samtools mpileup -uf ~/genomics_tutorial/reference_sequence/Ecoli_UTI89.fna
alignment.rmdup.sorted.bam | bcftools view -bvcg - > var.raw.bcf
```

This may take 15 minutes or so and will generate a BCF (Binary Call Format) file containing the raw unfiltered variant calls in a binary format.

This is not readable by humans, so let's use the `bcftools view` command and use the linux pipe (`|`) with the `vcfutils.pl varFilter` command. We can see what the options are for this program by typing in:

vcfutils.pl varFilter

Usage: `vcfutils.pl varFilter [options] <in.vcf>`

Options: `-Q INT` minimum RMS mapping quality for SNPs [10]
`-d INT` minimum read depth [2]
`-D INT` maximum read depth [10000000]
`-a INT` minimum number of alternate bases [2]
`-w INT` SNP within INT bp around a gap to be filtered [3]
`-W INT` window size for filtering adjacent gaps [10]
`-1 FLOAT` min P-value for strand bias (given PV4) [0.0001]
`-2 FLOAT` min P-value for baseQ bias [1e-100]
`-3 FLOAT` min P-value for mapQ bias [0]
`-4 FLOAT` min P-value for end distance bias [0.0001]
`-e FLOAT` min P-value for HWE (plus $F < 0$) [0.0001]
`-p` print filtered variants

Note: Some of the filters rely on annotations generated by SAMtools/BCFtools.

We will use the `-d` option to limit variant calls to those positions where there are at least 10 reads. Type:

bcftools view var.raw.bcf | vcfutils.pl varFilter -d 10 > var.flt.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 FORMAT alignment.calmd.rmdup.bam
CP000243.1 10739 . ATTT ACTTT 214 . INDEL;DP=32;VDB=0.0803;AF1=1;AC1=2;DP4=0,0,8,11;MQ=51;FQ=-91.5 GT:PL:GQ 1/1:255,57,0:99
CP000243.1 11229 . T TC 214 . INDEL;DP=34;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,14,11;MQ=44;FQ=-110 GT:PL:GQ 1/1:255,75,0:99
CP000243.1 15495 . G C 222 . DP=50;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,25,17;MQ=52;FQ=-150 GT:PL:GQ 1/1:255,123,0:99
CP000243.1 35301 . G A 222 . DP=47;VDB=0.0948;AF1=1;AC1=2;DP4=0,0,14,26;MQ=51;FQ=-147 GT:PL:GQ 1/1:255,120,0:99
CP000243.1 126378 . T A 222 . DP=47;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,18,20;MQ=54;FQ=-141 GT:PL:GQ 1/1:255,114,0:99
CP000243.1 131200 . C CA 214 . INDEL;DP=58;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,19,15;MQ=54;FQ=-137 GT:PL:GQ 1/1:255,102,0:99
CP000243.1 210716 . G C 211 . DP=45;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,19,19;MQ=47;FQ=-141 GT:PL:GQ 1/1:244,114,0:99
CP000243.1 314746 . A T 222 . DP=34;VDB=0.0948;AF1=1;AC1=2;DP4=0,0,12,14;MQ=53;FQ=-105 GT:PL:GQ 1/1:255,78,0:99
CP000243.1 326644 . G C 222 . DP=43;VDB=0.0803;AF1=1;AC1=2;DP4=0,0,18,19;MQ=47;FQ=-138 GT:PL:GQ 1/1:255,111,0:99
CP000243.1 336724 . A G 218 . DP=49;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,20,19;MQ=45;FQ=-144 GT:PL:GQ 1/1:251,117,0:99
CP000243.1 444808 . G A 153 . DP=32;VDB=0.0624;AF1=1;AC1=2;DP4=0,0,12,10;MQ=44;FQ=-93 GT:PL:GQ 1/1:186,66,0:99
CP000243.1 444809 . C A 138 . DP=33;VDB=0.0803;AF1=1;AC1=2;DP4=0,0,12,10;MQ=41;FQ=-93 GT:PL:GQ 1/1:171,66,0:99
```

You can see the chromosome, position, reference and alternate allele as well as a quality score for the SNP. This a VCF file (Variant Call File). This is a standard developed for the 1000 genomes project. The full specification is given at <http://www.1000genomes.org/wiki/Analysis/Variant%20Call%20Format/vcf-variant-call-format-version-41>

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

Variant qualities:

Typically one should only accept variant calls over a certain quality threshold. Typically a threshold of 60 is used (i.e. a 1 in 1000000 chance of a mis-called variant). Here you can see that all these variants would pass these thresholds. However, for future reference, we can use the Linux 'awk' command to filter the data on the quality column (i.e. column 6, which in the awk command is denoted by \$6):

```
awk '($6>=60)' var.ft.vcf > out.snps.vcf4
```

Again viewing the final output file out.snps.vcf4 using a text-editor or the 'more' command should yield:

```
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT alignment.calmd.rmdup.bam
CP000243.1 10739 . ATTT ACTTT 214 . INDEL;DP=32;VDB=0.0803;AF1=1;AC1=2;DP4=0,0,8,11;MQ=51;FQ=-91.5 GT:PL:GQ 1/1:255,57,0:99
CP000243.1 11229 . T TC 214 . INDEL;DP=34;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,14,11;MQ=44;FQ=-110 GT:PL:GQ 1/1:255,75,0:99
CP000243.1 15495 . G C 222 . DP=50;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,25,17;MQ=52;FQ=-150 GT:PL:GQ 1/1:255,123,0:99
CP000243.1 35301 . G A 222 . DP=47;VDB=0.0948;AF1=1;AC1=2;DP4=0,0,14,26;MQ=51;FQ=-147 GT:PL:GQ 1/1:255,120,0:99
CP000243.1 126378 . T A 222 . DP=47;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,18,20;MQ=54;FQ=-141 GT:PL:GQ 1/1:255,114,0:99
CP000243.1 131200 . C CA 214 . INDEL;DP=58;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,19,15;MQ=54;FQ=-137 GT:PL:GQ 1/1:255,102,0:99
CP000243.1 210716 . G C 211 . DP=45;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,19,19;MQ=47;FQ=-141 GT:PL:GQ 1/1:244,114,0:99
CP000243.1 314746 . A T 222 . DP=34;VDB=0.0948;AF1=1;AC1=2;DP4=0,0,12,14;MQ=53;FQ=-105 GT:PL:GQ 1/1:255,78,0:99
CP000243.1 326644 . G C 222 . DP=43;VDB=0.0803;AF1=1;AC1=2;DP4=0,0,18,19;MQ=47;FQ=-138 GT:PL:GQ 1/1:255,111,0:99
CP000243.1 336724 . A G 218 . DP=49;VDB=0.1028;AF1=1;AC1=2;DP4=0,0,20,19;MQ=45;FQ=-144 GT:PL:GQ 1/1:251,117,0:99
CP000243.1 444808 . G A 153 . DP=32;VDB=0.0624;AF1=1;AC1=2;DP4=0,0,12,10;MQ=44;FQ=-93 GT:PL:GQ 1/1:186,66,0:99
CP000243.1 444809 . C A 138 . DP=33;VDB=0.0803;AF1=1;AC1=2;DP4=0,0,12,10;MQ=41;FQ=-93 GT:PL:GQ 1/1:171,66,0:99
```

This forms our definitive list of variants for this sample.

Task 22: Compare the variants found using this method to those you found in section 2.5

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.

(Hint – check out the SNP/Indel combination in the example in section 3.5 again)

2.6.2: Quickly locating genes which are missing compared to the reference

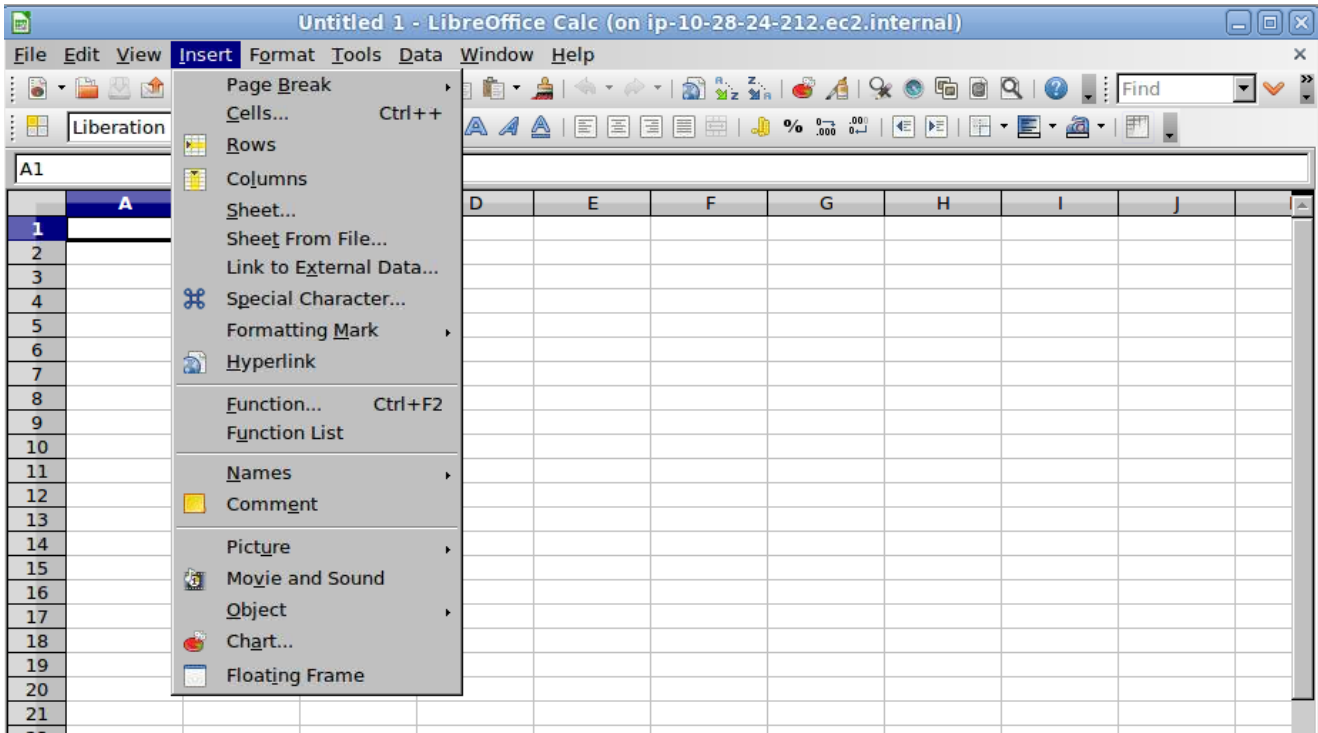
We can use a command from the BEDTools package (<http://code.google.com/p/bedtools/>) to identify annotated genes which are not covered by reads across their full length.

Type the following one one line:

```
coverageBed -abam alignment.rmdup.sorted.bam -b
~/genomics_tutorial/reference_sequence/Ecoli_UTI89.gff > gene_coverage.txt
```

This should only take a minute or so. Once complete we can view the file in LibreOffice spreadsheet.

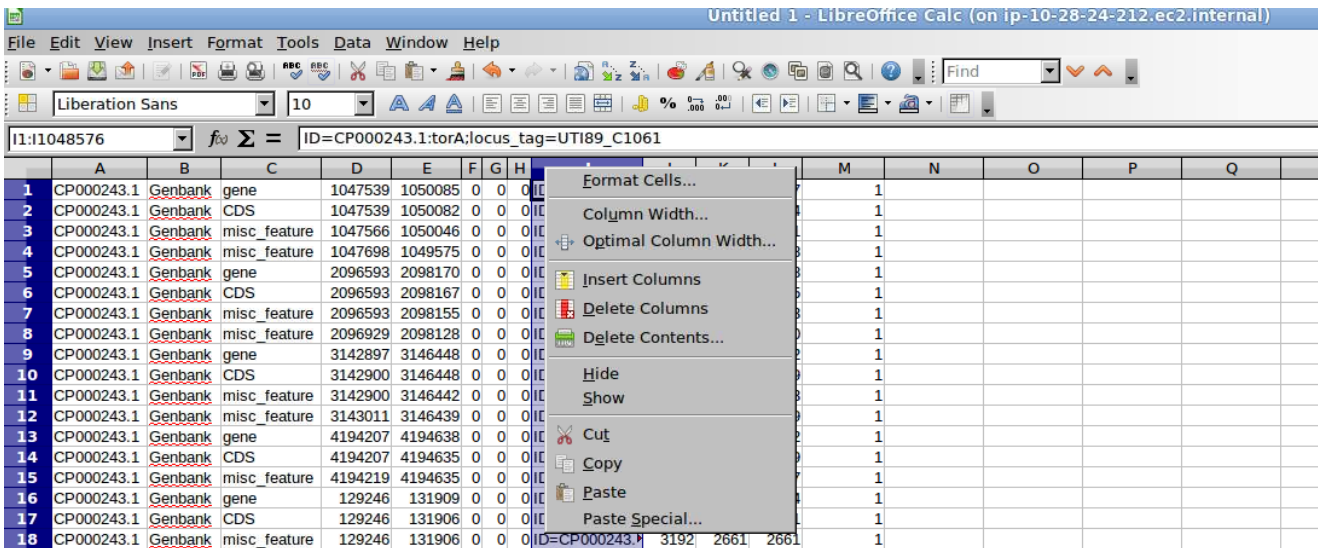
Launch 'LibreOffice Calc' from the Applications->Office menu.



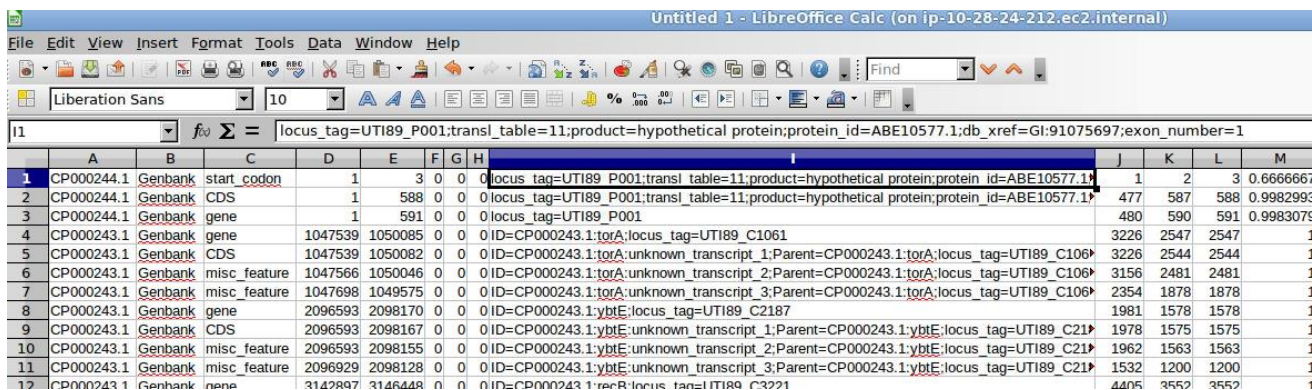
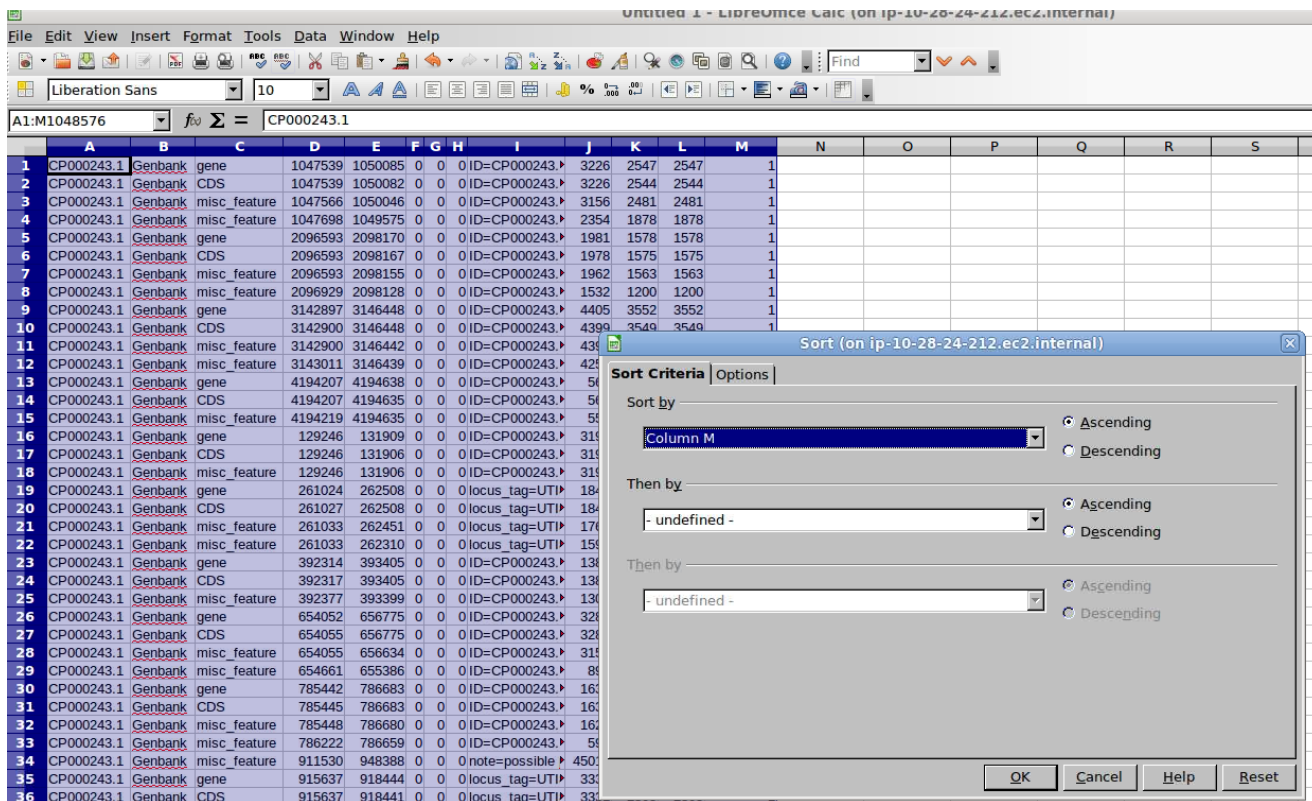
Click 'Insert' and then select 'Sheet From File...'. Select the gene_coverage.txt file in the ~/genomics_tutorial/strain1/remapping_to_reference directory.

When the program asks you which delimiters to use, select 'Tab'. Make sure the others are deselected.

You may need to right click on the column labels for column I and select 'Column width'. Reduce this to 5”.



Column M represents the coverage of the particular GFF entry. A value of 1 is fully covered by reads. Less than one indicates regions with no reads mapping. Let's sort on column M in ascending order so that anything missing with missing chunks shows up at the top. Select all columns and click on Data->Sort...



There appear to be very few missing regions. In fact there is only one missing base near the origin of replication for this bacteria. Because the aligner doesn't know that the genome is circular (and thus to allow reads to span the beginning and end of the chromosome), the very first base is missing from the alignment. It is however more than likely to be present in the reads.

2.6.3: Evaluating the impact of variants

So, aside from a few variants we've discovered, this strain of *E. coli* does not seem to have lost anything and seems very closely related to the reference genome. So let's take a closer look at the variants. We'd like to obtain a list of genes in which these variants occur and whether they result in amino acid changes.

To do this we'll use the SnpEff program (<http://snpeff.sourceforge.net/index.html>) in conjunction with the filtered VCF file we've created.

The relevant database has already been installed. However, if we try to run the pipeline now, we will get an error stating that the chromosome name cannot be found. This is because the chromosome ID in our version of the *E. coli* annotation is different from that in the SnpEff database.

This is a common problem – there are several standards for chromosome IDs.

Try running the pipeline using:

```
java -Xmx4g -jar ~/software/snpEff/snpEff.jar -v  
Escherichia_coli_UTI89_uid58541 out.snps.vcf4 > snp.eff.vcf
```

The output indicates that the chromosome names should be 'NC_007946' and 'NC_007941'. As these chromosome names are not found an error is detected.

```
#          START codon errors :    751 ( 14.55% )  
#          STOP codon warnings :      0 (  0.00% )  
#          Total Errors :    751 ( 14.55% )  
# Cds      : 5162  
# Exons    : 5162  
# Exons with sequence : 5162  
# Exons without sequence : 0  
# Avg. exons per transcript : 1.00  
# WARNING  : No mitochondrion chromosome found  
# Number of chromosomes : 2  
# Chromosomes names [sizes] : 'NC_007946' [5065741]      'NC_007941' [114230]  
00:00:05.538    Predicting variants  
  
ERRORS: Some errors were detected  
Error type      Number of errors  
ERROR_CHROMOSOME_NOT_FOUND      12  
  
00:00:05.595    Creating summary file: snpEff_summary.html  
00:00:06.326    Creating genes file: snpEff_genes.txt  
00:00:07.188    done.  
00:00:07.201    Logging  
00:00:08.203    Checking for updates...
```


If we look at the out.snps.vcf4 file we can see that we have a different chromosome ID to what is in the snpEff database:

more out.snps.vcf4

```
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT alignment.rmdup.sorted.bam
CP000243.1 10738 . G GC 174 . INDEL;IS=13,0.541667;DP=24;QS=0.000000
1;AC1=2;DP4=0,0,5,5;MQ=56;FQ=-64.5 GT:PL:GQ 1/1:215,30,0:57
CP000243.1 11230 . G GA 214 . INDEL;IS=1,0.052632;DP=19;QS=0.000000
1;AC1=2;DP4=0,0,9,8;MQ=56;FQ=-85.5 GT:PL:GQ 1/1:255,51,0:99
CP000243.1 15495 . G C 222 . DP=44;QS=0.000000,0.972101,0.027899,0
4;MQ=59;FQ=-135 GT:PL:GQ 1/1:255,108,0:99
CP000243.1 35301 . G A 222 . DP=41;QS=0.000000,1.000000,0.000000
```

So, what we need to do is change the CP000243.1 chromosome ID to the NC_007946 ID snpEff expects. We'll also need to change the plasmid ID CP000244.1 to NC_007941.

Fortunately in Unix this is easy to do. We can just use the 'sed' command. First we'll make a copy of the original out.snps.vcf4 file as we'll need it later.

Type:

```
cp out.snps.vcf4 out.snps.vcf4.renamed
```

```
sed -i 's/CP000243.1/NC_007946/g' out.snps.vcf4.renamed
```

and

```
sed -i 's/CP000244.1/NC_007941/g' out.snps.vcf4.renamed
```

The -i flag tells sed to alter the file in-place rather than output the changes to the screen.

Look again at the out.snps.vcf4.renamed file and you should see that the changes have been made. Obviously in real-life you need to make sure that these errors really are due to a naming convention issue and not because you are using the wrong reference sequence!

```
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT alignment.rmdup.sorted.bam
NC_007946 10738 . G GC 174 . INDEL;IS=13,0.541667;DP=24;QS=0.000000
1;AC1=2;DP4=0,0,5,5;MQ=56;FQ=-64.5 GT:PL:GQ 1/1:215,30,0:57
NC_007946 11230 . G GA 214 . INDEL;IS=1,0.052632;DP=19;QS=0.000000
1;AC1=2;DP4=0,0,9,8;MQ=56;FQ=-85.5 GT:PL:GQ 1/1:255,51,0:99
NC_007946 15495 . G C 222 . DP=44;QS=0.000000,0.972101,0.027899,0
4;MQ=59;FQ=-135 GT:PL:GQ 1/1:255,108,0:99
NC_007946 35301 . G A 222 . DP=41;QS=0.000000,1.000000,0.000000
```

We can now re-run snpEff.

```
java -Xmx4g -jar ~/software/snpEff/snpEff.jar -v
Escherichia_coli_UTI89_uid58541 out.snps.vcf4.renamed > snp.eff.vcf
```

That should complete quite quickly.

```
00:00:02.538 done
00:00:02.589 Building interval forest
00:00:04.085 done.
00:00:04.086 Genome stats :
# Genome name : 'Escherichia_coli_UTI89_uid58541'
# Genome version : 'Escherichia_coli_UTI89_uid58541'
# Has protein coding info : true
# Genes : 5272
# Protein coding genes : 5162
# Transcripts : 5162
# Avg. transcripts per gene : 0.98
# Protein coding transcripts : 5162
# Length errors : 0 ( 0.00% )
# STOP codons in CDS errors : 0 ( 0.00% )
# START codon errors : 751 ( 14.55% )
# STOP codon warnings : 0 ( 0.00% )
# Total Errors : 751 ( 14.55% )
# Cds : 5162
# Exons : 5162
# Exons with sequence : 5162
# Exons without sequence : 0
# Avg. exons per transcript : 1.00
# WARNING : No mitochondrion chromosome found
# Number of chromosomes : 2
# Chromosomes names [sizes] : 'NC_007946' [5065741] 'NC_007941' [114230]
00:00:05.464 Predicting variants
00:00:05.576 Creating summary file: snpEff_summary.html
00:00:06.781 Creating genes file: snpEff_genes.txt
00:00:10.067 done.
00:00:10.070 Logging
00:00:11.077 Checking for updates...
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/remapping_to_reference$ ls
alignment.rmdup.sorted.bam mappingstats.txt snpEff_genes.txt snp.eff.vcf var.raw.bcf
alignment.rmdup.sorted.bam.bai out.snps.vcf4 snpEff_summary.html var.flt.vcf
```

You can see that snpEff has created some new files. snpEff_genes.txt, snpEff_summary.html and snp.eff.vcf.

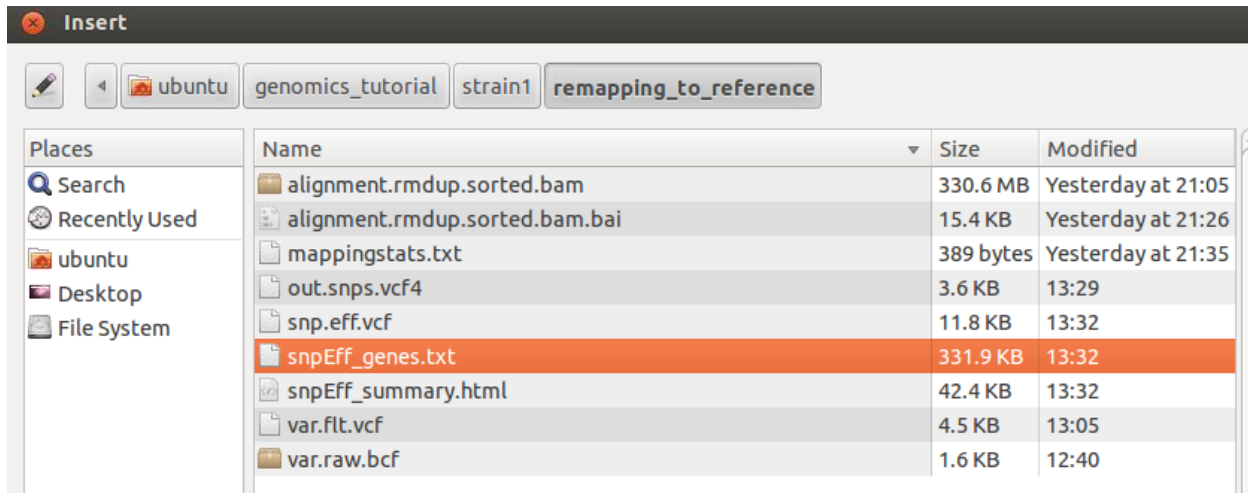
We can view the summary information in a web-browser:

firefox snpEff_summary.html

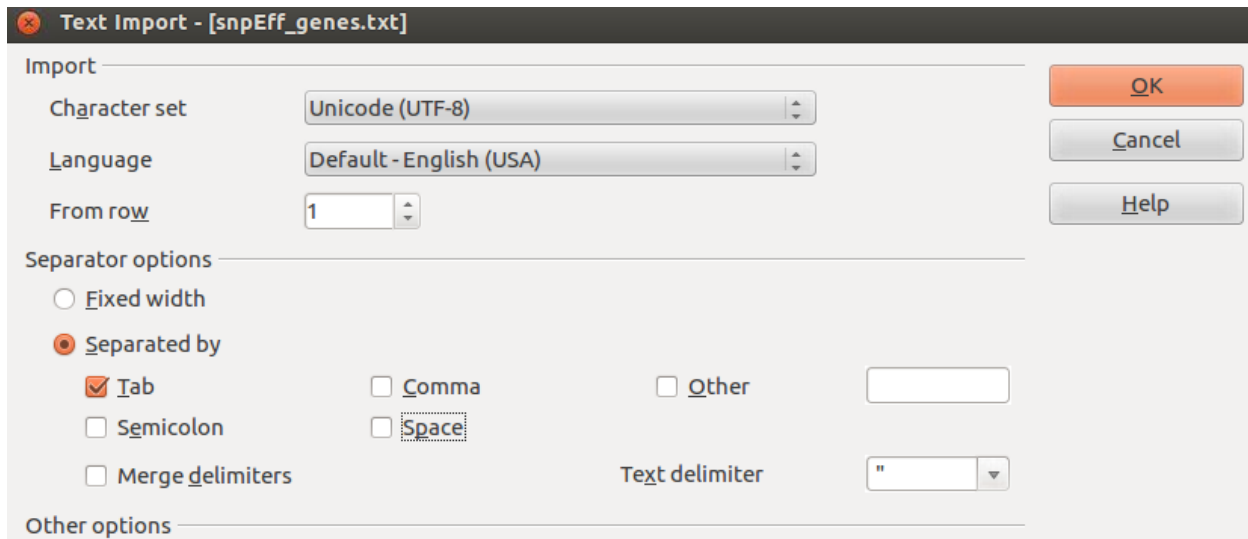
Take a look at this to see what kind of information you can extract. Would this be useful for your projects? How many mutations were found? Of what type? What do the high/low/moderate classifications of impact correspond to?

Look at the snpEff website to see whether your species of interest is included in the list of databases (<http://snpeff.sourceforge.net/download.html#databases>). If you have a GFF file and a FASTA file they can be imported even if they are not on the list (http://snpeff.sourceforge.net/SnpEff_manual.html#databases).

Then we can look at the detailed gene-by-gene information snpEff has provided us with. Load up the snpEff_genes.txt file in LibreOffice Calc (you'll need to use Insert-> Sheet from file).



Make sure you *only* select the 'tab' as the delimiter:



Untitled 1 - LibreOffice Calc

File Edit View Insert Format Tools Data Window Help

LibreOffice Calc toolbar

A1 f(x) Σ = # The following table is formatted as tab separated values.

	A	B	C	D	E	F	G	H	I
1	# The following table is formatted as tab separated values								
2	#GeneId	GeneName	BioType	Bases affected (DOWNSTREAM)	Total score (DOWNSTREAM)	Length (DOWNSTREAM)	Bases affected (EXON)	Total score (EXON)	Length
3	UT189_C0001	thrL		0	0	5000	0	0	
4	UT189_C0002	thrA		0	0	5000	0	0	
5	UT189_C0003	thrB		0	0	5000	0	0	
6	UT189_C0004	thrC		0	0	5000	0	0	
7	UT189_C0005	yaaX		0	0	5000	0	0	
8	UT189_C0006	UT189_C0006		0	0	5000	0	0	
9	UT189_C0007	yaaA		0	0	5000	0	0	
10	UT189_C0008	yaaJ		0	0	5000	0	0	
11	UT189_C0009	talB		0	0	5000	0	0	
12	UT189_C0010	mogA		0	0	5000	0	0	
13	UT189_C0011	yaaH		0	0	5000	0	0	
14	UT189_C0012	yaaW		0	0	5000	2	0	
15	UT189_C0014	yaaI		2	0	5000	0	0	
16	UT189_C0015	UT189_C0015		0	0	5000	0	0	
17	UT189_C0016	dnaK		1	0	5000	0	0	
18	UT189_C0017	dnaJ		0	0	5000	0	0	
19	UT189_C0018	gef		0	0	5000	1	0	
20	UT189_C0019	UT189_C0019		0	0	5000	0	0	
21	UT189_C0020	UT189_C0020		0	0	5000	0	0	
22	UT189_C0021	UT189_C0021		0	0	5000	0	0	
23	UT189_C0022	nhaA		0	0	5000	0	0	
24	UT189_C0023	nhaR		0	0	5000	0	0	
25	UT189_C0024	UT189_C0024		0	0	5000	0	0	
26	UT189_C0025	rpsI		0	0	5000	0	0	
27	UT189_C0026	yaaY		0	0	5000	0	0	

Sheet 1 / 4 Default STD Sum=0 100%

Delete the top row “#The following table is formatted as tab separated values”. Then sort as follows:

Sort

Sort Criteria Options

Sort by

Count (SYNONYMOUS_CODING) Ascending Descending

Then by

Count (NON_SYNONYMOUS_CODING) Ascending Descending

Then by

Count (FRAME_SHIFT) Ascending Descending

OK Cancel Help Reset

What are the genes with non-synonymous mutations involved with?

Finally, we can look at the snp.eff.vcf file. Compare this with the file we put into snpEff (out.snps.vcf4.renamed). What extra information has snpEff added?

Determine the effect of variants using an alternative quick and dirty script

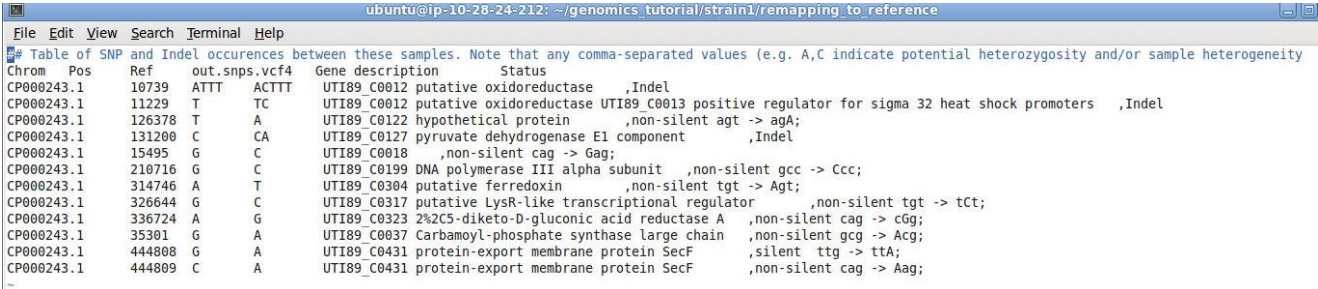
Later on in the tutorial we will need to compare the SNPs between samples. Although SnpEff is very good with single samples, it does not lend itself well to dealing with multiple samples. To overcome this, we have commandeered a script written by David Studholme and Konrad Paszkiewicz. This provides limited information on the effect of a given SNP, but the output is a bit easier to deal with when we have large numbers of samples.

Type (all on one line):

```
/usr/local/scripts/snp_comparator.pl 10  
~/genomics_tutorial/reference_sequence/Ecoli_UTI89.fna  
~/genomics_tutorial/reference_sequence/Ecoli_UTI89.gff out.snps.vcf4 >  
snp_report.txt
```

This program takes the information from the reference sequence and annotation, and the VCF snp files and determines whether the variant occurs within a gene, and if so the effect of each mutation.

Once complete, view the snp_report.txt file using the more command:



```
ubuntu@ip-10-28-24-212: ~/genomics_tutorial/strain1/remapping_to_reference
File Edit View Search Terminal Help
# Table of SNP and Indel occurrences between these samples. Note that any comma-separated values (e.g. A,C indicate potential heterozygosity and/or sample heterogeneity)
Chrom Pos Ref out.snps.vcf4 Gene description Status
CP000243.1 10739 A TTT ACTTT UTI89_C0012 putative oxidoreductase ,Indel
CP000243.1 11229 T TC UTI89_C0012 putative oxidoreductase UTI89_C0013 positive regulator for sigma 32 heat shock promoters ,Indel
CP000243.1 126378 T A UTI89_C0122 hypothetical protein ,non-silent agt -> agA;
CP000243.1 131200 C CA UTI89_C0127 pyruvate dehydrogenase E1 component ,Indel
CP000243.1 15495 G C UTI89_C0018 ,non-silent cag -> Gag;
CP000243.1 210716 G C UTI89_C0199 DNA polymerase III alpha subunit ,non-silent gcc -> Ccc;
CP000243.1 314746 A T UTI89_C0304 putative ferredoxin ,non-silent tgt -> Agt;
CP000243.1 326644 G C UTI89_C0317 putative LysR-like transcriptional regulator ,non-silent tgt -> tCt;
CP000243.1 336724 A G UTI89_C0323 2%2C5-diketo-D-gluconic acid reductase A ,non-silent cag -> cGg;
CP000243.1 35301 G A UTI89_C0037 Carbamoyl-phosphate synthase large chain ,non-silent gcg -> Acg;
CP000243.1 444808 G A UTI89_C0431 protein-export membrane protein SecF ,silent ttg -> ttA;
CP000243.1 444809 C A UTI89_C0431 protein-export membrane protein SecF ,non-silent cag -> Aag;
```

Later on we will see how we can use this program to compare results between different strains.

Task 24: Check each variant in IGV

N.B. If a variant doesn't seem to match what the snp_report file says, check the reverse reading frames.

Task 25: Check each non-synonymous SNP and Indel for a possible link to haemorrhaging phenotypes (often known as EHEC)

Use the web for this. Search UniProt (<http://www.uniprot.org>) or other databases for evidence that any of these proteins may be linked to the outbreak.

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 installment 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.

2014 Workshop on Genomics

Part 3: Short read genomics: Assembly of unmapped reads

Instructors:

- Konrad Paszkiewicz k.h.paszkiwicz@exeter.ac.uk

Objectives:

By the end of the workshop you will be expected to be able to:

- Extract reads which do not map to the reference sequence
- Assemble these reads de novo using Velvet
- Generate summary statistics for the assembly
- Identify potential genes within the assembly
- Search for matches within the Swissprot database via BLAST and against the Pfam database
- Visualize the taxonomic distribution of BLAST hits
- Perform gene prediction and annotation using RAST

3.1 Introduction

In this section of the workshop we will continue the analysis of a strain of *E. coli* which is involved in urinary tract infections. 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 Velvet, generate assembly statistics and then produce some annotation via Pfam, BLAST and RAST.

3.2 Extraction and QC of unmapped reads

Task 1: Extract the unmapped reads

First of all make sure you are in the ~/genomics_tutorial/strain1 directory (hint: use the cd command). Then create a directory called denovo_assembly in which we will do our de novo assembly and analysis.

```
mkdir denovo_assembly/
```

Then move to that directory:

```
cd denovo_assembly/
```

Now we will use the bam2fastq program (<http://www.hudsonalpha.org/gsl/software/bam2fastq.php>) 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/alignment.rmdup.sorted.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. Below we can see that the program has successfully created the two files.

```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assembly$ bam2fastq --no-aligned -o unaligned\#.fastq ../remapping_to_reference/alignment.rmdup.sorted.bam  
This looks like paired data from lane 0.  
Output will be in unaligned_1.fastq and unaligned_2.fastq  
6494900 sequences in the BAM file  
385214 sequences exported  
WARNING: 77266 reads could not be matched to a mate and were not exported  
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assembly$
```

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 (hint! Using grep). 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.

3.3 De novo assembly

We will be using the Velvet (<http://www.ebi.ac.uk/~zerbino/velvet/>) assembler to stitch together the unmapped reads. This will enable us to obtain large enough contigs to determine which genes, plasmids or contaminant material is present (if any!).

Task 3: Learn more about de novo assemblers

To understand more about de-novo assemblers, read the technical note at:

http://res.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.

Task 4: Generate the Velvet hash table

Velvet requires a hash table which stores all of the k-mer and paired-end information prior to trying to assemble the data. Note that velvet ignores quality scores, so whether you pass it a FASTA or a FASTQ file doesn't matter. However, it will automatically convert all N bases to A (this is because to save memory it can only represent 4 bases internally). It pays to remove sequences containing Ns at the initial filtering stage to avoid introducing ambiguities.

```
velveth assembly/ 31 -fastq -shortPaired -separate unaligned_1.fastq
unaligned_2.fastq
```

```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assembly$ velveth assembly/ 31 -fastq -shortPaired -separate unaligned_1.fastq unaligned_2.fastq
[0.000001] Reading FastQ file unaligned_1.fastq;
[0.000125] Reading FastQ file unaligned_2.fastq;
[0.700833] 307948 sequences found in total in the paired sequence files
[0.700886] Done
[0.701014] Reading read set file assembly//Sequences;
[0.750782] 307948 sequences found
[1.086043] Done
[1.086110] 307948 sequences in total.
[1.086203] Writing into roadmap file assembly//Roadmaps...
[1.222899] Inputting sequences...
[1.222979] Inputting sequence 0 / 307948
[3.220293] === Sequences loaded in 2.008558 s
[3.220409] Done inputting sequences
[3.220432] Destroying splay table
[3.367367] Splay table destroyed
```

This will perform a de novo assembly using a k-mer length of 31 and put the results into the assembly/ directory. Note that if you have a large number of reads or longer reads, you may well need to increase the k-mer length to reduce memory requirements. This will have the effect of excluding reads shorter than the k-mer size (e.g. selecting a k-mer length of 41 will exclude all reads less than 39bp long).

Task 5: Perform denovo assembly

We can now perform the de novo assembly of the unmapped reads. To do this we will use the program velvetg (stands for velvet graph). Velvetg comes with many options, and if you are performing de novo assembly of a full complex genome you may well need to tinker with these parameters to get the best results (see the Velvet manual for more detail). Here, we are using cov_cutoff 0 and exp_cov 0 to indicate that we do not want Velvet to throw any data away based on coverage metrics. This is because we are only interested in the unmapped reads and there may be variable copy numbers of inserts,

plasmids and other genetic material of interest. Type (all on one line):

```
velvetg assembly/ -cov_cutoff 0 -exp_cov 0 -unused_reads yes -  
very_clean yes
```

```
[2.997762] Removed 0 null nodes  
[2.997790] Concatenation over!  
[2.997826] Clipping short tips off graph, drastic  
[2.997851] Concatenation...  
[2.997868] Renumbering nodes  
[2.997884] Initial node count 3  
[2.997905] Removed 0 null nodes  
[2.997927] Concatenation over!  
[2.997948] 3 nodes left  
[2.997969] WARNING: NO EXPECTED COVERAGE PROVIDED  
[2.997989] Velvet will be unable to resolve any repeats  
[2.998010] See manual for instructions on how to set the expected coverage parameter  
[2.998033] Concatenation...  
[2.998054] Renumbering nodes  
[2.998074] Initial node count 3  
[2.998095] Removed 0 null nodes  
[2.998117] Concatenation over!  
[2.998138] Removing reference contigs with coverage < 0.000000...  
[2.998162] Concatenation...  
[2.998183] Renumbering nodes  
[2.998203] Initial node count 3  
[2.998224] Removed 0 null nodes  
[2.998246] Concatenation over!  
[3.000272] Writing contigs into assembly//contigs.fa...  
[3.008532] Writing into stats file assembly//stats.txt...  
[3.008778] Printing unused reads into assembly//UnusedReads.fa  
Final graph has 3 nodes and n50 of 3275, max 3275, total 3337, using 20000/307948 reads
```

Note that velvetg runs quickly. Normally this is not the case, but because we have so few reads (and because this example is artificial) it speeds through the assembly in seconds.

The options used with velvetg -cov_cutoff and -exp_cov were set to zero. -cov_cutoff specifies how frequently a k-mer should appear to avoid being thrown out as noise (i.e. sequencing errors or contamination). -exp_cov sets the expected level of coverage (i.e. how many times each k-mer should occur). This allows the assembler to make an educated guess as to how many copies of a sequence are present. This is especially useful in repetitive regions. However they are only useful when assembling single complete genomes. As we are dealing with just the unmapped reads here, it makes little sense to try to filter based on coverage since the unmapped material may represent duplicated genes or plasmids.

The -unused_reads flag and -very_clean flag just mean that velvetg should output any reads which were not included in the assembly and that it should delete as many intermediate files as possible once it has finished.

If we change to the assembly directory and list its contents we should see several files:

```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assembly/assembly$ ls  
contigs.fa Log PreGraph stats.txt UnusedReads.fa  
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assembly/assembly$ █
```

Let's take them in turn:

1. Log

This lists the version number of Velvet, as well as some summary statistics regarding the assembly. (read http://en.wikipedia.org/wiki/N50_statistic for more details about what N50 means). We'll come back to this file in the next section.

```
Fri Dec 27 23:26:39 2013
velvetg assembly/ -cov_cutoff 0 -exp_cov 0 -unused_reads yes -very_clean yes
Version 1.2.10
Copyright 2007, 2008 Daniel Zerbino (zerbino@ebi.ac.uk)
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Compilation settings:
CATEGORIES = 2
MAXKMERLENGTH = 300

Final graph has 3 nodes and n50 of 3275, max 3275, total 3337, using 20000/307948 reads
```

2. Pregraph

This is a file generated by Velvetg when generating the k-mer graph but is normally not required.

3. UnusedReads.fa

This is a FASTA formatted file containing the reads which could not be assembled into contigs larger than twice the k-mer length (i.e. $2 \times 31 = 62$ bp). This could be because they represent highly repetitive sequences, or because the sequences are shorter than the k-mer length selected. (Don't worry if your file looks different to the example below).

```
>SEQUENCE_1_length_40
CGTCTTTGGTGTACAGGACGGTCACGATGATGTGGTGCTG
>SEQUENCE_2_length_39
TGCGGGAGAGTCCCACAACACGCAGGACTCTTTTGCTTT
>SEQUENCE_3_length_39
ACCAGCACCTGTGAACAGCATTCTGTTACCATCGATGGC
>SEQUENCE_4_length_40
CGGGATAGAGATAAGCCGCCTTTAGTAGCCAAAAGGCGCT
>SEQUENCE_5_length_40
TCAAATCCGGCAAAGAGACGATGGACGAGCTGGACGATGC
>SEQUENCE_6_length_40
ACTTTGACCGCCTCTACATACTGATTGAGGAAACCGATAA
```


4. stats.txt

```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assembly/assembly$ more stats.txt
ID      lgth    out     in      long_cov  short1_cov  short1_Ocov  short2_cov  short2_Ocov  long_nb  short1_nb
1       3275    2       2       0.000000  60.430840  59.890687   0.000000   0.000000    0       9926    0
2       31      1       1       0.000000  32.774194  32.774194   0.000000   0.000000    0       125    0
3       31      1       1       0.000000  33.580645  33.580645   0.000000   0.000000    0       127    0
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assembly/assembly$
```

We can see that Velvet has managed to assemble 3 contigs (node). This file is a simple tab-delimited description of the contigs. The column names are pretty much self-explanatory. Note however that contig lengths are given in k-mers. To obtain the length in nucleotides of each node you simply need to add $k - 1$, where k is the hash length used in velvet. The in and out columns correspond to the number of arcs on the 5' and 3' ends of the contig respectively. The coverages in columns short1 cov, short1 Ocov, short2 cov, and short2 Ocov are provided in k-mer coverage.

The relation between k-mer coverage C_k and standard (nucleotide-wise) coverage C is $C_k = C * (L - k + 1)/L$ where k is your hash length, and L your read length.

5. contigs.fa

This contains the contigs as assembled by velvetg in FASTA format. It constitutes the main output file. Note that this does not contain any annotation – Velvet only deals with assembly of DNA. Any scaffolding information is indicated by a long (>10) series of Ns.

Task 6: Count number of unassembled reads in UnusedReads.fa and the number of assembled contigs in contigs.fa. Does it tally with the Log file?

Hint, use the grep -c command

3.4 More denovo assembly statistics

Now that we have a de novo assembly, we can generate some basic statistics for it using a script called fasta_summary.pl. Note that we can use this for any FASTA formatted file. Make sure you are in the ~/genomics_tutorial/strain1/denovo_assembly/assembly directory.

Task 7: Generate summary statistics

```
fasta_summary.pl -i contigs.fa -o assembly_statistics -t contig -r 1
```

```
cd assembly_statistics
```

```
ls
```

```

ubuntu@ip-10-28-24-212:~/genomics_tutorial/strain1/denovo_assembly/assembly$ cd assembly_statistics/
ubuntu@ip-10-28-24-212:~/genomics_tutorial/strain1/denovo_assembly/assembly/assembly_statistics$ ls
histogram_bins.dat      sorted_contigs.fa      sum_reads_vs_read_len.dat      summed_contig_lengths.dat
histogram_bins.dat.png stats.txt               sum_reads_vs_read_len.dat.png  summed_contig_lengths.dat.png
ubuntu@ip-10-28-24-212:~/genomics_tutorial/strain1/denovo_assembly/assembly/assembly_statistics$

```

Note that this directory contains a file called stats.txt, but that this is a different file from that produced by Velvet. View the *.txt files using a text-editor (e.g. gedit, nano) or the more command.

```

Statistics for contig lengths:
  Min contig length:      61
  Max contig length:     3,305
  Mean contig length:    1142.33
  Standard deviation of contig length: 1529.24
  Median contig length:  61
  N50 contig length:     3,305

Statistics for numbers of contigs:
  Number of contigs:      3
  Number of contigs >=1kb: 1
  Number of contigs in N50: 1

Statistics for bases in the contigs:
  Number of bases in all contigs: 3,427
  Number of bases in contigs >=1kb: 3,305
  GC Content of contigs: 43.65 %

Simple Dinucleotide repeats:
  Number of contigs with over 70% dinucleotide repeats: 0.00 % (0 contigs)
  AT: 0.00 % (0 contigs)
  CG: 0.00 % (0 contigs)
  AC: 0.00 % (0 contigs)
  TG: 0.00 % (0 contigs)
  AG: 0.00 % (0 contigs)
  TC: 0.00 % (0 contigs)

```

Note that Velvet was able to assemble 3,305bp in 3 contigs with a GC content slightly lower than that of *E.coli*. Typically with a real dataset you would see many contigs here with a wide variety of lengths.

You can also use gthumb to view various plots which have the extension .png. (If gthumb is not installed, type `sudo apt-get install gthumb`). In this case they are not very informative, but in future projects they can be very valuable when faced with thousands of contigs.

3.5 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. Finally, we will search the open reading frames against the Pfam database of protein families (<http://pfam.sanger.ac.uk>). **NOTE: the BLAST nt/nr databases and the Pfam databases are not installed on the VirtualBox VM, so if you are using VirtualBox, steps that require those databases will fail.**

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, unless the sequences involved are relatively short in which case convergent evolution may be an alternative explanation. 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 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 (<http://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`. Make sure you are back in the `assembly/` directory.

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

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 the first ORF occurs on NODE 1 and is between position 1073 and 1948. Whereas the fourth ORF occurs between positions 606-989 on the reverse strand. This is a relatively short peptide 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_1_length_3275_cov_60.430840_1 [3 - 557]
CDYIAHHFSTVLPPVFCRRTFQSDNTVTAKKQQCFVGNLQGTGHVQLLYRAYMHAITI
TLVRASPRHPMSDTEPCFMTKRSGSNTGRRAMSHPVRLTAEEDQEIRKRAAECGKTVSGF
LRAAALGKKVNSLDDRALKEVMRLGALQKFLIDGKRVGDREYAEVLIATKEYHRALLS
RLMAD
>NODE_1_length_3275_cov_60.430840_2 [2988 - 3305]
RVSERGSGRAPYVSFSPYALLCRSASCPERYISVLFSTSKRVCMLFWSSAASSCSFSHMV
ACSSASSASSFSSSVRLWLFMNPDMLSAVCCCLFIFLFPFCLSSA
>NODE_1_length_3275_cov_60.430840_3 [2904 - 2521] (REVERSE SENSE)
SRKEHVSKRQSTGRSQHRRRFSIGSAPLTSITKIDAEARGGETRQDYKDTRRRFPPEAPSC
ALLFRPCHLPDTCPPFSLREAWRFLIAHAVGISVRCRSFAPSWALCTNPPFSPTAAPYPV
TIVLSPTR
>NODE_1_length_3275_cov_60.430840_4 [2302 - 1175] (REVERSE SENSE)
INRSIKSTNSYSVDKTRSVHIQKKQRIDQMAQNPFKALNINIDKIESALTQNGVTNYSS
NVKNERETHISGYTKGIDFLIKLMPSGGNTTIGRASGQNNNTYFDEIALIIKENCLYSDTK
NFEYTIPKFSDDDRANLFEFLSEEGITITEDNNDLNCXHQYIMTTSYGDRVRAKIYKRG
SIQFQKGKYLQIASWINDFMCSILNMKEVVEQKNKEFNVDIKKETIESELHSKLPKSIDKI
HEDIKKQLSSSLIMKKIDVEMEDYSTYCFALRAIEGFYQILNDVCNPSSSKNLGEYFT
ENKPKYIIREIHQETINGEIAEVLCECYTYWHENRHGLFHMKPGIADTKTINKLESIAII
DTDCQLKDGGVARLKL
>NODE_1_length_3275_cov_60.430840_5 [1184 - 798] (REVERSE SENSE)
VEIMKKDKKYQIEAIKNKDKTLFIVYATDIYSPSEFFSKIESDLKKKSKGDVFFDLIIP
NGGKKDRYVYTSFNGGKFSSYTLNKVTKTDEYNDLSELSASFFKKNFDKINVNLLSKATS
FALKKGIPI

```

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_descriptions` and `-num_alignments` flags tell blastp to only display the top 10 results for each hit, the `-num_threads` that it should use 4 CPU cores and `-show_gis` that it should include general identifier (GI) numbers in the output.

Type (all on one line):

```
blastp -db nr -query contigs.orf.fa -out contigs.orf.blastp -evalue 1e-06 -
num_threads 4 -show_gis -num_descriptions 10 -num_alignments 10
```

This should take a few minutes. Once complete view the contigs.orf.blastp file in your favourite text-editor (e.g. nano, gedit etc).

Task 10: Review the BLAST format

Have a read of <http://cs124.cs.ucdavis.edu/appendices/BlastResults.html>

```
Sequences producing significant alignments:                                Score      E
                                                                    (Bits)      Value

gi|326340056|gb|EGD63862.1| plasmid mobilization [Escherichia co...  372      1e-101
gi|198448640|ref|YP_002221413.1| plasmid mobilization [Salmonell...  372      2e-101
gi|333972753|gb|AEG39558.1| Plasmid mobilization protein [Escher...  370      8e-101
gi|15743562|ref|NP_277065.1| hypothetical protein pCRP3p04 [Citr...  223      1e-56
gi|355469712|gb|AER93362.1| plasmid mobility protein [Escherichi...  220      6e-56
gi|10955263|ref|NP_052604.1| plasmid mobilization [Escherichia c...  204      5e-51
gi|354868508|gb|EHF28925.1| hypothetical protein EUEG_05023 [Esc...  204      7e-51
gi|168239834|ref|ZP_02664892.1| plasmid mobilization [Salmonella...  203      9e-51
gi|332085423|gb|EGI90590.1| ribbon-helix-helix protein, copG fam...  202      3e-50
gi|315614549|gb|EFU95193.1| ribbon-helix-helix , copG family pro...  201      3e-50

>gi|326340056|gb|EGD63862.1| plasmid mobilization [Escherichia coli 0157:H7 str. 1044]
Length=212

Score = 372 bits (956), Expect = 1e-101, Method: Compositional matrix adjust.
Identities = 179/185 (97%), Positives = 180/185 (98%), Gaps = 0/185 (0%)

Query 1   CDYIAHHFSTVLPPVFCRRTFQSDNTVTAKKQQCFVGNSNLQGTGHVQLLYRAYMHAITI 60
          CDYIAHHFSTVLPPVFCRRTFQSDNTVTAKKQQCFVGNSNLQTG DVQLLYRAYMHAITI
Sbjct 28   CDYIAHHFSTVLPPVFCRRTFQSDNTVTAKKQQCFVGNSNLQGTQDVQLLYRAYMHAITI 87

Query 61  TLVRASPRHPMSDTEPCFMTKRSGSNTGRRAMSHPVRLTAEEDQEIRKRAAECGKTVSGF 120
-Maple (5%)
```

We can see that the first ORF has a hit to a plasmid mobilization protein. It seems possible that this single 3kb stretch of DNA sequence which has been assembled could in fact be part of or a whole plasmid.

However, this could simply be a common protein which is shared by many plasmids and/or genomes. Given that this protein is labelled as a 'plasmid mobilization' protein, the chances are that it shares a considerable portion of its sequence with other plasmids. As such the protein sequence is not the best method to search for the source of this sequence. A better way is to compare the nucleotide sequence and use the added variation caused by synonymous and non-coding regions to identify the likely source of this extra DNA.

Task 11: Search contigs against NCBI non-redundant database

The following command executes a nucleotide BLAST search (blastn) of the sequences in the contigs.fa file against the non-redundant database. Again we restrict ourselves to 10 results per hit and an e-value cutoff of 1e-06.

```
blastn -db nt -query contigs.fa -out contigs.fa.blastn -evaluate 1e-06 -  
num_alignments 10 -num_descriptions 10 -show_gis -num_threads 4
```

Opening the results file should yield:

```
BLASTN 2.2.25+

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: All GenBank+EMBL+DDBJ+PDB sequences (but no EST, STS,  
GSS, environmental samples or phase 0, 1 or 2 HTGS sequences)  
15,769,693 sequences; 40,107,367,207 total letters

Query= NODE_1_length_3275_cov_60.430840

Length=3305

Sequences producing significant alignments:

                                Score      E
                                (Bits)    Value
gi|188504043|gb|EU675685.1| Escherichia coli strain 86-24 seroty... 5620    0.0
gi|4589704|dbj|AB011548.2| Escherichia coli O157:H7 str. Sakai p... 5620    0.0
gi|3152962|emb|Y14016.1| Escherichia coli plasmid p4821 mobA gen... 5616    0.0
gi|150419|gb|L05392.1|NT1NMRKAN Plasmid NTP16 complete nucleotid... 5373    0.0
gi|115394176|gb|DQ916413.1| Salmonella typhimurium plasmid pAnkS... 5363    0.0
gi|152570|gb|J01784.1|RFSE10300B Plasmid RFSE1030 replication ori... 1498    0.0
```

The plasmid encoded is in-fact *Escherichia coli* O157:H7 plasmid pOSAK1/pSP70. However, note how similar the alignments are for the first few hits. This is because plasmids are often very similar (particularly near transposase sites). Although BLAST does by default filter out low complexity sequence, in such a small plasmid, it may have difficulty picking out the correct plasmid. In this example we can get a clue by looking at the amount of coverage the plasmid gets from the assembled contigs. We find that only *Escherichia coli* O157:H7 plasmid pOSAK1 has almost full coverage, the remainder (even though they have better scores and e-values) are not the best match from a biological perspective.

Task 12: Find out more about the E.coli pOSAK1 plasmid

Makino, K. et al Complete Nucleotide Sequences of 93-kb and 3.3-kb Plasmids of an Enterohemorrhagic Escherichia coli O157:H7 Derived from Sakai Outbreak . DNA Research 5,(1-9) 1998

Additional checks (not necessary here, but will be for other genomes!):

Task 13: Check that 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), than this might happen. To double check:

```
bl2seq -i contigs.fa -j ~/genomics_tutorial/reference_sequence/Ecoli_UTI89.fna -p
blastn -e 1e-06
```

Here we use bl2seq (part of the BLAST package) 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 sequencing. Running this, you should find that no hits are found with an e-value less than 1e-06. Therefore we can say with some degree of confidence that it is novel plasmid in this strain.

Task 14: Run open reading frames through pfam_scan

Pfam is a database of protein families. They are grouped together using a number of criteria based on their function. For more information read <http://en.wikipedia.org/wiki/Pfam>. Pfam is grouped into several databases depending on the level of curation. Pfam-A is high-quality manual curation and consists of around 12,500 families. Pfam-B is full of automated predictions which may be informative but should not be relied upon without additional evidence. Pfam will also search for signatures of active-sites if you specify the correct flag.

Here we want to search the Pfam database of Hidden Markov Models to see which protein families are contained within this contig. You'll notice that this runs considerably faster than BLAST. Here we search using the contigs.orf.fa file against the Pfam databases in ~/software/PfamScan and output the results to contigs.orf.pfam. We'll use 4 cpu cores for the search and state that we want to search PfamB entries as well as active site residues.

```
pfam_scan.pl -fasta contigs.orf.fa -dir ~/software/PfamScan/ -outfile
contigs.orf.pfam -cpu 4 -pfamB -as
```

```
# <seq id> <alignment start> <alignment end> <envelope start> <envelope end> <hmm acc> <hmm name> <type> <hmm start> <hmm end> <hmm length> <bit scor
> <E-value> <significance> <clan> <predicted_active_site_residues>
NODE_1_length_3275_cov_60.430840_1 96 129 95 131 PF01402.16 RHH_1 Domain 4 37 39 29.8 3.4e-07 1 CL0057
NODE_1_length_3275_cov_60.430840_3 6 126 4 128 PB005118 Pfam-B_5118 Pfam-B 17 137 595 182.6 1.5e-53 NA NA
```

One of the hits is against Pfam-B, but one is against Pfam-A. Let's take a look at Pfam domain PF01402 (RHH_1). Not an illuminating description, but a quick google search brings up the entry in the Pfam database (or just go to <http://pfam.sanger.ac.uk> and enter the ID in the search box).



Family: **RHH_1 (PF01402)**

27 architectures 3978 sequences 2 interactions 1486 species 18 structures

Summary: Ribbon-helix-helix protein, copG family

Pfam includes annotations and additional family information from a range of different sources. These sources can be accessed via the tabs below.

No Wikipedia article Pfam Interpre

This tab holds the annotation information that is stored in the Pfam database. As we move to using Wikipedia as our main source of annotation, the contents of this tab will be gradually replaced by the Wikipedia tab.

Ribbon-helix-helix protein, copG family [Add annotation](#)

The structure of this protein repressor, which is the shortest reported to date and the first isolated from a plasmid, has a homodimeric ribbon-helix-helix arrangement [2]. The helix-turn-helix-like structure is involved in dimerisation and not DNA binding as might have been expected [2].

Literature references

1. Acebo P, Garcia de Lacoba M, Rivas G, Andreu JM, Espinosa M, del Solar G. Proteins 1998;32:248-261.: Structural features of the plasmid pMV158-encoded transcriptional repressor CopG, a protein sharing similarities with both helix-turn-helix and beta-sheet DNA binding proteins. [PUBMED:9714163](#)
2. Gomis-Rth FX, Sol M, Acebo P, Parraga A, Guasch A, Eritja R, Gonzalez A, Espinosa M, del Solar G, Coll M. EMBO J 1998;17:7404-7415.: The structure of plasmid-encoded transcriptional repressor CopG unliganded and bound to its operator. [PUBMED:9857198](#)

Clan

This family is a member of clan [Met repress \(CL0057\)](#), which has a total of **21 members**.

Example structure
PDB entry: 1Q5V Apo-NHR
View a different structure: 1Q5V

Definitely some form of DNA binding protein, but not much information beyond that. Check out some of the other entries in Pfam just to get a feel for the sort of detail you can expect. The other Pfam-B matches do not tell you much that is useful.


3.6 Analysing the results in RAST

By now you should be able to see that analysing results for de novo assembled reads of any sort can be difficult and time-consuming. Bear in mind that we have only been faced with a single contig of 3kb. Quite often you may find yourself dealing with hundreds, if not thousands of contigs. Some will be a few 100kb long. Others may only be 200-300bp. How should we go about analysing these in a more efficient manner? There are a number of options here. For eukaryotes I would suggest looking at MAKER (<http://www.yandell-lab.org/software/maker.html>) to obtain at least some gene predictions (if not gene annotations). For prokaryotes or archaea the situation is somewhat easier and we can use a web-based service known as RAST. This is not the only service, but it is one of the most common.

RAST is a website where you upload the results of your de novo assembly and RAST will attempt to provide annotation in commonly used GFF and Genbank formats. This can be used to load up the annotation in Artemis or Apollo. Alternatively RAST has its own in-built viewer.

Task 15: Log in to RAST

Make sure you are in Firefox within the Amazon instance. Go to <http://rast.nmpdr.org/> Log-in with the details RAST provided you before you started this series of workshops. If you do not have one, you may need to wait several days for your login to be issued by RAST. Please skip ahead and come back to this section.



RAST Rapid Annotation using
Subsystem Technology version 4.0
The NMPDR, SEED-based, prokaryotic genome annotation service.
For more information about The SEED please visit theSEED.org.

Info: You have been logged out.
Info: To monitor RAST's load and view other news and statistics for RAST and the SEED, please visit "[The Daily SEED](#)."

RAST (Rapid Annotation using Subsystem Technology) is a fully-automated service for annotating bacterial and archaeal genomes. It provides as the number of more or less complete bacterial and archaeal genome sequences is constantly rising, the need for high quality automated we provide RAST as a free service to the community. It leverages the data and procedures established within the [SEED framework](#) to provide quality genome sequences AND the analysis of draft genomes. The service normally makes the annotated genome available within 12-24 hours.

Please note that while the SEED environment and SEED data structures (most prominently [FIGfams](#)) are used to compute the automatic annotation of the SEED. Once annotation is completed, genomes can be downloaded in a variety of formats or viewed online. The genome annotation provided to be able to contact you once the computation is finished and in case user intervention is required, we request that users register with email.

If you use our service, please cite:
The RAST Server: Rapid Annotations using Subsystems Technology.
Aziz RK, Bartels D, Best AA, DeJongh M, Disz T, Edwards RA, Formsma K, Gerdes S, Glass EM, Kubal M, Meyer F, Olsen GJ, Olson R, Ost Vonstein V, Wilke A, Zagnitko O.
BMC Genomics, 2008, [[article](#)]

This project has been funded in whole or in part with Federal funds from the National Institute of Allergy and Infectious Diseases, National Institute of Health.

Login
Password

Task 16: Upload the assembled contigs and annotate using RAST

Click on Your jobs->Upload New Job



RAST Rapid Annotation using
Subsystem Technology version 4.0
The NMPDR, SEED-based, prokaryotic genome annotation service.
For more information about The SEED please visit theSEED.org.

[Home](#) [Your Jobs](#)

Upload a Genome

A prokaryotic genome in one or more contigs should be uploaded in either a single [FASTA](#) format file or in a Genbank format file. Our pipeline will use the taxonomy identifier as a handle for the genome. Therefore it and genus, species and strain in the following upload workflow.

Please note, that only if you submit all relevant contigs (i.e. all chromosomes, if more than one, and all plasmids) that comprise the genomic information of your organism of interest in one job. Features like *Metabolic* picture.

Confidentiality information: Data entered into the server will not be used for any purposes or in fact integrated into the main SEED environment, it will remain on this server for 120 days or until deleted by the submitter.

If you use the results of this annotation in your work, please cite:

The RAST Server: Rapid Annotations using Subsystems Technology.

Aziz RK, Bartels D, Best AA, DeJongh M, Disz T, Edwards RA, Formsma K, Gerdes S, Glass EM, Kubal M, Meyer F, Olsen GJ, Olson R, Osterman AL, Overbeek RA, McNeil LK, Paarmann D, Paczian T, Parrello B, Piuze A, Zagnitko O.
BMC Genomics, 2008, [[article](#)]

File formats: You can either use [FASTA](#) or Genbank format.

- If in doubt about FASTA, [this service](#) allows conversion into FASTA format.
- Due to limits on identifier sizes imposed by some of the third-party bioinformatics tools that RAST uses, we limit the size of contig identifiers to 70 characters or fewer.
- If you use Genbank, you have the option of preserving the gene calls in the options block below. By default, genes will be recalled.

Please note: This service is intended for complete or nearly complete prokaryotic genomes. For now we are not able to reliably process sequence data of very small size, like small plasmid, phages or fragments.

File Upload:

Sequences File

Upload the contigs.fa file obtained by the de novo assembly of unmapped reads.

Review genome data

We have analyzed your upload and have computed the following information.

Contig statistics

Statistic	As uploaded	After splitting into scaffolds
Sequence size	3326	3326
Number of contigs	1	1
GC content (%)	43.4	43.4
Shortest contig size	3326	3326
Median sequence size	3326	3326
Mean sequence size	3326.0	3326.0
Longest contig size	3326	3326

Please enter or verify the following information about this organism:

Required information:

Taxonomy ID: (leave blank if NCBI Taxonomy ID unknown)

Find the taxonomy id for your organism by searching for its name in the [NCBI taxonomy browser](#).

Taxonomy string:

Domain: Bacteria Archaea Virus

Genus:

Species:

Strain:

Genetic Code: 11 (Archaea, most Bacteria, most Virii, and some Mitochondria)
 4 (Mycoplasmata, Spiroplasmata, Ureoplasmata, and Fungal Mitochondria)

We know this is an *E.coli* genome so we can enter 562 as the Taxonomy ID and click on 'Look up taxonomy at NCBI'. If you're dealing with a different organism, be sure to change this number. RAST will automatically split any scaffolds (i.e. contigs with bits missing in the middle – denoted by Ns).

Upload a Genome

Complete Upload

By answering the following questions you will help us improve our ability to track problems in processing your genome:

Optional information:

Sequencing Method Sanger Mix of Sanger and Pyrosequencing Pyrosequencing other

Coverage

Number of contigs

Average Read Length (leave blank if unknown)

Please consider the following options for the RAST annotation pipeline:


RAST Annotation Settings:

Select gene caller	<input type="text" value="RAST"/>	<i>Please select which type of gene calling you would like RAST to perform. Note that</i>
Select FIGfam version for this run	<input type="text" value="Release45"/>	<i>Choose the version of FIGfams to be used to process this genome.</i>
Automatically fix errors?	<input checked="" type="checkbox"/> Yes	<i>The automatic annotation process may run into problems, such as gene candidates requires deleting some gene candidates), please check this box.</i>
Fix frameshifts?	<input type="checkbox"/> Yes	<i>If you wish for the pipeline to fix frameshifts, check this option. Otherwise frameshifts</i>
Build metabolic model?	<input checked="" type="checkbox"/> Yes	<i>If you wish RAST to build a metabolic model for this genome, check this option.</i>
Backfill gaps?	<input checked="" type="checkbox"/> Yes	<i>If you wish for the pipeline to blast large gaps for missing genes, check this option</i>
Turn on debug?	<input type="checkbox"/> Yes	<i>If you wish debug statements to be printed for this job, check this box.</i>
Set verbose level	<input type="text" value="0"/>	<i>Set this to the verbosity level of choice for error messages.</i>
Disable replication	<input type="checkbox"/> Yes	<i>Even if this job is identical to a previous job, run it from scratch.</i>

Replicate the settings above and click on 'Finish the upload'.

Your job may take several hours to run. In the meantime, proceed to the next workshop and come back to this later.

Once complete, RAST should email you a message. You can then view the results or download them in standardized formats (e.g. GFF3, Genbank, EMBL etc).



RAST Rapid Annotation using Subsystem Technology version 4.0

The NMPDR, SEED-based, prokaryotic genome annotation service.
For more information about The SEED please visit theSEED.org.

[Home](#) | [Your Jobs](#) | [Manage Job #41704](#)

Job Details #41704

» [Browse annotated genome in SEED Viewer](#)

» Available downloads for this job:

» [Share this genome with selected users](#)

» [Back to the Jobs Overview](#)

✔ Genome Upload has been successfully completed.

Genome ID - Name:	562.698 - Escherichia coli
Job:	#41704
User:	khp204
Date:	Tue Jan 31 08:03:05 2012
Sequencing method:	other
Coverage:	gt8
Number of contigs:	unknown
Read length:	
Genetic code:	11
Include into SEED:	no
Preserve gene calls:	no
Automatically fix errors:	yes
Fix frameshifts:	no
Backfill gaps:	yes

✔ Rapid Propagation has been successfully completed.

✔ Quality Check has been successfully completed.

For detailed explanations of the terms used in our quality report, please refer to [our wiki](#).

Number of features:	4
Number of warnings:	0
Number of fatal problems:	0

✔ Quality Revision has been successfully completed.

If you download the results in GFF3 format and open in a text-editor you can see the annotations. Again they are not particularly enlightening as such, but we can see that this plasmid appears to consist of 4 proteins.

```
##gff-version 3
NODE_1_length_3306_cov_120.625832    FIG    CDS    1162   1356   .    -    1    ID=fig|562.698.peg.1;Name=probable RNAI modulator protein
NODE_1_length_3306_cov_120.625832    FIG    CDS    1313   1951   .    +    2    ID=fig|562.698.peg.2;Name=Plasmid mobilization protein A
NODE_1_length_3306_cov_120.625832    FIG    CDS    2186   2566   .    -    2    ID=fig|562.698.peg.3;Name=14.5 kDa protein
NODE_1_length_3306_cov_120.625832    FIG    CDS    2563   3204   .    -    1    ID=fig|562.698.peg.4;Name=hypothetical protein
```

How does this compare to the original paper which describes this plasmid? We can also look at other features of RAST such as the SEED Viewer. Here we can see each predicted gene along with annotation and further information as to how the evidence was obtained if you click on the Feature ID.

Browse Genome: [Escherichia coli \(562.698\)](#)

Location Focus Upload List

contig	NODE_1_length_3306_cov_120.625832 (3,326 bp)
start base	0
window	16,000 bp
Color features	by focus

<== draw ==>

export table clear all filters

display items per page
displaying 1 - 4 of 4

Feature ID	Type	Contig	Start	Stop	Length (bp)	Function	Subsystems	Region
fig 562.698.peg.1	CDS	NODE_1_length_3306_cov_120.625832	1356	1162	195	probable RNAI modulator protein	- none -	show
fig 562.698.peg.2	CDS	NODE_1_length_3306_cov_120.625832	1313	1951	639	Plasmid mobilization protein A	- none -	show
fig 562.698.peg.3	CDS	NODE_1_length_3306_cov_120.625832	2566	2186	381	14.5 kDa protein	- none -	show
fig 562.698.peg.4	CDS	NODE_1_length_3306_cov_120.625832	3204	2563	642	hypothetical protein	- none -	show

displaying 1 - 4 of 4

Browse the rest of the RAST server and get a feel for the possibilities the platform may offer you.

When you're ready, move on to (or back to) the de novo assembly part of the workshop!

2014 Workshop on Genomics

Part 4 Short read genomics: De-novo assembly

Instructors:

- Konrad Paszkiewicz k.h.paszkiewicz@exeter.ac.uk

Objectives:

By the end of the workshop you will be expected to be able to:

- Perform QC and adaptor-trim Illumina reads
- Assemble these reads de novo using Velvet and VelvetOptimiser
- Generate summary statistics for the assembly
- Identify open reading frames within the assembly
- Search for matches within the Swissprot database via BLAST and against the Pfam database
- Visualize species distribution of potential matches

4.1 Introduction

In this section of the workshop we will continue the analysis of a strain of *E.coli* which is involved in urinary tract infections. 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.

Although it is not strictly necessary for this particular example (because there appears to be very little new material), we will run through the procedure here so that you are able to do this in the next section.

To do this we will use a program called VelvetOptimiser 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.

4.2 Task 1 Preparation

Ensure you are in the `~/genomics_tutorial/strain1` directory. We will move the `denovo_analysis` directory into the `remapping_to_reference` directory so we don't end up confusing the current de novo assembly-from-scratch, with the previous one using unmapped reads.

```
mv denovo_assembly/ remapping_to_reference/
```

Let's create a new directory to store this denovo-assembly-from-scratch. We'll call it `complete_denovo_assembly`

```
mkdir complete_denovo_assembly/
```

Listing the contents of the `~/genomics_tutorial/strain1` directory you should now have:

```
ubuntu@ip-10-28-24-176:~/genomics_tutorial/strain1$ ls
complete_denovo_assembly  illumina_reads  remapping_to_reference
ubuntu@ip-10-28-24-176:~/genomics_tutorial/strain1$
```

```
cd complete_denovo_assembly/
```

Assembly theory

Velvet is an assembly algorithm which requires a number of parameters. The three most important are k-mer length, expected coverage and coverage cutoff.

- 1. K-mer length.** Rather than store all reads individually which would be unfeasible for Illumina type datasets, Velvet converts each read to a series of k-mers and stores 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, but generally results in higher memory consumption and shorter contigs).
- 2. Expected coverage.** This is rough indication of how much sequence information there is relative to the genome size. It is used to try to resolve repetitive regions if paired-end information is missing or insufficient.
- 3. Coverage cutoff.** Relates to the cutoff Velvet should use to decide that a particular k-mer is noise (i.e. error-laden or low-level contamination) and should be excluded from the assembly.

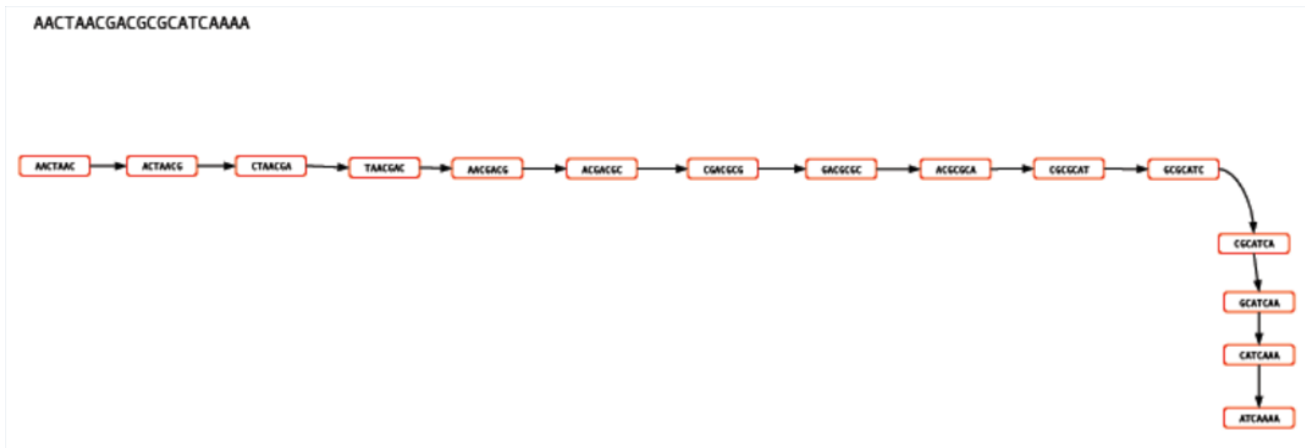
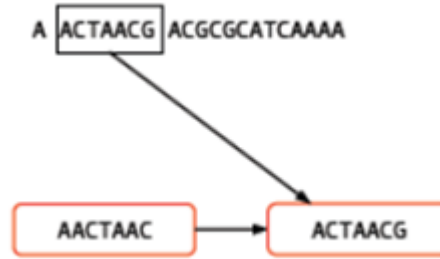
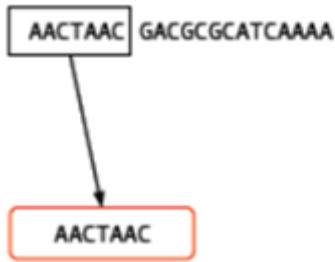
Generally it is necessary to try a large combination of parameters to ensure that you obtain the 'best' possible assembly for a given dataset. What 'best' actually means in the context of genome assembly is ill-defined. For a genomic assembly you want to try to obtain the lowest number of contigs, with 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.

Description of k-mers:

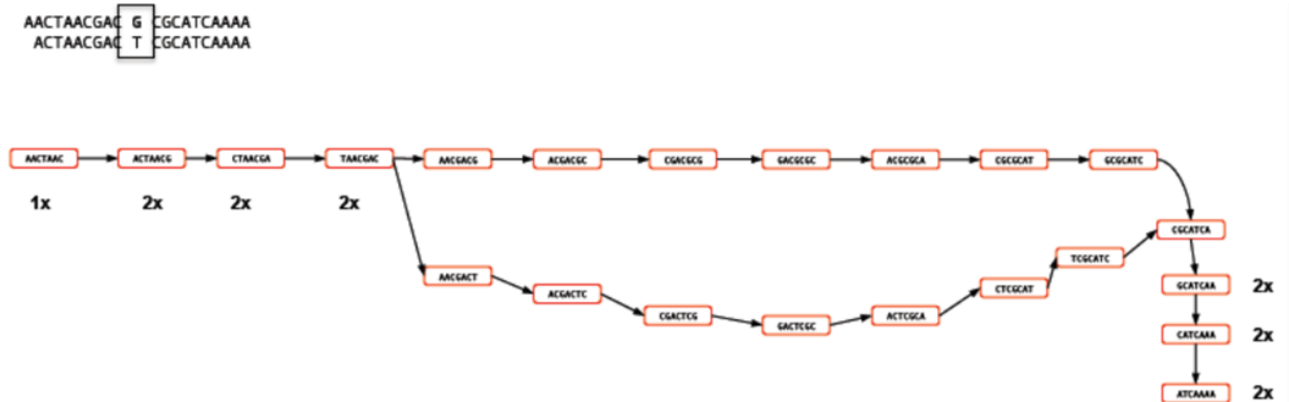
What are they? Let's say you have a single read:

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:



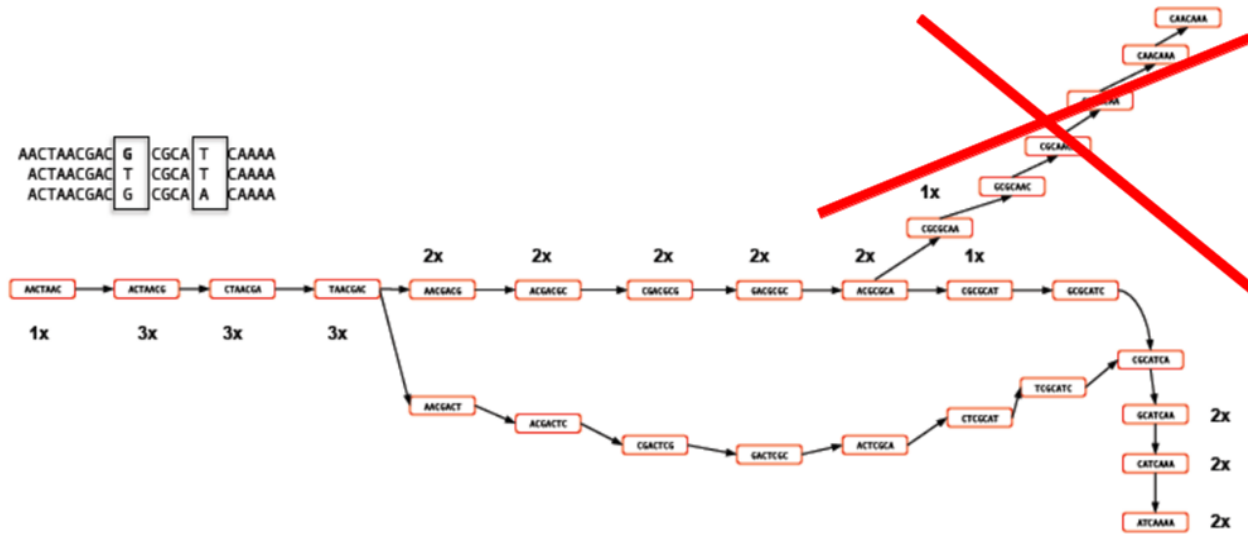
You may well ask, “So what? How does that help”? For a single read, it really doesn't help. However let's say that you have another read which is identical except for a single base:



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.



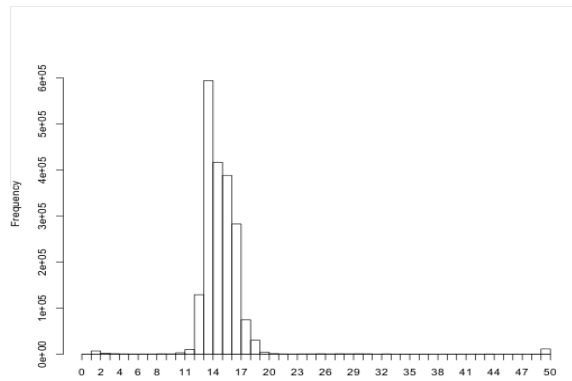
The job of any k-mer based assembler is to find a path through the k-mer graph which correctly represents the genome sequence.

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/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 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 exclude those contigs which have coverage less than this – they are likely to be errors.

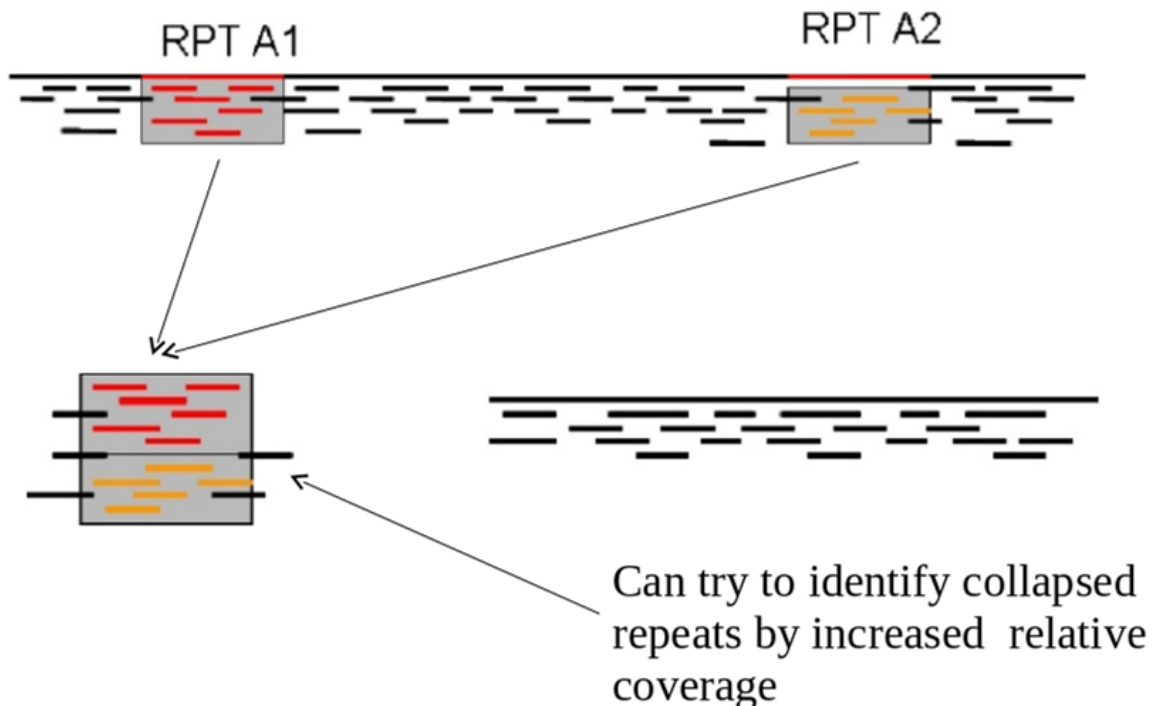


Description of expected coverage:

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.



Task 2: Using VelvetOptimiser

VelvetOptimiser cycles Velvet through various k-mer, expected coverage and coverage cutoff values to see which one gives the 'best' result. Best is defined by the user as either N50 length, length of longest contig, total number of bases in contigs etc. It is distributed with Velvet and is a 'contributed' script (i.e. not written by the original author of Velvet, but thought to be so useful as to be worth bundling with the Velvet release).

Type:

VelvetOptimiser.pl

```
*****
VelvetOptimiser.pl Version 2.2.4
*****
Number of CPUs available: 2
Current free RAM: 6.530GB
Velvet OMP compiler setting: 1
Usage: /usr/local/bin/VelvetOptimiser.pl [options] -f 'velveth input line'
--help          This help.
--version!      Print version to stdout and exit. (default '0').
--v|verbose+    Verbose logging, includes all velvet output in the logfile. (default '0').
--s|hashs=i     The starting (lower) hash value (default '19').
--e|hashe=i     The end (higher) hash value (default '300').
--x|step=i      The step in hash search.. min 2, no odd numbers (default '2').
--f|velvethfiles=s The file section of the velveth command line. (default '0').
--a|amosfile!   Turn on velvet's read tracking and amos file output. (default '0').
--o|velvetgoptions=s Extra velvetg options to pass through. eg. -long_mult_cutoff
--t|threads=i   The maximum number of simultaneous velvet instances to run. (default '1').
--g|genomesize=f The approximate size of the genome to be assembled in megabases.
                 Only used in memory use estimation. If not specified, memory
                 will not occur. If memory use is estimated, the results are s
--k|optFuncKmer=s The optimisation function used for k-mer choice. (default 'n50').
--c|optFuncCov=s The optimisation function used for cov_cutoff optimisation. (default 'n50').
--m|minCovCutoff=f The minimum cov_cutoff to be used. (default '0').
```

You should see the options list for Velvet. Note that there are a number of options which we need to set. The first two are the upper and lower bounds for the k-mer search. The lower bound should never be lower than 19 as memory requirements tend to spiral, and the upper bound should not be longer than the read length (i.e. 40) – otherwise no k-mers can be formed at all!

The next parameter we need to feed is the -f parameter which just specifies whether the reads are paired or unpaired and what format they are in. We can also set -t if we have a large memory machine to do multiple assemblies at the same time to speed things up. If we want to estimate how much memory an assembly will require, we can set the -g flag and indicate in megabases how large we expect the genome to be. We can also set the optimisation functions (i.e. how VelvetOptimiser decides what the 'best' assembly is), both for k-mer size and for coverage cutoff, but we'll leave those alone for now. Feel

free to experiment with these at a later date though!

Before we can run VelvetOptimiser.pl we need to interleave the filtered FASTQ files. Unfortunately, unlike our previous use of velveth and velvetg, VelvetOptimiser cannot work with separate read 1 and read 2 files. It needs to have these files interleaved (i.e. in a single file where the first read in read 1 is followed by the first read in read 2 etc). It needs this additional step because of a software incompatibility (this may be fixed soon).

Type (all on one line):

```
perl ~/software/velvet/shuffleSequences_fastq.pl
~/genomics_tutorial/strain1/illumina_reads/strain1_read1.filtered.fastq
~/genomics_tutorial/strain1/illumina_reads/strain1_read2.filtered.fastq
strain1_interleaved.fastq
```

Now we can run VelvetOptimiser. Let's try it first with the -g flag set to 5Mb so that we can estimate the maximum RAM Velvet will use. Type: (all on one line)

```
VelvetOptimiser.pl -s 19 -e 39 -f '-shortPaired -fastq strain1_interleaved.fastq' -t 1 -g 5
```

```
Maximum number of velvetinstances to run: 1
Will run velvet optimiser with the following parameters:
Velveth parameter string:
    -shortPaired -fastq strain1_interleaved.fastq
Velveth start hash values:      19
Velveth end hash value:        39
Velveth hash step value:       2
Velvetg minimum coverage cutoff to use: 0

Read tracking for final assembly off.
i is 19
i is 21
i is 23
i is 25
i is 27
i is 29
i is 31
i is 33
i is 35
i is 37
i is 39
File: strain1_interleaved.fastq has 6759881 reads of length 40
Total reads: 6.8 million. Avg length: 40.0

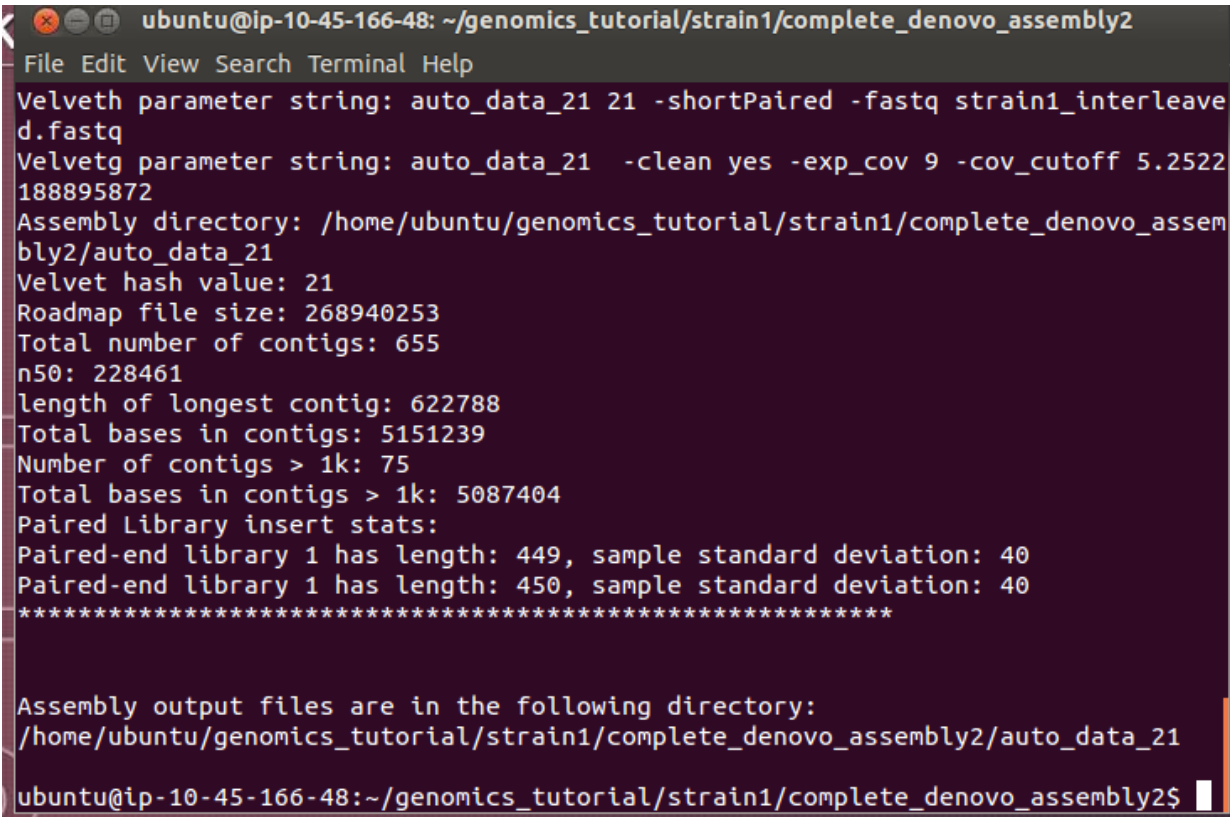
Memory use estimated to be: 1.6GB for 1 threads.

You should have enough memory to complete this job. (Though this estimate is no guarantee..)
```

So the results indicate that we require around 1.6Gb of RAM to do one assembly at a time. We have a few Gb of RAM free so although we could try two threads at a time; let's leave it at 1 thread. If the memory requirements are exceeded you may find that your instance stops responding and you'll need to restart VelvetOptimiser with a higher

starting k-mer (e.g. 35 instead of 19).

```
VelvetOptimiser.pl -s 19 -e 39 -f '-shortPaired -fastq strain1_interleaved.fastq' -t 1
```

A terminal window screenshot showing the output of the VelvetOptimiser.pl script. The terminal title is 'ubuntu@ip-10-45-166-48: ~/genomics_tutorial/strain1/complete_denovo_assembly2'. The output includes: 'Velveth parameter string: auto_data_21 21 -shortPaired -fastq strain1_interleaved.fastq', 'Velvetg parameter string: auto_data_21 -clean yes -exp_cov 9 -cov_cutoff 5.2522188895872', 'Assembly directory: /home/ubuntu/genomics_tutorial/strain1/complete_denovo_assembly2/auto_data_21', 'Velvet hash value: 21', 'Roadmap file size: 268940253', 'Total number of contigs: 655', 'n50: 228461', 'length of longest contig: 622788', 'Total bases in contigs: 5151239', 'Number of contigs > 1k: 75', 'Total bases in contigs > 1k: 5087404', 'Paired Library insert stats: Paired-end library 1 has length: 449, sample standard deviation: 40', 'Paired-end library 1 has length: 450, sample standard deviation: 40', a line of asterisks, and 'Assembly output files are in the following directory: /home/ubuntu/genomics_tutorial/strain1/complete_denovo_assembly2/auto_data_21'. The prompt 'ubuntu@ip-10-45-166-48:~/genomics_tutorial/strain1/complete_denovo_assembly2\$' is visible at the bottom.

```
ubuntu@ip-10-45-166-48: ~/genomics_tutorial/strain1/complete_denovo_assembly2
File Edit View Search Terminal Help
Velveth parameter string: auto_data_21 21 -shortPaired -fastq strain1_interleaved.fastq
Velvetg parameter string: auto_data_21 -clean yes -exp_cov 9 -cov_cutoff 5.2522188895872
Assembly directory: /home/ubuntu/genomics_tutorial/strain1/complete_denovo_assembly2/auto_data_21
Velvet hash value: 21
Roadmap file size: 268940253
Total number of contigs: 655
n50: 228461
length of longest contig: 622788
Total bases in contigs: 5151239
Number of contigs > 1k: 75
Total bases in contigs > 1k: 5087404
Paired Library insert stats:
Paired-end library 1 has length: 449, sample standard deviation: 40
Paired-end library 1 has length: 450, sample standard deviation: 40
*****
Assembly output files are in the following directory:
/home/ubuntu/genomics_tutorial/strain1/complete_denovo_assembly2/auto_data_21
ubuntu@ip-10-45-166-48:~/genomics_tutorial/strain1/complete_denovo_assembly2$
```

Once completed after about 30 minutes you will see something similar to the above. Here we can see that a k-mer length of 21 was found to give the highest N50 value (see http://en.wikipedia.org/wiki/N50_statistic for details). A k-mer coverage cutoff of 5.25 and expected coverage of 9 were the values which gave the largest number of base pairs in contigs larger than 1kb.

In addition to the statistics, we also get a report of the estimated insert size and standard deviation. It is important to check these (they should be around 300-600 bp) to make sure the library prep went well.

This is a reasonable optimised assembly. It has a high N50 (228,461bp) and although there are a total of 655 contigs, most of the genome seems to be assembled into just 75 contigs.

Check out the contents of the auto_data_21 directory. You'll see that the results are in a file called 'contigs.fa'. The files have exactly the same format as those seen in the previous part.

A good check at this point is to map the original reads back to the contigs.fa 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.

4.3 Task 3: Map reads back to assembly

Here we will use BWA again to index the contigs.fa file and remap the reads. This is almost identical to the procedure we followed in Part 3, the only difference is that instead of aligning to the reference genome, we are aligning to the contigs.fa file.

Make sure you are in the following directory.

```
~/genomics_tutorial/strain1/complete_denovo_assembly/auto_data_21
```

Let's start by indexing the contigs.fa file-management. Type:

```
bwa index contigs.fa
```

```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assembly/auto_data_25
$ bwa index contigs.fa
[bwa_index] Pack FASTA... 0.10 sec
[bwa_index] Construct BWT for the packed sequence...
[bwa_index] 3.39 seconds elapse.
[bwa_index] Update BWT... 0.07 sec
[bwa_index] Pack forward-only FASTA... 0.06 sec
[bwa_index] Construct SA from BWT and Occ... 1.16 sec
[main] Version: 0.7.5a-r405
[main] CMD: bwa index contigs.fa
[main] Real time: 5.273 sec; CPU: 4.796 sec
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assembly/auto_data_25
$ ls
contigs.fa      contigs.fa.amb  contigs.fa.bwt  contigs.fa.pac  contigs.fa.sa  Graph  Graph2  Log  PreGraph  Sequences  stats.txt
```

Once complete we can start to align read 1 back to the contigs. Type (all on one line):

```
bwa mem -t 4 contigs.fa  
~/genomics_tutorial/strain1/illumina_reads/strain1_read1.filtered.fastq  
~/genomics_tutorial/strain1/illumina_reads/strain1_read2.filtered.fastq > alignment.sam
```

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

```
samtools view -bS -T contigs.fa alignment.sam > alignment.bam
```


And then we can sort the BAM file:

```
samtools sort alignment.bam alignment.sorted
```

Once completed, we can index the BAM file:

```
samtools index alignment.sorted.bam
```

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

```
samtools flagstat alignment.sorted.bam
```

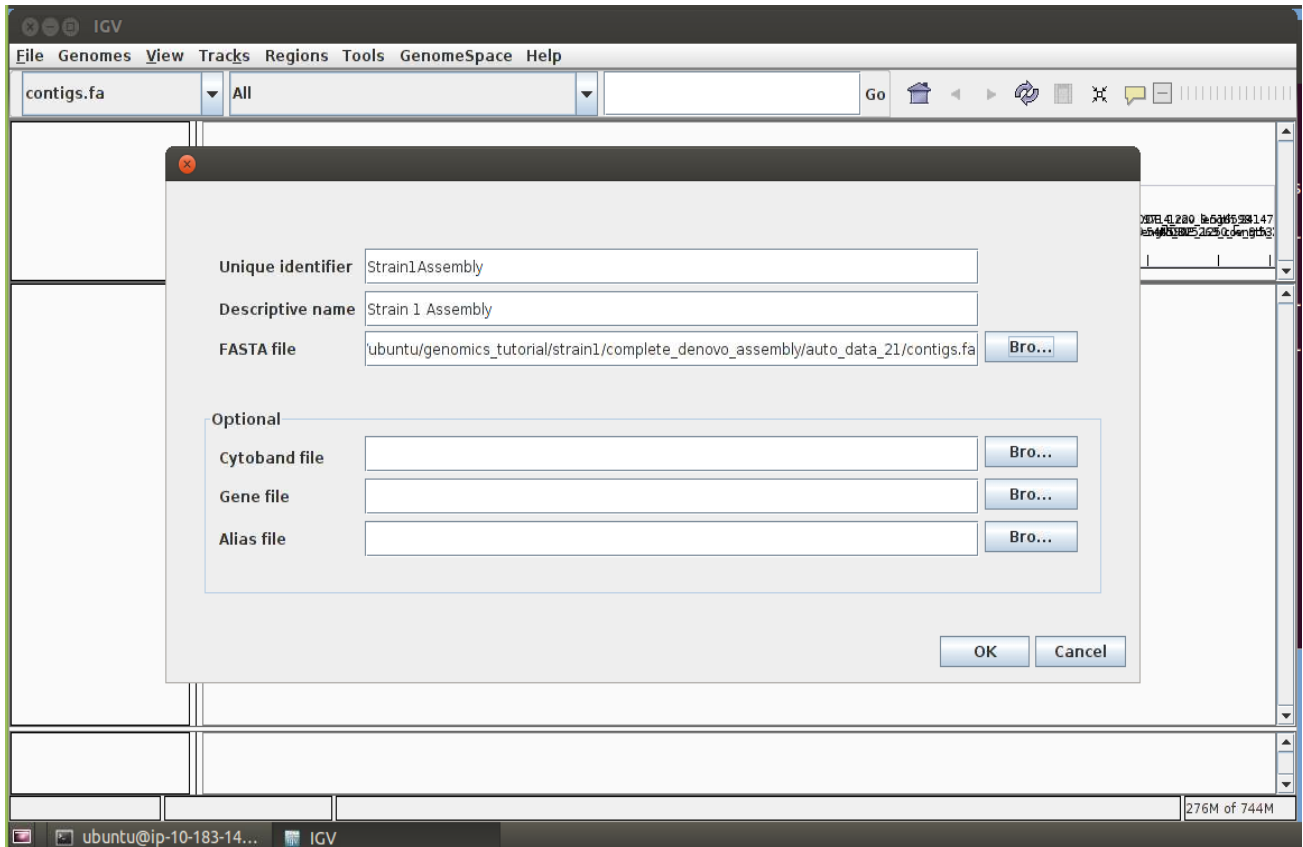
```
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/den
$ samtools flagstat alignment.sorted.bam
6494900 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 duplicates
5710898 + 0 mapped (87.93%:-nan%)
6494900 + 0 paired in sequencing
3247450 + 0 read1
3247450 + 0 read2
5431346 + 0 properly paired (83.62%:-nan%)
5496532 + 0 with itself and mate mapped
214366 + 0 singletons (3.30%:-nan%)
59216 + 0 with mate mapped to a different chr
55943 + 0 with mate mapped to a different chr (mapQ>=5)
```

We can see here that over 12% of reads did not map back to the contigs.

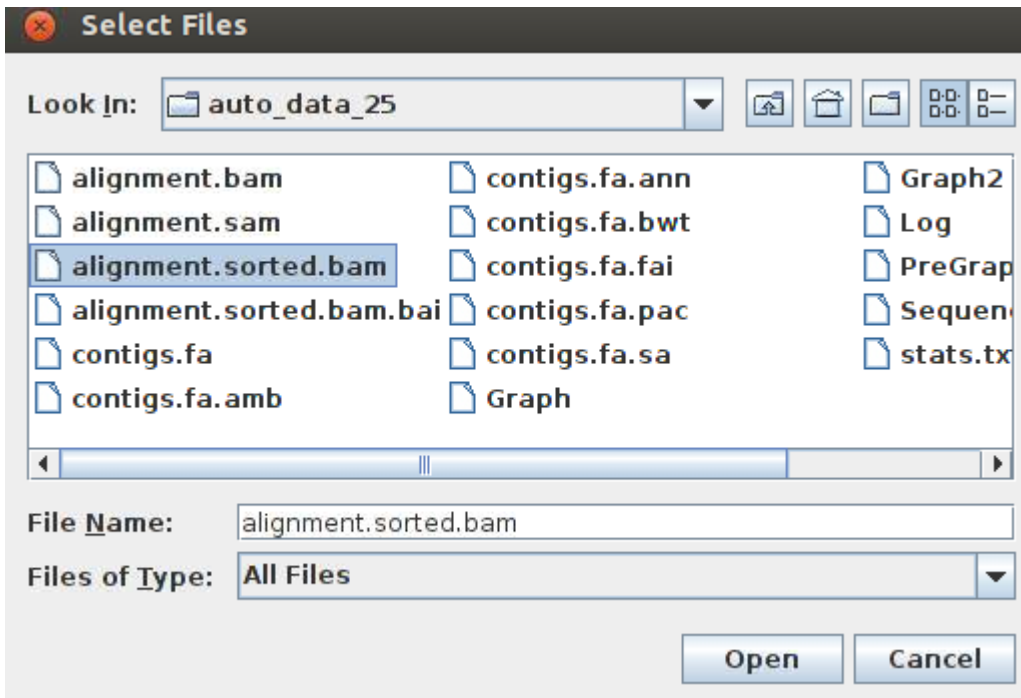
Task 4: View assembly in IGV

As in part 3, load up Firefox (make sure it is within your Amazon instance and not on your desktop). Google for the IGV viewer and load up the 2Gb version of the viewer. (On 32-bit windows machines you will only be able to load the 1.6Gb version). Alternatively, load IGV by typing `igv.sh` in a separate terminal window.

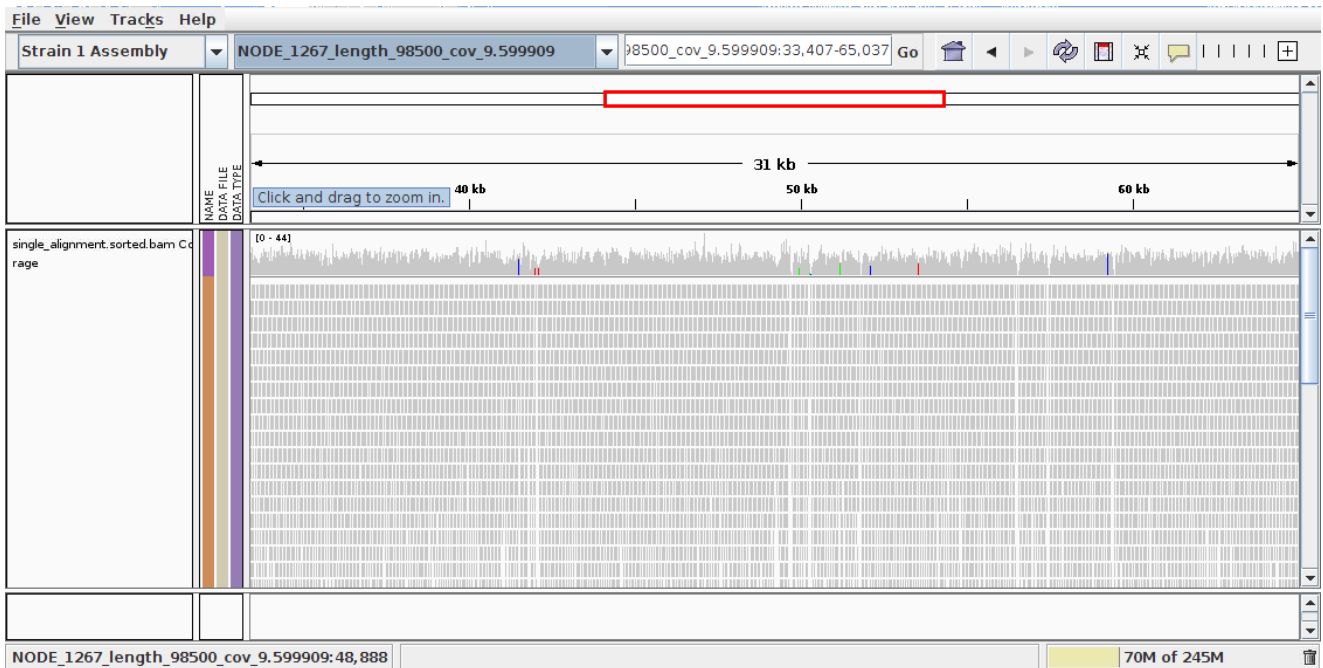
Click Genomes->Create .genome file. We are going to import the contigs we have assembled as the reference. Unlike the reference genome though, we have no annotation available. Make sure you select the contigs.fa file for the complete de novo assembly (not the unmapped reads assembly).



Once saved, click on File->Load From File... select the alignment.sorted.bam file. Again, make sure you load the file in the complete_denovo_assembly directory.



Once loaded, explore some of the contigs in IGV. Can you find examples of some which have no reads mapping back to the assembly? Do you notice anything they tend to have in common?



Annotation of de-novo assembled contigs

We will now annotate the contigs using the open source package Prokka (<http://bioinformatics.net.au/software/prokka.shtml>). This uses BLAST and other tools to generate files suitable for submission to Genbank. This is extremely useful as preparing files for submission to NCBI can be a headache.

Prokka is actually a ‘wrapper’ for many other tools which we have already seen in this workshop (e.g. Blast). Prokka will also perform gene prediction (as well as calling open reading frames) and do some additional annotation using SignalP to predict signal peptides and Infernal to predict conserved rRNA and tRNA. For full details check out the Prokka website listed above.

Task 5: Run Prokka on the assembled contigs

The useful thing about Prokka is that it is very straightforward to execute. A single command will start gene prediction, Blast searches, SignalP and a whole host of other useful annotation tools. This saves us an awful lot of work compared to our previous look at the unmapped reads.

Ensure you are in the auto_data_21 directory containing the contigs.fa file and type (again all on one line):

```
prokka --outdir prokka_annotation --genus Escherichia --species coli --strain UT189 --kingdom Bacteria --gcode 11 --gram neg --cpus 0 contigs.fa
```

This will take around 30 minutes to complete. Once it has finished you can see the files Prokka generated in the prokka_annotation directory:

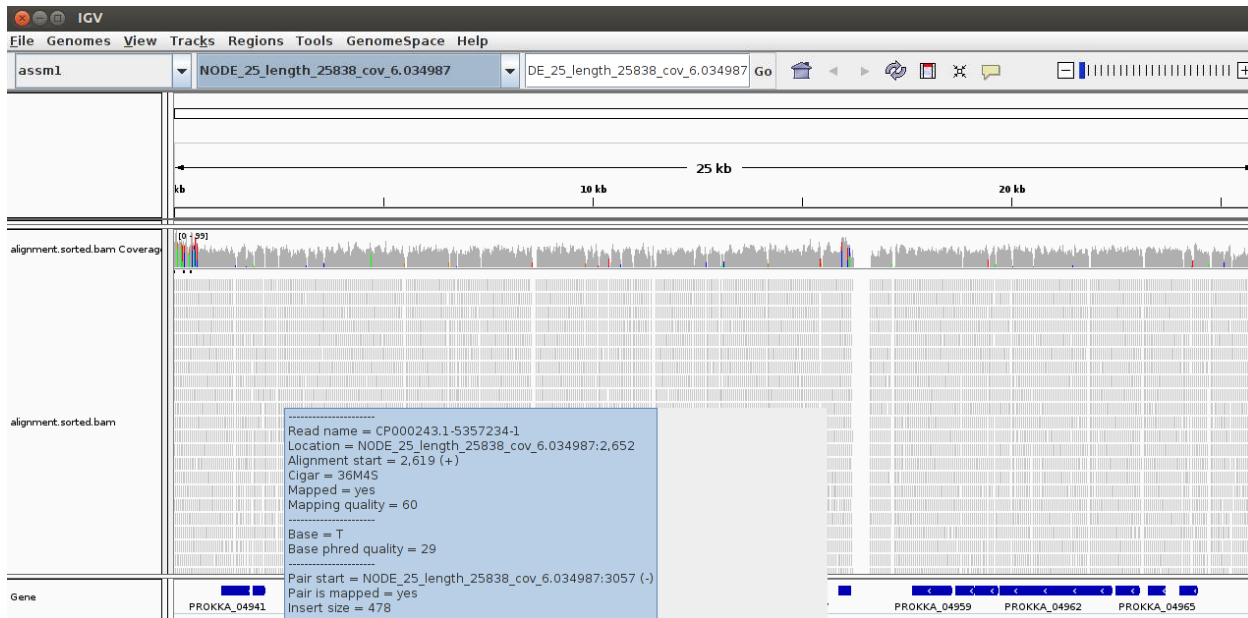
```
PROKKA_12282013.err PROKKA_12282013.fsa PROKKA_12282013.sqn  
PROKKA_12282013.faa PROKKA_12282013.gbk PROKKA_12282013.tbl  
PROKKA_12282013.ffn PROKKA_12282013.gff PROKKA_12282013.txt  
PROKKA_12282013.fna PROKKA_12282013.log  
ubuntu@ip-10-168-53-230:~/genomics_tutorial/strain1/denovo_assem  
/prokka_annotation$ █
```

In summary:

- The .faa file contains the amino acid sequence of each protein found in the assembly
- The .ffn file contains the nucleotide level sequence for each gene
- The .fna file contains the full nucleotide sequence of the contigs

- The .fsa file contains the full nucleotide sequence of the contigs with the full headers including species name and codon usage table
- The annotation files in Genbank and GFF format
- The .txt file with a summary of how many features were found
- The sqn and tbl files are needed for submission to genbank

Task 6: (optional) Load the annotation into IGV (as per Task 4)



Task 6 (Optional): Run the contigs through the RAST server and import the resulting GFF annotation into IGV

Task 7: Obtain open reading frames

As with the unmapped reads, generate open reading frames using getorf

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

Task 8: Run open reading frames through Pfam

As with the unmapped reads we will search the open reading frames against the Pfam HMM database of protein families. Later on we will be able to use these results to identify Pfam domains which are unique to a particular strain.

```
pfam_scan.pl -fasta contigs.orf.fa -dir ~/software/PfamScan/ -outfile  
contigs.orf.pfam -cpu 4 -pfamB -as
```

This will take around 5 hours so it is recommended that you leave this running while continuing with the rest of the activity. If it is still running when you finish your session for today, leave your instance running overnight, but please be sure to turn it off in the morning!

2014 Workshop on Genomics

Part 5 Short read genomics: Comparison of results between different strains

Instructors:

- Konrad Paszkiewicz k.h.paszkiewicz@exeter.ac.uk

Objectives:

In this workshop you will:

By the end of the workshop you will be expected to be able to:

- Run parts 1-3 of the workshop on 2 new datasets
- Use pre-prepared scripts to compare SNPs and Indels between strains
- Generate pseudo-sequences based on synonymous SNPs
- Draw simple trees to illustrate the likely evolutionary relationship between strains
- Compare Pfam matches between strains

Projects!

In the previous sections you have been taken through the steps required to:

1. QC and filter Illumina data
2. Remap Illumina short-read data to a reference sequence
3. View the results in IGV
4. Identify SNPs and Indels in an automated fashion using samtools and bcftools
5. Determine whether SNPs result in synonymous or non-synonymous changes in the corresponding amino acid
6. Extract unmapped reads
7. Assemble unmapped reads and obtain assembly statistics
8. Annotate unmapped reads using Pfam, RAST and/or BLAST
9. Assemble a bacterial genome de-novo using Velvet
10. Obtain assembly statistics
11. Annotate as per the unmapped reads (where computationally feasible).

Now we want you to do the same for another two strains of *E.coli* urinary tract infections which have been isolated. For each strain, make a list of:

1. SNPs, Indels and their effects (from the remapping)
2. Missing genes (from the remapping)
3. Novel plasmids and/or genes (Pfam domains are the easiest way to do this via denovo assembly of unmapped reads)

Once completed, see if you can predict what the phenotype of these bacteria might be. Then proceed to the final part of the tutorial where we will compare the results from all of these strains.

N.B.

It is recommended that you follow the same directory naming convention we have followed here (i.e. one for remapping to the reference, another for assembly of unmapped reads and a final one for the denovo assembly).

You will find the data for the other two strains in the `~/genomics_tutorial/strain2` and `~/genomics_tutorial/strain3`. These tasks may take you several days. However, remember that all of the basic procedures are detailed in the previous sections – only the input FASTQ files will have changed. Feel free to refer to these previous tasks to remind yourself of the commands and parameters. By all means feel free to play around with different parameters if you wish, although remember that the results may differ from those you see here.

Just to give you some guidance: You should find that strain 2 yields many more SNPs than strain 1. Strain 2 should be missing a Beta lactamase and Strain 3 Type III secretory pathway and flagellar proteins. You should also find a different plasmid. You may find that some scripts and programs run more slowly because of these extra differences. Also, if you find the de novo assembly process causes your NX session to end, the chances are that Velvet has caused your instance to run out of memory. If this happens, increase the minimum k-mer size in VelvetOptimiser.

Comparing variants between several samples and a reference genome:

Here we will use a script to compare the variants called in each sample. Ensure you are in the ~/genomics_tutorial directory

First of all, let's make a directory to store the results of the comparison:

```
mkdir snp_comparison/
```

```
cd snp_comparison/
```

```
cp ~/genomics_tutorial/strain1/remapping_to_reference/out.snps.vcf4 ./strain1
```

```
cp ~/genomics_tutorial/strain2/remapping_to_reference/out.snps.vcf4 ./strain2
```

```
cp ~/genomics_tutorial/strain3/remapping_to_reference/out.snps.vcf4 ./strain3
```

Note that the last two copy commands may require modification depending on where you have saved the variant call results.

```
perl /usr/local/scripts/snp_comparator.pl 10  
~/genomics_tutorial/reference_sequence/Ecoli_UTI89.fna  
~/genomics_tutorial/reference_sequence/Ecoli_UTI89.gff strain1 strain2 strain3 >  
snp_comparison.txt
```

Looking at the snp_comparison.txt file (either in a text editor, or in a spreadsheet):

```
## Table of SNP and Indel occurrences between these samples. Note that any comma-separated values (e.g. A,C indicate potential heterozygosity and/or sample heterogeneity)  
Chrom Pos Ref strain1 strain2 strain3 Gene description Status  
CP000243.1 1000078 C C G C UTI89_C1007 alkanesulfonate monooxygenase ,silent tcg -> tcC;  
CP000243.1 1000196 G G A G UTI89_C1007 alkanesulfonate monooxygenase ,non-silent gcc -> gTc;  
CP000243.1 1000403 G G T G UTI89_C1007 alkanesulfonate monooxygenase ,non-silent tca -> tAa;  
CP000243.1 1000438 G G T G UTI89_C1007 alkanesulfonate monooxygenase ,non-silent cac -> caA;
```

Here we can see the chromosome ID, the position in bp, the reference base and the base at each position as well as the gene (if any) the variant occurs in as well as the effect (silent, non-silent or indel).

Obtaining a phylogeny based on synonymous SNPs only:

How are the three strains related on the basis of these variants? We can ask a number of questions, but if we are looking at the long-term evolutionary history of the strains we should only look at synonymous (i.e. silent) mutations as these should not confer a significant selective advantage to any strain. Using the data snp_comparison.txt file we can form 'pseudo-sequences' using the script in /usr/local/scripts/snp2tree_fullsequence.pl. These are concatenated bases consisting of only those positions which are silent across all strains. It is essentially the same as turning each column of each strain in the snp_comparison.txt file into a FASTA entry.


```
perl /usr/local/scripts/snp2tree_fullsequence.pl snp_comparison.txt >
synonymous_tree.fasta
```

Examine the contents of the tree.fasta file. We can then treat this file as an alignment (since each base in each sequence is at the same position on the chromosome) and pass it to a phylogeny program called FastTree. FastTree will take an input alignment and output a Newick formatted tree (http://en.wikipedia.org/wiki/Newick_format).

```
FastTree -nt -gtr < synonymous_tree.fasta > synonymous_tree.newick
```

Now we can visually represent this tree by using the newick2pdf program to create a visual representation of the tree based on the synonymous_tree.newick file.

```
newicktopdf synonymous_tree.newick
```

We can then view the resulting PDF file using the evince program or Adobe Acrobat.

```
evince synonymous_tree.pdf
```

Advanced task (optional):

Copy the snp2tree_fullsequence.pl script to this directory and modify it so that it selects positions containing only non-silent mutations (not indels as these modify the alignment). Generate a new alignment and compare the resulting tree against the silent mutations.

Advanced task (optional):

Try to repeat one of the analyses with the new BreSeq pipeline (<http://barricklab.org/twiki/pub/Lab/ToolsBacterialGenomeResequencing/documentation/introduction.html>). This pipeline automatically aligns your reads to a reference genome and calls SNPs along with generation of pretty HTML files detailing the likely effects of mutations. Compare the SNP calls you get with BreSeq with the SNPs you get using the methods used here. You will need to use the GenBank (gbk) files in the reference_sequence directory.

This is an example of the command you may want to execute.

```
breseq -r /home/ubuntu/reference_sequence/Ecoli_UTI89_mainchr.gbk -r
/home/ubuntu/reference_sequence/Ecoli_UTI89_plasmid.gbk
/home/ubuntu/strain1/illumina_reads/strain1_read1.fastq
/home/ubuntu/strain1/illumina_reads/strain1_read2.fastq
```

As always, keep your results organized – put the breseq results in a separate directory.

Comparing Pfam domains found in each strain:

Here we will use a script to compare the various Pfam domains found in each sample. Ensure you are in the ~/genomics_tutorial directory

First of all, lets make a directory to store the results of the comparison:

```
mkdir pfam_comparison/
```

```
cd pfam_comparison/
```

```
cp
```

```
~/genomics_tutorial/strain1/complete_denovo_assembly/contigs.orf.pfam ./strain1
```

```
cp ~/genomics_tutorial/strain2/
```

```
complete_denovo_assembly/contigs.orf.pfam ./strain2
```

```
cp ~/genomics_tutorial/strain3/
```

```
complete_denovo_assembly/contigs.orf.pfam ./strain3
```

Note that the last two copy commands may require modification depending on where you have saved the Pfam search results.

```
perl /usr/local/scripts/compare_pfam.pl strain1 strain2 strain3 >  
pfam_comparison.txt
```

Examining the pfam_comparison.txt file you should see something similar to:

PF02705	K_trans	strain2.pfam, strain1.pfam,
PF02706	Wzz	strain2.pfam, strain1.pfam,
PF02729	OTCace_N	strain2.pfam, strain1.pfam,
PF02730	AFOR_N	strain2.pfam, strain1.pfam,
PF02733	Dak1	strain2.pfam, strain1.pfam,
PF02734	Dak2	strain2.pfam, strain1.pfam,
PF02737	3HCDH_N	strain2.pfam, strain1.pfam,
PF02738	Ald_Xan_dh_C2	strain2.pfam, strain1.pfam,
PF02739	5_3_exonuc_N	strain2.pfam, strain1.pfam,
PF02742	Fe_dep_repr_C	strain1.pfam,
PF02744	GalP_UDP_tr_C	strain2.pfam, strain1.pfam,

Search the Pfam database to see what some of these differences are (<http://pfam.sanger.ac.uk>)

Concluding remarks:

Well done! If you have reached this far, you deserve a round of applause. You have completed some of the most common tasks in short-read sequencing. You can use the same machine and the same scripts to perform analysis of any short-read dataset! All you need to do is transfer the FASTQ files to the server – you can either do this via Firefox if using the Exeter Sequencing Service (just click on the link in the email), or if you have them on your personal desktop you can use WinSCP (windows) , Fugu or Cyberduck (Mac OSX) or any other SFTP program.

A tutorial can be found at http://www.siteground.com/tutorials/ssh/ssh_winscp.htm