TRAINER'S MANUAL

Introduction to Next Generation Sequencing Hands-on Workshop

Bioplatforms Australia (BPA)
The Commonwealth Scientific and Industrial Research Organisation (CSIRO)

TRAINER'S MANUAL

Licensing

Contents

Workshop Information

The Trainers

Dr. Nandan Deshpande

Postdoctoral Fellow
The University of New South Wales (UNSW), NSW
n.deshpande@unsw.edu.au

Dr. Konsta Duesing

Research Team Leader - Statistics & Bioinformatics
CSIRO Animal, Food and Health Science, NSW
konsta.duesing@csiro.au

Dr. Xi (Sean) Li

Bioinformatics Analyst
Bioinformatics Core, CSIRO Mathematics, Informatics and Statistics, ACT
sean.li@csiro.au

Mr. Sean McWilliam

Bioinformatics Analyst
CSIRO Animal, Food and Health Sciences, QLD
sean.mcwilliam@csiro.au

Dr. Paula Moolhuijzen

Senior Bioinformatics Officer
Centre for Comparative Genomics, Murdoch University, WA
pmoolhuijzen@ccg.murdoch.edu.au

Dr. Sonika Tyagi

Senior Bioinformatics Officer
Australian Genome Research Facility Ltd, The Walter and Eliza Hall Institute, VIC
sonika.tyagi@agrf.org.au

Dr. Nathan S. Watson-Haigh

Research Fellow in Bioinformatics
The Australian Centre for Plant Functional Genomics (ACPFG), SA
nathan.haigh@acpfg.com.au

Table 1:

Providing Feedback

While we endeavour to deliver a workshop with quality content and documentation in a venue conducive to an exciting, well run hands-on workshop with a bunch of knowledgeable and likable trainers, we know there are things we could do better.

Whilst we want to know what didn't quite hit the mark for you, what would be most helpful and least depressing, would be for you to provide ways to improve the workshop. i.e. constructive feedback. After all, if we knew something wasn't going to work, we wouldn't have done it or put it into the workshop in the first place! Remember, we're experts in the field of bioinformatics not experts in the field of biology!

Clearly, we also want to know what we did well! This gives us that "feel good" factor which will see us through those long days and nights in the lead up to such hands-on workshops!

With that in mind, we'll provide three really high tech mechanism through which you can provide anonymous feedback during the workshop:

1. A sheet of paper, from a flip-chart, sporting a "happy" face and a "not so happy" face. Armed with a stack of colourful post-it notes, your mission is to see how many comments you can stick on the "happy" side!

2. Some empty ruled pages at the back of this handout. Use them for your own personal notes or for write specific comments/feedback about the workshop as it progresses.

3. An online post-workshop evaluation survey. We'll ask you to complete this before you leave. If you've used the blank pages at the back of this handout to make feedback notes, you'll be able to provide more specific and helpful feedback with the least amount of brain-drain!

Document Structure

We have provided you with an electronic copy of the workshop's hands-on tutorial documents. We have done this for two reasons: 1) you will have something to take away with you at the end of the workshop, and 2) you can save time (mis)typing commands on the command line by using copy-and-paste.

We advise you to use Acrobat Reader to view the PDF. This is because it properly supports some features we have implemented to ensure that copy-and-paste of commands works as expected. This includes the appropriate copy-and-paste of special characters like tilde and hyphens as well as skipping line numbers for easy copy-and-past of whole code blocks.

While you could fly through the hands-on sessions doing copy-and-paste you will learn more if you take the time, saved from not having to type all those commands, to understand what each command is doing!

The commands to enter at a terminal look something like this:

```
tophat --solexa-quals -g 2 --library-type fr-unstranded -j \
    annotation/Danio_rerio.Zv9.66.spliceSites -o tophat/ZV9_2cells \
    genome/ZV9 data/2cells_1.fastq data/2cells_2.fastq
```

The following styled code is not to be entered at a terminal, it is simply to show you the syntax of the command. You must use your own judgement to substitute in the correct arguments, options, filenames etc

```
tophat [options]* <index_base> <reads_1> <reads_2>
```

The following is an example how of R commands are styled:

```
R --no-save
library(plotrix)
data <- read.table("run_25/stats.txt", header=TRUE)
weighted.hist(data$short1_cov+data$short2_cov, data$lgth, breaks=0:70)
q()
```

The following icons are used in the margin, throughout the documentation to help you navigate around the document more easily:


Important


For reference


Follow these steps


Questions to answer


Warning - STOP and read


Bonus exercise for fast learners


Advanced exercise for super-fast learners

Resources Used

We have provided you with an environment which contains all the tools and data you need for the duration of this workshop. However, we also provide details about the tools and data used by each module at the start of the respective module documentation.

Module: Data Quality

Primary Author(s):
Sonika Tyagi sonika.tyagi@agrf.org.au

Contributor(s):
Nathan S. Watson-Haigh nathan.watson-haigh@awri.com.au

Key Learning Outcomes

After completing this practical the trainee should be able to:

- Assess the overall quality of NGS sequence reads
- Visualise the quality, and other associated matrices, of reads to decide on filters and cutoffs for cleaning up data ready for downstream analysis
- Clean up and pre-process the sequences data for further analysis

Resources You'll be Using

Tools Used

FastQC
http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

FASTX-Toolkit
http://hannonlab.cshl.edu/fastx_toolkit/

Picard
http://picard.sourceforge.net/

Useful Links

FASTQ Encoding
http://en.wikipedia.org/wiki/FASTQ_format#Encoding

Introduction

Going on a blind date with your read set? For a better understanding of the consequences please check the data quality!

For the purpose of this tutorial we are focusing only on Illumina sequencing which uses 'sequence by synthesis' technology in a highly parallel fashion. Although Illumina high throughput sequencing provides highly accurate sequence data, several sequence artifacts, including base calling errors and small insertions/deletions, poor quality reads and primer/adapter contamination are quite common in the high throughput sequencing data. The primary errors are substitution errors. The error rates can vary from 0.5-2.0% with errors mainly rising in frequency at the 3' ends of reads.

One way to investigate sequence data quality is to visualize the quality scores and other metrics in a compact manner to get an idea about the quality of a read data set. Read data sets can be improved by post processing in different ways like trimming off low quality bases, cleaning up any sequencing adapters and removing PCR duplicates. We can also look at other statistics such as, sequence length distribution, base composition, sequence complexity, presence of ambiguous bases etc. to assess the overall quality of the data set.

Highly redundant coverage (>15X) of the genome can be used to correct sequencing errors in the reads before assembly and errors. Various k-mer based error correction methods exist but are beyond the scope of this tutorial.

Quality Value Encoding Schema

In order to use a single character to encode Phred qualities, ASCII characters are used (http://shop.alterlinks.com/ascii-table/ascii-table-us.php). All ASCII characters have a decimal number associated with them but the first 32 characters and non-printable (e.g. backspace, shift, return, escape). Therefore, the first printable ASCII character is number 33, the exclamation mark (!). In Phred+33 encoded quality values the exclamation mark takes the Phred quality score of zero.

Early Solexa (now Illumina) sequencing needed to encode negative quality values. Because ASCII characters < 33 and non-printable, using the Phred+33 encoding was not possible. Therefore, they simply moved the offset from 33 to 64 thus inventing the Phred+64 encoded quality values. In this encoding a Phred quality of zero is denoted by the ASCII number 64 (the @ character). Since Illumina 1.8, quality values are now encoded using Phred+33.

FASTQ does not provide a way to describe what quality encoding is used for the quality values. Therefore, you should find this out from your sequencing provider. Alternatively, you may be able to figure this out by determining what ASCII characters are present in the FASTQ file. E.g the presence of numbers in the quality strings, can only mean the quality values are Phred+33 encoded. However, due to the overlapping nature of

the Phred+33 and Phred+64 encoding schema it is not always possible to identify what encoding is in use. For example, if the only characters seen in the quality string are (@ABCDEFGHI), then it is impossible to know if you have really good Phred+33 encoded qualities or really bad Phred+64 encoded qualities.

For a grapical representation of the different ASCII characters used in the two encoding schema see: http://en.wikipedia.org/wiki/FASTQ_format#Encoding.

Prepare the Environment

To investigate sequence data quality we will demonstrate tools called FastQC and FASTX-Toolkit. FastQC will process and present the reports in a visual manner. Based on the results, the sequence data can be processed using the FASTX-Toolkit. We will use one data set in this practical, which can be found in the QC directory on your desktop.

Open the Terminal and go to the directory where the data are stored:

```
cd ~/QC/
pwd
```

At any time, help can be displayed for FastQC using the following command:

```
fastqc -h
```

Quality Visualisation

We have a file for a good quality and bad quality statistics. FastQC generates results in the form of a zipped and unzipped directory for each input file.

Execute the following command on the two files:

```
fastqc -f fastq bad_example.fastq
fastqc -f fastq good_example.fastq
```

View the FastQC report file of the dab data using a web browser such as firefox.

```
firefox bad_example_fastqc/fastqc_report.html &
```

The report file will have a Basic Statistics table and various graphs and tables for different quality statistics. E.g.:

Table 2: FastQC Basic Statistics table

Filename	bad_example.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	40000
Filtered Sequences	0
Sequence length	100
%GC	48

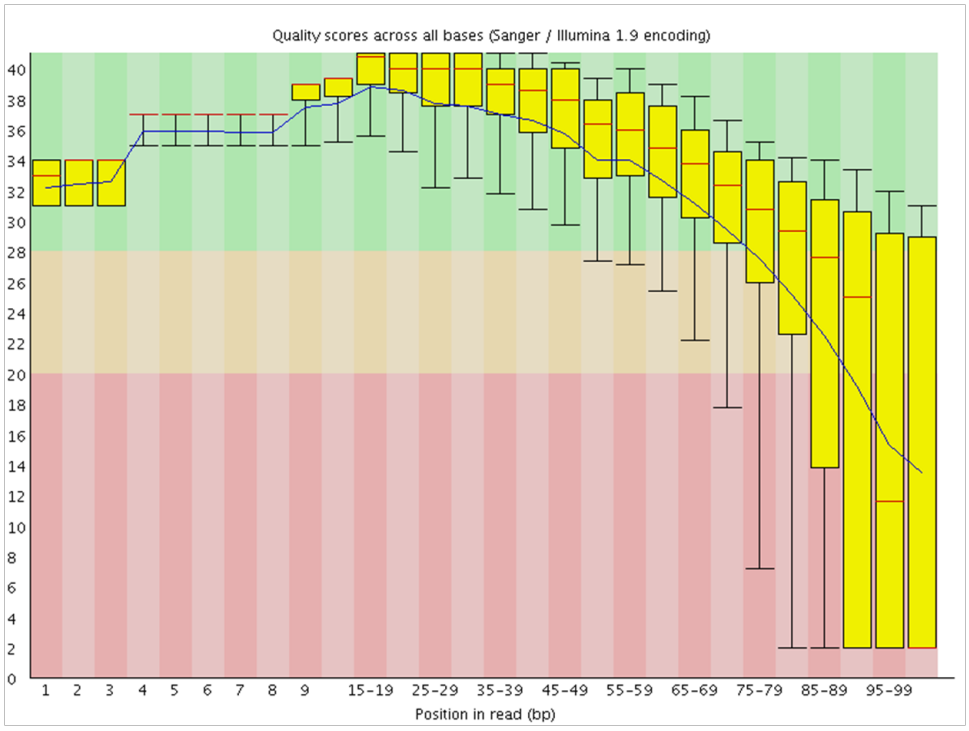

Figure 1: Per base sequence quality plot for `bad_example.fastq`.

A Phred quality score (or Q-score) expresses an error probability. In particular, it serves as a convenient and compact way to communicate very small error probabilities. The probability that base A is wrong ($P(\sim A)$) is expressed by a quality score, $Q(A)$, according to the relationship:

$$Q(A) = -10log10(P(\sim A))$$

The relationship between the quality score and error probability is demonstrated with the following table:

Table 3: Error probabilities associated with various quality (Q) values

Quality score, Q(A)	Error probability, P(∼A)	Accuracy of the base call
10	0.1	90%
20	0.01	99%
30	0.001	99.9%
40	0.0001	99.99%
50	0.00001	99.999%

How many sequences were there in your file? What is the read length? 40,000. read length=100bp

Does the quality score values vary throughout the read length? (hint: look at the 'per base sequence quality plot') Yes. Quality scores are dropping towards the end of the reads.

What is the quality score range you see? 2-40

At around which position do the scores start falling below Q20? Around 80 bp position

How can we trim the reads to filter out the low quality data? By trimming off the bases after a fixed position of the read or by trimming off bases based on the quality score.

Good Quality Data

View the FastQC report files `fastqc_report.html` to see examples of a good quality data and compare the quality plot with that of the `bad_example_fastqc`.

```
1 firefox good_example_fastqc/fastqc_report.html &
```

Sequencing errors can complicate the downstream analysis, which normally requires that reads be aligned to each other (for genome assembly) or to a reference genome (for detection of mutations). Sequence reads containing errors may lead to ambiguous paths in the assembly or improper gaps. In variant analysis projects sequence reads are aligned against the reference genome. The errors in the reads may lead to more mismatches than expected from mutations alone. But if these errors can be removed or corrected, the read alignments and hence the variant detection will improve. The assemblies will also improve

after pre-processing the reads with errors.

Read Trimming

Read trimming can be done in a variety of different ways. Choose a method which best suits your data. Here we are giving examples of fixed-length trimming and quality-based trimming.

Fixed Length Trimming

Low quality read ends can be trimmed using a fixed-length trimming. We will use the `fastx_trimmer` from the FASTX-Toolkit. Usage message to find out various options you can use with this tool. Type `fastx_trimmer -h` at anytime to display help.

We will no do fixed-length trimming of the `bad_example.fastq` file using the following command.

```
cd ~/QC
fastx_trimmer -h
fastx_trimmer -Q 33 -f 1 -l 80 -i bad_example.fastq -o \
    bad_example_trimmed01.fastq
```

We used the following options in the command above:

`-Q 33` Indicates the input quality scores are Phred+33 encoded

`-f` First base to be retained in the output

`-l` Last base to be retained in the output

`-i` Input FASTQ file name

`-o` Output file name

Run FastQC on the trimmed file and visualise the quality scores of the trimmed file.

```
fastqc -f fastq bad_example_trimmed01.fastq
firefox bad_example_trimmed01_fastqc/fastqc_report.html &
```

The output should look like:

Table 4: FastQC Basic Statistics table

Filename	bad_example_trimmed01.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	40000
Filtered Sequences	0
Sequence length	80
%GC	48

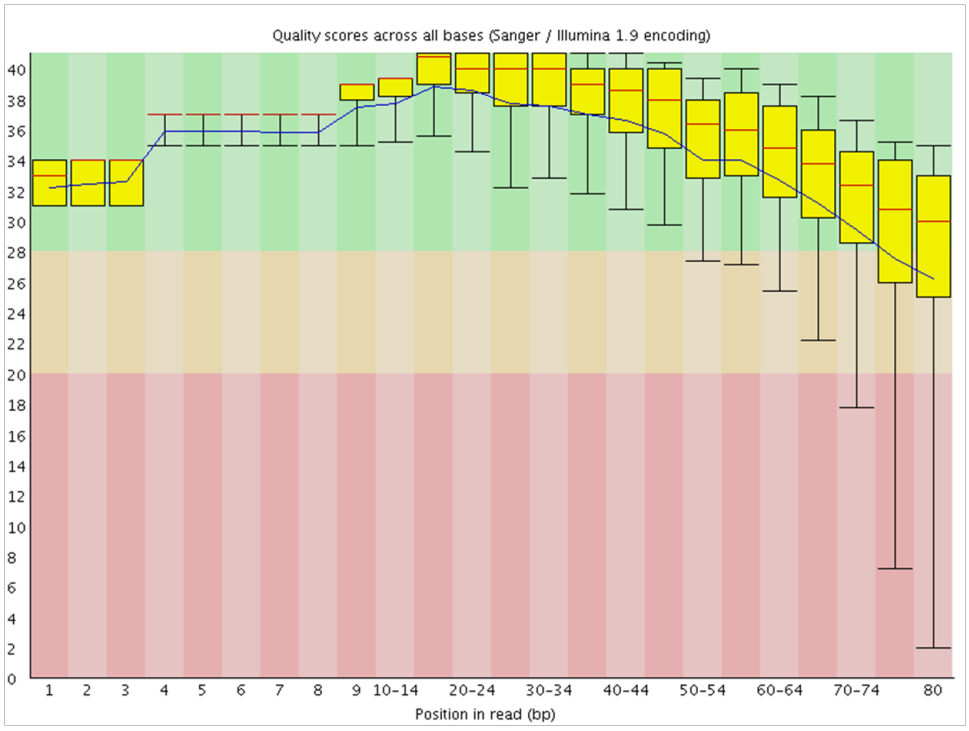

Figure 2: Per base sequence quality plot for the fixed-length trimmed `bad_example.fastq` reads.

What values would you use for `-f` if you wanted to trim off 10 bases at the 5' end of the reads? `-f 11`

Quality Based Trimming

Base call quality scores can also be used to dynamically determine the trim points for each read. A quality score threshold and minimum read length following trimming can be used to remove low quality data.

Run the following command to quality trim your data:

```
cd ~/QC
fastq_quality_trimmer -h
fastq_quality_trimmer -Q 33 -t 20 -l 50 -i bad_example.fastq -o \
    bad_example_quality_trimmed.fastq
```

Run FastQC on the quality trimmed file and visualise the quality scores.

```
fastqc -f fastq bad_example_quality_trimmed.fastq
firefox bad_example_quality_trimmed_fastqc/fastqc_report.html &
```

The output should look like:

Table 5: FastQC Basic Statistics table

Filename	bad_example_quality_trimmed.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	38976
Filtered Sequences	0
Sequence length	50-100
%GC	48

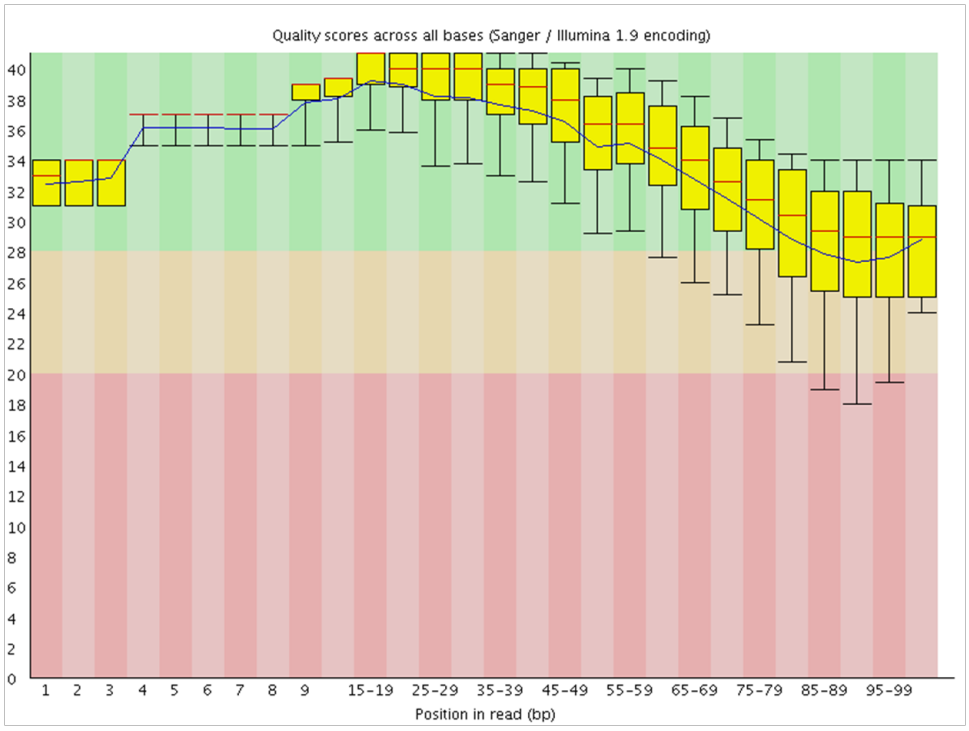

Figure 3: Per base sequence quality plot for the quality-trimmed `bad_example.fastq` reads.

How did the quality score range change with two types of trimming? Some poor quality bases (Q <20) are still present at the 3' end of the fixed-length trimmed reads. It also removes bases that are good quality.

Quality-based trimming retains the 3' ends of reads which have good quality scores.

Did the number of total reads change after two types of trimming? Quality trimming discarded >1000 reads. However, We retain a lot of maximal length reads which have good quality all the way to the ends.

What reads lengths were obtained after quality based trimming? 50-100

Reads <50 bp, following quality trimming, were discarded.

Did you observe adapter sequences in the data? No. (Hint: look at the overrepresented sequences.

Adapter Clipping

Sometimes sequence reads may end up getting the leftover of adapters and primers used in the sequencing process. It's good practice to screen your data for these possible contamination for more sensitive alignment and assembly based analysis.

This is particularly important when read lengths can be longer than the molecules being sequenced. For example when sequencing miRNAs.

Various QC tools are available to screen and/or clip these adapter/primer sequences from your data. (e.g. FastQC, FASTX-Toolkit, cutadapt).

Here we are demonstrating `fastx_clipper` to trim a given adapter sequence.

```
1 cd ~/QC
2 fastx_clipper -h
3 fastx_clipper -v -Q 33 -l 20 -M 15 -a \
    GATCGGAAGAGCGGTTCAGCAGGAATGCCGAG -i bad_example.fastq -o \
    bad_example_clipped.fastq
```

An alternative tool, not installed on this system, for adapter clipping is `fastq-mcf`. A list of adapters is provided in a text file. For more information, see FastqMcf at http://code.google.com/p/ea-utils/wiki/FastqMcf.

Removing Duplicates

Duplicate reads are the ones having the same start and end coordinates. This may be the result of technical duplication (too many PCR cycles), or over-sequencing (very high fold coverage). It is very important to put the duplication level in context of your experiment. For example, duplication level in targeted or re-sequencing projects may mean something different in RNA-seq experiments. In RNA-seq experiments oversequencing is usually necessary when detecting low abundance transcripts.

The duplication level computed by FastQC is based on sequence identity at the end of reads. Another tool, Picard, determines duplicates based on identical start and end positions in SAM/BAM alignment files.

We will not cover Picard but provide the following for your information.

Picard is a suite of tools for performing many common tasks with SAM/BAM format files. For more information see the Picard website and information about the various command-line tools available:

http://picard.sourceforge.net/command-line-overview.shtml

Picard 1.69 is installed on this system in `/usr/share/java/picard-1.69/`

One of the Picard tools (MarkDuplicates) can be used to analyse and remove duplicates from the raw sequence data. The input for Picard is a sorted alignment file in BAM format. Short read aligners such as, bowtie, BWA and tophat can be used to align FASTQ files against a reference genome to generate SAM/BAM alignment format.

Interested users can use the following general command to run the MarkDuplicates tool at their leisure. You only need to provide a BAM file for the INPUT argument (not provided):

```
cd ~/QC
java -jar /usr/share/java/picard/MarkDuplicates.jar \
    INPUT=<alignment_file.bam> VALIDATION_STRINGENCY=LENIENT \
    OUTPUT=alignment_file.dup METRICS_FILE=alignment_file.matric \
    ASSUME_SORTED=true REMOVE_DUPLICATES=true
```

Module: Read Alignment

Primary Author(s):
Myrto Kostadima kostadim@ebi.ac.uk

Contributor(s):
Xi Li sean.li@csiro.au

Key Learning Outcomes

After completing this practical the trainee should be able to:

- Perform the simple NGS data alignment task against one interested reference data
- Interpret and manipulate the mapping output using SAMtools
- Visualise the alignment via a standard genome browser, e.g. IGV browser

Resources You'll be Using

Tools Used

Bowtie
http://bowtie-bio.sourceforge.net/index.shtml

Bowtie 2
http://bowtie-bio.sourceforge.net/bowtie2/index.shtml

Samtools
http://picard.sourceforge.net/

BEDTools
http://code.google.com/p/bedtools/

UCSC tools
http://hgdownload.cse.ucsc.edu/admin/exe/

IGV genome browser
http://www.broadinstitute.org/igv/

Useful Links

SAM Specification
http://samtools.sourceforge.net/SAM1.pdf

Explain SAM Flags
http://picard.sourceforge.net/explain-flags.html

Sources of Data

http://www.ebi.ac.uk/arrayexpress/experiments/E-GEOD-11431

Introduction

The goal of this hands-on session is to perform an unspliced alignment for a small subset of raw reads. We will align raw sequencing data to the mouse genome using Bowtie and then we will manipulate the SAM output in order to visualize the alignment on the IGV browser.

Prepare the Environment

We will use one data set in this practical, which can be found in the `ChIP-seq` directory on your desktop.

Open the Terminal.

First, go to the right folder, where the data are stored.

```
1 cd ~/ChIP-seq
```

The `.fastq` file that we will align is called `Oct4.fastq`. This file is based on Oct4 ChIP-seq data published by Chen *et al.* (2008). For the sake of time, we will align these reads to a single mouse chromosome.

Alignment

You already know that there are a number of competing tools for short read alignment, each with its own set of strengths, weaknesses, and caveats. Here we will try Bowtie, a widely used ultrafast, memory efficient short read aligner.

Bowtie has a number of parameters in order to perform the alignment. To view them all type

```
bowtie --help
```

Bowtie uses indexed genome for the alignment in order to keep its memory footprint small. Because of time constraints we will build the index only for one chromosome of the mouse genome. For this we need the chromosome sequence in FASTA format. This is stored in a file named `mm9`, under the subdirectory `bowtie_index`.

The indexed chromosome is generated using the command:

```
bowtie-build bowtie_index/mm9.fa bowtie_index/mm9
```

This command will output 6 files that constitute the index. These files that have the prefix `mm9` are stored in the `bowtie_index` subdirectory. To view if they files have been successfully created type:

```
1 ls -l bowtie_index
```

Now that the genome is indexed we can move on to the actual alignment. The first argument for `bowtie` is the basename of the index for the genome to be searched; in our case this is `mm9`. We also want to make sure that the output is in SAM format using the `-S` parameter. The last argument is the name of the FASTQ file.

Align the Oct4 reads using Bowtie:

```
1 bowtie bowtie_index/mm9 -S Oct4.fastq > Oct4.sam
```

The above command outputs the alignment in SAM format and stores them in the file `Oct4.sam`.

In general before you run Bowtie, you have to know what quality encoding your FASTQ files are in. The available FASTQ encodings for bowtie are:

`--phred33-quals`	Input qualities are Phred+33 (default).
`--phred64-quals`	Input qualities are Phred+64 (same as `--solexa1.3-quals`).
`--solexa-quals`	Input qualities are from GA Pipeline ver. < 1.3.
`--solexa1.3-quals`	Input qualities are from GA Pipeline ver. ≥ 1.3.
`--integer-quals`	Qualities are given as space-separated integers (not ASCII).

The FASTQ files we are working with are Sanger encoded (Phred+33), which is the default for Bowtie.

Bowtie will take 2-3 minutes to align the file. This is fast compared to other aligners which sacrifice some speed to obtain higher sensitivity.

Look at the top 10 lines of the SAM file by typing:

```
1 head -n 10 Oct4.sam
```

Can you distinguish between the header of the SAM format and the actual alignments? The header line starts with the letter '@', i.e.:

```
@HD   VN:1.0      SO:unsorted
@SQ   SN:chr1     LN:197195432
@PG   ID:Bowtie   VN:0.12.8      CL:"bowtie bowtie_index/mm9 -S Oct4.fastq"
```

While, the actual alignments start with read id, i.e.:

```
SRR002012.45  0   chr1  etc
SRR002012.48  16  chr1  etc
```

What kind of information does the header provide?

- @HD: Header line; VN: Format version; SO: the sort order of alignments.
- @SQ: Reference sequence information; SN: reference sequence name; LN: reference sequence length.
- @PG: Read group information; ID: Read group identifier; VN: Program version; CL: the command line that produces the alignment.

To which chromosome are the reads mapped? Chromosome 1.

Manipulate SAM output

SAM files are rather big and when dealing with a high volume of NGS data, storage space can become an issue. As we have already seen, we can convert SAM to BAM files (their binary equivalent that are not human readable) that occupy much less space.

Convert SAM to BAM using `samtools view` and store the output in the file `Oct4.bam`. You have to instruct `samtools view` that the input is in SAM format (`-S`), the output should be in BAM format (`-b`) and that you want the output to be stored in the file specified by the `-o` option:

```
samtools view -bSo Oct4.bam Oct4.sam
```

Compute summary stats for the Flag values associated with the alignments using:

```
samtools flagstat Oct4.bam
```

Visualize alignments in IGV

IGV is a stand-alone genome browser. Please check their website (http://www.broadinstitute.org/igv/) for all the formats that IGV can display. For our visualization purposes we will use the BAM and bigWig formats.

When uploading a BAM file into the genome browser, the browser will look for the index of the BAM file in the same folder where the BAM files is. The index file should have the same name as the BAM file and the suffix `.bai`. Finally, to create the index of a BAM file you need to make sure that the file is sorted according to chromosomal coordinates.

Sort alignments according to chromosomal position and store the result in the file with the prefix `Oct4.sorted`:

```
samtools sort Oct4.bam Oct4.sorted
```

Index the sorted file.

```
samtools index Oct4.sorted.bam
```

The indexing will create a file called `Oct4.sorted.bam.bai`. Note that you don't have to specify the name of the index file when running `samtools index`, it simply appends a `.bai` suffix to the input BAM file.

Another way to visualize the alignments is to convert the BAM file into a bigWig file. The bigWig format is for display of dense, continuous data and the data will be displayed as a graph. The resulting bigWig files are in an indexed binary format.

The BAM to bigWig conversion takes place in two steps. Firstly, we convert the BAM file into a bedgraph, called `Oct4.bedgraph`, using the tool `genomeCoverageBed` from BEDTools. Then we convert the bedgraph into a bigWig binary file called `Oct4.bw`, using `bedGraphToBigWig` from the UCSC tools:

```
genomeCoverageBed -bg -ibam Oct4.sorted.bam -g \
    bowtie_index/mouse.mm9.genome > Oct4.bedgraph
bedGraphToBigWig Oct4.bedgraph bowtie_index/mouse.mm9.genome Oct4.bw
```

Both of the commands above take as input a file called `mouse.mm9.genome` that is stored under the subdirectory `bowtie_index`. These genome files are tab-delimited and describe the size of the chromosomes for the organism of interest. When using the UCSC Genome Browser, Ensembl, or Galaxy, you typically indicate which species/genome build you are working with. The way you do this for BEDTools is to create a "genome" file, which simply lists the names of the chromosomes (or scaffolds, etc.) and their size (in basepairs).

BEDTools includes pre-defined genome files for human and mouse in the `genomes` subdirectory included in the BEDTools distribution.

Now we will load the data into the IGV browser for visualization. In order to launch IGV double click on the `IGV 2.1` icon on your Desktop. Ignore any warnings and when it opens you have to load the genome of interest.

On the top left of your screen choose from the drop down menu `Mus musculus (mm9)`. Then in order to load the desire files go to:

```
File > Load from File
```

On the pop up window navigate to Desktop > ChIP-seq folder and select the file `Oct4.sorted.bam`.

Repeat these steps in order to load `Oct4.bw` as well.

Select `chr1` from the drop down menu on the top left. Right click on the name of `Oct4.bw` and choose Maximum under the Windowing Function. Right click again and select Autoscale.

In order to see the aligned reads of the BAM file, you need to zoom in to a specific region. For example, look for gene `Lemd1` in the search box.

What is the main difference between the visualization of BAM and bigWig files? The actual alignment of reads that stack to a particular region can be displayed using the information stored in a BAM format. The bigWig format is for display of dense, continuous data that will be displayed in the Genome Browser as a graph.

Using the `+` button on the top right, zoom in to see more of the details of the alignments.

What do you think the different colors mean? The different color represents four nucleotides, e.g. blue is Cytidine (C), red is Thymidine (T).

Practice Makes Perfect!

In the ChIP-seq folder you will find the file `gfp.fastq`. Follow the above described analysis, from the bowtie alignment step, for this dataset as well. You will need these files for the ChIP-Seq module.

Module: ChIP-Seq

Primary Author(s):
Remco Loos, EMBL-EBI remco@ebi.ac.uk
Myrto Kostadima kostadim@ebi.ac.uk

Contributor(s):
Xi Li sean.li@csiro.au

Key Learning Outcomes

After completing this practical the trainee should be able to:

- Perform simple ChIP-Seq analysis, e.g. the detection of immuno-enriched areas using the chosen peak caller program MACS
- Visualize the peak regions through a genome browser, e.g. Ensembl, and identify the real peak regions
- Perform functional annotation and detect potential binding sites (motif) in the predicted binding regions using motif discovery tool, e.g. MEME.

Resources You'll be Using

Tools Used

MACS
http://liulab.dfci.harvard.edu/MACS/index.html

Ensembl
http://www.ensembl.org

PeakAnalyzer
http://www.ebi.ac.uk/bertone/software

MEME
http://meme.ebi.edu.au/meme/cgi-bin/meme.cgi

TOMTOM
http://meme.ebi.edu/meme/cgi-bin/tomtom.cgi

DAVID
http://david.abcc.ncifcrf.gov

GOstat
http://gostat.wehi.edu.au

Sources of Data

http://www.ebi.ac.uk/arrayexpress/experiments/E-GEOD-11431

Introduction

The goal of this hands-on session is to perform some basic tasks in the analysis of ChIP-seq data. In fact, you already performed the first step, alignment of the reads to the genome, in the previous session. We start from the aligned reads and we will find immuno-enriched areas using the peak caller MACS. We will visualize the identified regions in a genome browser and perform functional annotation and motif analysis on the predicted binding regions.

Prepare the Environment

The material for this practical can be found in the `ChIP-seq` directory on your desktop. This directory also contains an electronic version of this document, which can be useful to copy and paste commands. Please make sure that this directory also contains the SAM/BAM files you produced during the alignment practical.

If you didn't have time to align the control file called `gfp.fastq` during the alignment practical, please do it now. Follow the same steps, from the bowtie alignment step, as for the `Oct4.fastq` file.

In ChIP-seq analysis (unlike in other applications such as RNA-seq) it can be useful to exclude all reads that map to more than one location in the genome. When using Bowtie, this can be done using the `-m 1` option, which tells it to report only unique matches (See `bowtie --help` for more details).

Open the Terminal and go to the `ChIP-seq` directory:

```
1 cd ~/ChIP-seq
```

Finding enriched areas using MACS

MACS stands for Model based analysis of ChIP-seq. It was designed for identifying transcription factor binding sites. MACS captures the influence of genome complexity to evaluate the significance of enriched ChIP regions, and improves the spatial resolution of binding sites through combining the information of both sequencing tag position and orientation. MACS can be easily used for ChIP-Seq data alone, or with a control sample to increase specificity.

Consult the MACS help file to see the options and parameters:

```
1 macs --help
```

The input for MACS can be in ELAND, BED, SAM, BAM or BOWTIE formats (you just have to set the `--format` option).

Options that you will have to use include:

`-t` To indicate the input ChIP file.

`-c` To indicate the name of the control file.

`--format` To change the file format. The default format is bed.

`--name` To set the name of the output files.

`--gsize` This is the mappable genome size. With the read length we have, 70% of the genome is a fair estimation. Since in this analysis we include only reads from chromosome 1 (197Mbases), we will use a `--gsize` of 138Mbases (70% of 197Mbases).

`--tsize` To set the read length (look at the FASTQ files to check the length).

`--wig` To generate signal wig files for viewing in a genome browser. Since this process is time consuming, it is recommended to run MACS first with this flag off, and once you decide on the values of the parameters, run MACS again with this flag on.

`--diag` To generate a saturation table, which gives an indication whether the sequenced reads give a reliable representation of the possible peaks.

Now run macs using the following command:

```
macs -t <Oct4_aligned_bam_file> -c <gfp_aligned_bam_file> --format=BAM \
   --name=Oct4 --gsize=138000000 --tsize=26 --diag --wig
```

Look at the output saturation table (`Oct4_diag.xls`). To open this file file, right-click on it and choose "Open with" and select LibreOffice. Do you think that more sequencing is necessary?

Open the Excel peak file and view the peak details. Note that the number of tags (column 6) refers to the number of reads in the whole peak region and not the peak height.

Viewing results with the Ensembl genome browser

It is often instructive to look at your data in a genome browser. Before, we used IGV, a stand-alone browser, which has the advantage of being installed locally and providing fast access. Web-based genome browsers, like Ensembl or the UCSC browser, are slower, but provide more functionality. They do not only allow for more polished and flexible visualisation, but also provide easy access to a wealth of annotations and external data sources. This makes it straightforward to relate your data with information about repeat regions, known genes, epigenetic features or areas of cross-species conservation, to name just a few. As such, they are useful tools for exploratory analysis.

They will allow you to get a 'feel' for the data, as well as detecting abnormalities and problems. Also, exploring the data in such a way may give you ideas for further analyses.

Launch a web browser and go to the Ensembl website at `http://www.ensembl.org/index.html`

Choose the genome of interest (in this case, mouse) on the left side of the page, browse to any location in the genome or click one of the demo links provided on the web page.

Click on the **Manage your data** link on the left, then choose **Attach remote file**.

Wig files are large so are inconvenient for uploading directly to the Ensemble Genome browser. Instead, we will convert it to an indexed binary format and put this into a web accessible place such as on a HTTP, HTTPS, or FTP server. This makes all the browsing process much faster. Detailed instructions for generating a bigWig from a wig type file can be found at:

`http://genome.ucsc.edu/goldenPath/help/bigWig.html`.

We have generated bigWig files in advance for you to upload to the Ensembl browser. They are at the following URL: `http://www.ebi.ac.uk/~remco/ChIP-Seq_course/Oct4.bw`

To visualise the data:

- Paste the location above in the field File URL.
- Choose data format bigWig.
- Choose some informative name and in the next window choose the colour of your preference.
- Click **Save** and close the window to return to the genome browser.

Repeat the process for the gfp control sample, located at `http://www.ebi.ac.uk/~remco/ChIP-Seq_course/gfp.bw`.

After uploading, choose **Configure this page**, and under **Your data** tick both boxes. Closing the window will save these changes.

Go to a region on chromosome 1 (e.g. `1:34823162-35323161`), and zoom in and out to view the signal and peak regions. Be aware that the y-axis of each track is auto-scaled independently of each other, so bigger-looking peaks may not actually be bigger! Always look at the values on the left hand side axis.

What can you say about the profile of Oct4 peaks in this region? There are no significant Oct4 peaks over the selected region.

Compare it with H3K4me3 histone modification wig file we have generated at `http://www.ebi.ac.uk/~remco/ChIP-Seq_course/H3K4me3.bw`. H3K4me3 has a region that contains relatively high peaks than Oct4.

Jump to `1:36066594-36079728` for a sample peak. Do you think H3K4me3 peaks regions contain one or more modification sites? What about Oct4? Yes. There are roughly three peaks, which indicate the possibility of having more than one modification sites in this region.

For Oct4, no peak can be observed.

MACS generates its peak files in a file format called bed file. This is a simple text format containing genomic locations, specified by chromosome, begin and end positions, and some more optional information.

See `http://genome.ucsc.edu/FAQ/FAQformat.html#format1` for details.

Bed files can also be uploaded to the Ensembl browser.

Try uploading the peak file generated by MACS to Ensembl. Find the first peak in the file (use the `head` command to view the beginning of the bed file), and see if the peak looks convincing to you.

Annotation: From peaks to biological interpretation

In order to biologically interpret the results of ChIP-seq experiments, it is usually recommended to look at the genes and other annotated elements that are located in proximity to the identified enriched regions. This can be easily done using PeakAnalyzer.

Go to the PeakAnalyzer directory and launch the program by typing:

```
cd ~ChIP-seq/PeakAnalyzer
java -jar PeakAnalyzer.jar &
```

The first window allows you to choose between the split application (which we will try next) and peak annotation. Choose the peak annotation option and click **Next**.

We would like to find the closest downstream genes to each peak, and the genes that overlap with the peak region. For that purpose you should choose the **NDG** option and click **Next**.

Fill in the location of the peak file `Oct4_peaks.bed`, and choose the mouse GTF as the annotation file. You don't have to define a symbol file since gene symbols are included in the GTF file.

Choose the output directory and run the program.

When the program has finished running, you will have the option to generate plots, by pressing the **Generate plots** button. This is only possible if R is installed on your computer, as it is on this system. A PDF file with the plots will be generated in the output folder. You could generate similar plots with Excel using the output files that were generated by PeakAnalyzer.

This list of closest downstream genes (contained in the file `Oct4_peaks.ndg.bed`) can be the basis of further analysis. For instance, you could look at the Gene Ontology terms associated with these genes to get an idea of the biological processes that may be affected. Web-based tools like DAVID (http://david.abcc.ncifcrf.gov) or GOstat (http://gostat.wehi.edu.au) take a list of genes and return the enriched GO categories.

We can pull out Ensemble Transcript IDs from the `Oct4_peaks.ndg.bed` file and write them to another file ready for use with DAVID or GOstat:

```
cut -f 5 Oct4_peaks.ndg.bed | sed '1 d' > Oct4_peaks.ndg.tid
```

Motif analysis

It is often interesting to find out whether we can associate identified the binding sites with a sequence pattern or motif. We will use MEME for motif analysis. The input for MEME should be a file in FASTA format containing the sequences of interest. In our case, these are the sequences of the identified peaks that probably contain Oct4 binding sites.

Since many peak-finding tools merge overlapping areas of enrichment, the resulting peaks tend to be much wider than the actual binding sites. Sub-dividing the enriched areas by accurately partitioning enriched loci into a finer-resolution set of individual binding sites, and fetching sequences from the summit region where binding motifs are most likely to appear enhances the quality of the motif analysis. Sub-peak summit sequences can be retrieved directly from the Ensembl database using PeakAnalyzer.

If you have closed the PeakAnalyzer running window, open it again. If it is still open, just go back to the first window.

Choose the split peaks utility and click **Next**. The input consists of files generated by most peak-finding tools: a file containing the chromosome, start and end locations of the enriched regions, and a `.wig` signal file describing the size and shape of each peak. Fill in the location of both files `Oct4_peaks.bed` and the wig file generated by MACS, which is under the `Oct4_MACS_wiggle/treat/` directory, check the option to **Fetch subpeak sequences** and click **Next**.

In the next window you have to set some parameters for splitting the peaks.

`Separation float` Keep the default value. This value determines when a peak will be separated into sub-peaks. This is the ratio between a valley and its neighbouring summit (the lower summit of the two). For example, if you set this height to be `0.5`, two sub-peaks will be separated only if the height of the lower summit is twice the height of the valley.

`Minimum height` Set this to be `5`. Only sub-peaks with at least this number of tags in their summit region will be separated. Change the organism name from the default human to mouse and run the program.

Since the program has to read large wig files, it will take a few minutes to run. Once the run is finished, two output files will be produced. The first describes the location of the sub-peaks, and the second is a FASTA file containing 300 sequences of length 61 bases, taken from the summit regions of the highest sub-peaks.

Open a web bowser and go to the MEME website at http://meme.ebi.edu.au/meme/cgi-bin/meme.cgi, and fill in the necessary details, such as:

- Your e-mail address
- The sub-peaks FASTA file `Oct4_peaks.bestSubPeaks.fa` (will need uploading), or just paste in the sequences.
- The number of motifs we expect to find (1 per sequence)
- The width of the desired motif (between 6 to 20)
- The maximum number of motifs to find (3 by default). For Oct4 one classical motif is known.

You will receive the results by e-mail. This usually doesn't take more than a few minutes.

Open the e-mail and click on the link that leads to the HTML results page.

Scroll down until you see the first motif logo. We would like to know if this motif is similar to any other known motif. We will use TOMTOM for this. Scroll down until you see the option **Submit this motif to**. Click the TOMTOM button to compare to known motifs in motif databases, and on the new page choose to compare your motif to those in the JASPAR and UniPROBE database.

Which motif was found to be the most similar to your motif? Sox2

Reference

Chen, X et al.: Integration of external signaling pathways with the core transcriptional network in embryonic stem cells. Cell 133:6, 1106-17 (2008).

Module: RNA-Seq

Primary Author(s):
Myrto Kostadima, EMBL-EBI kostadmi@ebi.ac.uk
Remco Loos, EMBL-EBI remco@ebi.ac.uk

Contributor(s):
Nathan S. Watson-Haigh nathan.watson-haigh@awri.com.au

Key Learning Outcomes

After completing this practical the trainee should be able to:

- Understand and perform a simple RNA-Seq analysis workflow.
- Perform gapped alignments to an indexed reference genome using TopHat.
- Perform transcript assembly using Cufflinks.
- Visualize transcript alignments and annotation in a genome browser such as IGV.
- Be able to identify differential gene expression between two experimental conditions.

Resources You'll be Using

Tools Used

Tophat
http://tophat.cbcb.umd.edu/

Cufflinks
http://cufflinks.cbcb.umd.edu/

Samtools
http://samtools.sourceforge.net/

BEDTools
http://code.google.com/p/bedtools/

UCSC tools
http://hgdownload.cse.ucsc.edu/admin/exe/

IGV
http://www.broadinstitute.org/igv/

DAVID Functional Analysis
http://david.abcc.ncifcrf.gov/

Sources of Data

http://www.ebi.ac.uk/ena/data/view/ERR022484
http://www.ebi.ac.uk/ena/data/view/ERR022485

Introduction

The goal of this hands-on session is to perform some basic tasks in the downstream analysis of RNA-seq data. We will start from RNA-seq data aligned to the zebrafish genome using Tophat.

We will perform transcriptome reconstruction using Cufflinks and we will compare the gene expression between two different conditions in order to identify differentially expressed genes.

Prepare the Environment

We will use a dataset derived from sequencing of mRNA from *Danio rerio* embryos in two different developmental stages. Sequencing was performed on the Illumina platform and generated 76bp paired-end sequence data using polyA selected RNA. Due to the time constraints of the practical we will only use a subset of the reads.

The data files are contained in the subdirectory called `data` and are the following:

`2cells_1.fastq` and `2cells_2.fastq`
: These files are based on RNA-seq data of a 2-cell zebrafish embryo

`6h_1.fastq` and `6h_2.fastq`
: These files are based on RNA-seq data of zebrafish embryos 6h post fertilization

Open the Terminal and go to the `RNA-seq` working directory:

```
1 cd ~/RNA-seq/
```

All commands entered into the terminal for this tutorial should be from within the **~/RNA-seq** directory.

Check that the `data` directory contains the above-mentioned files by typing:

```
1 ls data
```

Alignment

There are numerous tools for performing short read alignment and the choice of aligner should be carefully made according to the analysis goals/requirements. Here we will use Tophat, a widely used ultrafast aligner that performs spliced alignments.

Tophat is based on the Bowtie aligner and uses an indexed genome for the alignment to speed up the alignment and keep its memory footprint small. Created the index for the *Danio rerio* genome:

```
cd ~/RNA-seq
bowtie-build genome/Danio_rerio.Zv9.66.dna.fa genome/ZV9
```

Tophat has a number of parameters in order to perform the alignment. To view them all type:

```
tophat --help
```

The general format of the tophat command is:

```
tophat [options]* <index_base> <reads_1> <reads_2>
```

Where the last two arguments are the `.fastq` files of the paired end reads, and the argument before is the basename of the indexed genome.

The quality values in the FASTQ files used in this hands-on session are Phred+33 encoded. We explicitly tell tophat of this fact by using the command line argument `--solexa-quals`.

You can look at the first few reads in the file `data/2cells_1.fastq` with:

```
head -n 20 data/2cells_1.fastq
```

Some other parameters that we are going to use to run Tophat are listed below:

`-g` Maximum number of multihits allowed. Short reads are likely to map to more than one location in the genome even though these reads can have originated from only one of these regions. In RNA-seq we allow for a limited number of multihits, and in this case we ask Tophat to report only reads that map at most onto 2 different loci.

`--library-type` Before performing any type of RNA-seq analysis you need to know a few things about the library

preparation. Was it done using a strand-specific protocol or not? If yes, which strand? In our data the protocol was NOT strand specific.

`-j` Improve spliced alignment by providing Tophat with annotated splice junctions. Pre-existing genome annotation is an advantage when analysing RNA-seq data. This file contains the coordinates of annotated splice junctions from Ensembl. These are stored under the sub-directory `annotation` in a file called `ZV9.spliceSites`.

`-o` This specifies in which subdirectory Tophat should save the output files. Given that for every run the name of the output files is the same, we specify different directories for each run.

It takes some time (approx. 20 min) to perform tophat spliced alignments, even for this subset of reads. Therefore, we have pre-aligned the `2cells` data for you using the following command:

You DO NOT need to run this command yourself - we have done this for you.

```
1 tophat --solexa-quals -g 2 --library-type fr-unstranded -j \
    annotation/Danio_rerio.Zv9.66.spliceSites -o tophat/ZV9_2cells \
    genome/ZV9 data/2cells_1.fastq data/2cells_2.fastq
```

Align the `6h` data yourself using the following command:

```
1 # Takes approx. 20mins
2 tophat --solexa-quals -g 2 --library-type fr-unstranded -j \
    annotation/Danio_rerio.Zv9.66.spliceSites -o tophat/ZV9_6h \
    genome/ZV9 data/6h_1.fastq data/6h_2.fastq
```

The `6h` read alignment will take approx. 20 min to complete. Therefore, we'll take a look at some of the files, generated by tophat, for the pre-computed `2cells` data.

Alignment Visualisation in IGV

The Integrative Genomics Viewer (IGV) is able to provide a visualisation of read alignments given a reference sequence and a BAM file. We'll visualise the information contained in the `accepted_hits.bam` and `junctions.bed` files for the pre-computed `2cells` data. The former, contains the tophat sliced alignments of the reads to the reference while the latter stores the coordinates of the splice junctions present in the data set.

Open the `RNA-seq` directory on your Desktop and double-click the `tophat` subdirectory and then the `ZV9_2cells` directory.

1. Launch IGV by double-clicking the "IGV 2.1" icon on the Desktop (ignore any warnings that you may get as it opens). *NOTE: IGV may take several minutes to load for the first time, please be patient.*
2. Choose "Zebrafish (Zv9)" from the drop-down box in the top left of the IGV window.
3. Load the `accepted_hits.sorted.bam` file by clicking the "File" menu, selecting "Load from File" and navigating to the `Desktop/RNA-seq/tophat/ZV9_2cells` directory.
4. Rename the track by right-clicking on its name and choosing "Rename Track". Give it a meaningful name like "2cells BAM".
5. Load the `junctions.bed` from the same directory and rename the track "2cells Junctions BED".
6. Load the Ensembl annotations file `Danio_rerio.Zv9.66.gtf` stored in the `RNA-seq/annotation` directory.
7. Navigate to a region on chromosome 12 by typing `chr12:20,270,921-20,300,943` into the search box at the top of the IGV window.

Can you identify the splice junctions from the BAM file? Slice junctions can be identified in the alignment BAM files. These are the aligned RNA-Seq reads that have skipped-bases from the reference genome (most likely introns).

Are the junctions annotated for `CBY1` consistent with the annotation? Read alignment supports an extended length in exon 5 to the gene model (cby1-001)

Are all annotated genes, from both RefSeq and Ensembl, expressed? No BX000473.1-201 is not expressed

Once tophat finishes aligning the 6h data you will need to sort the alignments found in the BAM file and then index the sorted BAM file.

```
samtools sort tophat/ZV9_6h/accepted_hits.bam \
    tophat/ZV9_6h/accepted_hits.sorted
samtools index tophat/ZV9_6h/accepted_hits.sorted.bam
```

Load the sorted BAM file into IGV, as described previously, and rename the track appropriately.

Isoform Expression and Transcriptome Assembly

There are a number of tools that perform reconstruction of the transcriptome and for this workshop we are going to use Cufflinks. Cufflinks can do transcriptome assembly either *ab initio* or using a reference annotation. It also quantifies the isoform expression in Fragments Per Kilobase of exon per Million fragments mapped (FPKM).

Cufflinks has a number of parameters in order to perform transcriptome assembly and quantification. To view them all type:

```
1 cufflinks --help
```

We aim to reconstruct the transcriptome for both samples by using the Ensembl annotation both strictly and as a guide. In the first case Cufflinks will only report isoforms that are included in the annotation, while in the latter case it will report novel isoforms as well.

The Ensembl annotation for *Danio rerio* is available in `annotation/Danio_rerio.Zv9.66.gtf`.

The general format of the `cufflinks` command is:

```
cufflinks [options]* <aligned_reads.(sam|bam)>
```

Where the input is the aligned reads (either in SAM or BAM format).

Some of the available parameters for Cufflinks that we are going to use to run Cufflinks are listed below:

`-o` Output directory.

`-G` Tells Cufflinks to use the supplied GTF annotations strictly in order to estimate isoform annotation.

`-b` Instructs Cufflinks to run a bias detection and correction algorithm which can significantly improve accuracy of transcript abundance estimates. To do this Cufflinks requires a multi-fasta file with the genomic sequences against which we have aligned the reads.

`-u` Tells Cufflinks to do an initial estimation procedure to more accurately weight reads mapping to multiple locations in the genome (multi-hits).

`--library-type` Before performing any type of RNA-seq analysis you need to know a few things about the library preparation. Was it done using a strand-specific

protocol or not? If yes, which strand? In our data the protocol was NOT strand specific.

Perform transcriptome assembly, strictly using the supplied GTF annotations, for the `2cells` and `6h` data using cufflinks:

```
# 2cells data (takes approx. 5mins):
cufflinks -o cufflinks/ZV9_2cells_gtf -G \
    annotation/Danio_rerio.Zv9.66.gtf -b \
    genome/Danio_rerio.Zv9.66.dna.fa -u --library-type fr-unstranded \
    tophat/ZV9_2cells/accepted_hits.bam
# 6h data (takes approx. 5mins):
cufflinks -o cufflinks/ZV9_6h_gtf -G annotation/Danio_rerio.Zv9.66.gtf \
    -b genome/Danio_rerio.Zv9.66.dna.fa -u --library-type fr-unstranded \
    tophat/ZV9_6h/accepted_hits.bam
```

Cufflinks generates several files in the specified output directory. Here's a short description of these files:

`genes.fpkm_tracking` Contains the estimated gene-level expression values.

`isoforms.fpkm_tracking` Contains the estimated isoform-level expression values.

`skipped.gtf` Contains loci skipped as a result of exceeding the maximum number of fragments.

`transcripts.gtf` This GTF file contains Cufflinks' assembled isoforms.

The complete documentation can be found at: http://cufflinks.cbcb.umd.edu/manual.html#cufflinks_output

So far we have forced cufflinks, by using the `-G` option, to strictly use the GTF annotations provided and thus novel transcripts will not be reported. We can get cufflinks to perform a GTF-guided transcriptome assembly by using the `-g` option instead. Thus, novel transcripts will be reported.

GTF-guided transcriptome assembly is more computationally intensive than strictly using the GTF annotations. Therefore, we have pre-computed these GTF-guided assemblies for you and have placed the results under subdirectories: `cufflinks/ZV9_2cells_gtf_guided` and `cufflinks/ZV9_6h_gft_guided`.

You DO NOT need to run these commands. We provide them so you know how we generated the the GTF-guided transcriptome assemblies:

```
# 2cells guided transcriptome assembly (takes approx. 30mins):
cufflinks -o cufflinks/ZV9_2cells_gtf_guided -g \
    annotation/Danio_rerio.Zv9.66.gtf -b \
    genome/Danio_rerio.Zv9.66.dna.fa -u --library-type fr-unstranded \
    tophat/ZV9_2cells/accepted_hits.bam
# 6h guided transcriptome assembly (takes approx. 30mins):
cufflinks -o cufflinks/ZV9_6h_gtf_guided -g \
    annotation/Danio_rerio.Zv9.66.gtf -b \
    genome/Danio_rerio.Zv9.66.dna.fa -u --library-type fr-unstranded \
    tophat/ZV9_6h/accepted_hits.bam
```

1. Go back to IGV and load the pre-computed, GTF-guided transcriptome assembly for the `2cells` data (`cufflinks/ZV9_2cells_gtf_guided/transcripts.gtf`).
2. Rename the track as "2cells GTF-Guided Transcripts".
3. In the search box type `ENSDART00000082297` in order for the browser to zoom in to the gene of interest.

Do you observe any difference between the Ensembl GTF annotations and the GTF-guided transcripts assembled by cufflinks (the "2cells GTF-Guided Transcripts" track)? Yes. It appears that the Ensembl annotations may have truncated the last exon. However, our data also doesn't contain reads that span between the last two exons.

Differential Expression

One of the stand-alone tools that perform differential expression analysis is Cuffdiff. We use this tool to compare between two conditions; for example different conditions could be control and disease, or wild-type and mutant, or various developmental stages.

In our case we want to identify genes that are differentially expressed between two developmental stages; a `2cells` embryo and `6h` post fertilization.

The general format of the cuffdiff command is:

```
cuffdiff [options]* <transcripts.gtf> \
   <sample1_replicate1.sam[,...,sample1_replicateM]> \
   <sample2_replicate1.sam[,...,sample2_replicateM.sam]>
```

Where the input includes a `transcripts.gtf` file, which is an annotation file of the genome of interest, and the aligned reads (either in SAM or BAM format) for the conditions. Some of the Cufflinks options that we will use to run the program are:

`-o` Output directory.

`-L` Labels for the different conditions

`-T` Tells Cuffdiff that the reads are from a time series experiment.

`-b` Instructs Cufflinks to run a bias detection and correction algorithm which can significantly improve accuracy of transcript abundance estimates. To do this Cufflinks requires a multi-fasta file with the genomic sequences against which we have aligned the reads.

`-u` Tells Cufflinks to do an initial estimation procedure to more accurately weight reads mapping to multiple locations in the genome (multi-hits).

`--library-type` Before performing any type of RNA-seq analysis you need to know a few things about the library preparation. Was it done using a strand-specific protocol or not? If yes, which strand? In our data the protocol was NOT strand specific.

Run cuffdiff on the cufflinks generated BAM files for the 2cells vs. 6h data sets:

```
cuffdiff -o cuffdiff/ -L ZV9_2cells,ZV9_6h -T -b \
    genome/Danio_rerio.Zv9.66.dna.fa -u --library-type fr-unstranded \
    annotation/Danio_rerio.Zv9.66.gtf \
    tophat/ZV9_2cells/accepted_hits.bam tophat/ZV9_6h/accepted_hits.bam
```

We are interested in the differential expression at the gene level. The results are reported by Cuffdiff in the file `cuffdiff/gene_exp.diff`. Look at the first few lines of the file using the following command:

```
head -n 20 cuffdiff/gene_exp.diff
```

We would like to see which are the most significantly differentially expressed genes. Therefore we will sort the above file according to the q value (corrected p value for multiple testing). The result will be stored in a different file called `gene_exp_qval.sorted.diff`.

```
sort -t$'\t' -g -k 13 cuffdiff/gene_exp.diff > \
    cuffdiff/gene_exp_qval.sorted.diff
```

Look again at the first few lines of the sorted file by typing:

```
head -n 20 cuffdiff/gene_exp_qval.sorted.diff
```

Copy an Ensembl transcript identifier from the first two columns for one of these genes (e.g. `ENSDARG00000077178`). Now go back to the IGV browser and paste it in the search box.

Do you see any difference in the read coverage between the `2cells` and `6h` conditions that might have given rise to this transcript being called as differentially expressed?

Note that the coverage on the Ensembl browser is based on raw reads and no normalisation has taken place contrary to the FPKM values.

The read coverage of this transcript (`ENSDARG00000077178`) in the 2cells data set is much higher than in the 6h data set.

Functional Annotation of Differentially Expressed Genes

After you have performed the differential expression analysis you are interested in identifying if there is any functionality enrichment for your differentially expressed genes. On your Desktop click:

```
Applications >> Internet >> Firefox Web Browser
```

And go to the following URL: http://david.abcc.ncifcrf.gov/ On the left side click on Functional Annotation. Then click on the Upload tab. Under the section Choose from File, click Choose File and navigate to the `cuffdiff` directory. Select the file called `globalDiffExprs_Genes_qval.01_top100.tab`. Under Step 2 select ENSEMBL_GENE_ID from the drop-down menu. Finally select Gene list and then press Submit List. Click on Gene Ontology and then click on the CHART button of the GOTERM_BP_ALL item.

Do these categories make sense given the samples we're studying? Developmental Biology

Browse around DAVID website and check what other information are available. Cellular component, Molecular function, Biological Processes, Tissue expression, Pathways, Literature, Protein domains

References

1. Trapnell, C., Pachter, L. & Salzberg, S. L. TopHat: discovering splice junctions with RNA-Seq. Bioinformatics 25, 1105-1111 (2009).

2. Trapnell, C. et al. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. Nat. Biotechnol. 28, 511-515 (2010).

3. Langmead, B., Trapnell, C., Pop, M. & Salzberg, S. L. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. 10, R25 (2009).

4. Roberts, A., Pimentel, H., Trapnell, C. & Pachter, L. Identification of novel transcripts in annotated genomes using RNA-Seq. Bioinformatics 27, 2325-2329 (2011).

5. Roberts, A., Trapnell, C., Donaghey, J., Rinn, J. L. & Pachter, L. Improving RNA-Seq expression estimates by correcting for fragment bias. Genome Biol. 12, R22 (2011).

Module: *de novo* Genome Assembly

Primary Author(s):
Matthias Haimel mhaimel@ebi.ac.uk
Nathan S. Watson-Haigh nathan.watson-haigh@awri.com.au

Contributor(s):

Key Learning Outcomes

After completing this practical the trainee should be able to:

- Compile velvet with appropriate compile-time parameters set for a specific analysis
- Be able to choose appropriate assembly parameters
- Assemble a set of single-ended reads
- Assemble a set of paired-end reads from a single insert-size library
- Be able to visualise an assembly in AMOS Hawkeye
- Understand the importance of using paired-end libraries in *de novo* genome assembly

Resources You'll be Using

Although we have provided you with an environment which contains all the tools and data you will be using in this module, you may like to know where we have sourced those tools and data from.

Tools Used

Velvet
http://www.ebi.ac.uk/~zerbino/velvet/

AMOS Hawkeye
http://apps.sourceforge.net/mediawiki/amos/index.php?title=Hawkeye

gnx-tools
https://github.com/mh11/gnx-tools

FastQC
http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/

R
http://www.r-project.org/

Sources of Data

- ftp://ftp.ensemblgenomes.org/pub/release-8/bacteria/fasta/Staphylococcus/s_aureus_mrsa252/dna/s_aureus_mrsa252.EB1_s_aureus_mrsa252.dna.chromosome.Chromosome.fa.gz
- http://www.ebi.ac.uk/ena/data/view/SRS004748
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022825/SRR022825.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022823/SRR022823.fastq.gz
- http://www.ebi.ac.uk/ena/data/view/SRX008042
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022852/SRR022852_1.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022852/SRR022852_2.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR023/SRR023408/SRR023408_1.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR023/SRR023408/SRR023408_2.fastq.gz
- http://www.ebi.ac.uk/ena/data/view/SRX000181
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR000/SRR000892/SRR000892.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR000/SRR000893/SRR000893.fastq.gz
- http://www.ebi.ac.uk/ena/data/view/SRX007709
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022863/SRR022863_1.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022863/SRR022863_2.fastq.gz

Introduction

The aim of this module is to become familiar with performing *de novo* genome assembly using Velvet, a de Bruijn graph based assembler, on a variety of sequence data.

Prepare the Environment

The first exercise should get you a little more comfortable with the computer environment and the command line.

First make sure that you are in your home directory by typing:

```
cd
```

and making absolutely sure you're there by typing:

```
pwd
```

Now create sub-directories for this and the two other velvet practicals. All these directories will be made as sub-directories of a directory for the whole course called NGS. For this you can use the following commands:

```
mkdir -p NGS/velvet/{part1,part2,part3}
```

The `-p` tells `mkdir` (make directory) to make any parent directories if they don't already exist. You could have created the above directories one-at-a-time by doing this instead:

```
mkdir NGS
mkdir NGS/velvet
mkdir NGS/velvet/part1
mkdir NGS/velvet/part2
mkdir NGS/velvet/part3
```

After creating the directories, examine the structure and move into the directory ready for the first velvet exercise by typing:

```
ls -R NGS
cd NGS/velvet/part1
pwd
```

Downloading and Compiling Velvet

For the duration of this workshop, all the software you require has been set up for you already. This might not be the case when you return to "real life". Many of the programs you will need, including velvet, are quite easy to set up, it might be instructive to try a couple.

Although you will be using the preinstalled version of velvet, it is useful to know how to compile velvet as some of the parameters you might like to control can only be set at compile time. You can find the latest version of velvet at:

http://www.ebi.ac.uk/~zerbino/velvet/

You could go to this URL and download the latest velvet version, or equivalently, you could type the following, which will download, unpack, inspect, compile and execute your locally compiled version of velvet:

```
cd ~/NGS/velvet/part1
pwd
tar xzf ~/NGS/Data/velvet_1.2.07.tgz
ls -R
cd velvet_1.2.07
make velveth velvetg
./velveth
```

Take a look at the executables you have created. They will be displayed as green by the command:

```
ls --color=always
```

The switch `--color`, instructs that files be coloured according to their type. This is often the default but we are just being explicit. By specifying the value `always`, we ensure that colouring is always applied, even from a script.

Have a look of the output the command produces and you will see that `MAXKMERLENGTH=31` and `CATEGORIES=2` parameters were passed into the compiler.

This indicates that the default compilation was set for de Bruijn graph k-mers of maximum size 31 and to allow a maximum of just 2 read categories. You can override these, and other, default configuration choices using command line parameters. Assume, you want to run velvet with a k-mer length of 41 using 3 categories, velvet needs to be recompiled to enable this functionality by typing:

```
make clean
make velveth velvetg MAXKMERLENGTH=41 CATEGORIES=3;
```

```
./velveth
```

Discuss with the persons next to you the following questions:
What are the consequences of the parameters you have given make for velvet? MAXKMERLENGTH: increase the max k-mer length from 31 to 41

CATEGORIES: paired-end data require to be put into separate categories. By increasing this parameter from 2 to 3 allows you to process 3 paired / mate-pair libraries and unpaired data.

Why does Velvet use k-mer 31 and 2 categories as default? Possibly a number of reason:
- odd number to avoid palindromes
- The first reads were very short (20-40 bp) and there were hardly any paired-end data
around so there was no need to allow for longer k-mer lengths / more categories.
- For programmers: 31 bp get stored in 64 bits (using 2bit encoding)

Should you get better results by using a longer k-mer length? If you can achieve a good k-mer coverage - yes.

velvet can also be used to process SOLID colour space data. To do this you need a further make parameter. With the following command clean away your last compilation and try the following parameters:

```
make clean
make MAXKMERLENGTH=41 CATEGORIES=3 color
./velveth_de
```

What effect would the following compile-time parameters have on velvet:
`OPENMP=Y` Turn on multithreading

`LONGSEQUENCES=Y` Assembling reads / contigs longer than 32kb long

`BIGASSEMBLY=Y` Using more than 2.2 billion reads

`VBIGASSEMBLY=Y` Not documented yet

`SINGLE_COV_CAT=Y` Merge all coverage statistics into a single variable - save memory

For a further description of velvet compile and runtime parameters please see the velvet Manual: https://github.com/dzerbino/velvet/wiki/Manual

Assembling Single-end Reads

The following exercise focuses on velvet using single-end reads, how the available parameters effect an assembly and how to measure and compare the changes.

Even though you have carefully compiled velvet in your own workspace, we will be use the pre-installed version.

The data you will use is from Staphylococcus aureus USA300 which has a genome of around 3MBases. The reads are unpaired Illumina, also known as single-end library.

The data for this section was obtained from the Sequence Read Archive (SRA), using `SRR022825` and `SRR022823` run data from SRA Sample `SRS004748`. The SRA experiment can be viewed at:

http://www.ebi.ac.uk/ena/data/view/SRS004748

To begin with, first move back to the directory you prepared for this exercise, create a new folder with a suitable name for this part and move into it. There is no need to download the read files, as they are already stored locally. Instead we will create symlinks to the files. Continue by copying (or typing):

```
cd ~/NGS/velvet/part1
mkdir SRS004748
cd SRS004748
pwd
ln -s ~/NGS/Data/SRR022825.fastq.gz ./
ln -s ~/NGS/Data/SRR022823.fastq.gz ./
ls -l
```

You are ready to process your data with Velvet. There are two main components to Velvet:

`velveth` Used to construct, from raw read data, a dataset organised in the fashion expected by the second component, `velvetg`.

`velvetg` The core of velvet where the de Bruijn graph assembly is built and manipulated.

You can always get further information about the usage of both velvet programs by typing `velvetg` or `velveth` in your terminal.

Now run `velveth` for the reads in `SRR022825.fastq.gz` and `SRR022823.fastq.gz` using the following options:

- A de Bruijn graph k-mer of 25
- An output directory called run_25

```
velveth run_25 25 -fastq.gz -short SRR022825.fastq.gz SRR022823.fastq.gz
```

`velveth` Once `velveth` finishes, move into the output directory `run_25` and have a look at what `velveth` has generated so far. The command `less` allows you to look at output files (press `q` to quit and return to the command prompt). Here are some other options for looking at file contents:

```
cd run_25
ls -l
head Sequences
cat Log
```

What did you find in the folder `run_25`? Sequences, Roadmaps, Log

Describe the content of the two `velveth` output files? Sequences: FASTA file version of provided reads
Roadmaps: Internal file of velvet - basic information about reads, k-mer size

What does the `Log` file store for you? Time stamp, Executed commands; velvet version + compiler parameters, results

Now move one directory level up and run `velvetg` on your output directory, with the commands:

```
cd ../
time velvetg run_25
```

Move back into your results directory to examine the effects of `velvetg`:

```
cd run_25
ls -l
```

What extra files do you see in the folder `run_25`? PreGraph, Graph, stats.txt, contigs.fa, LastGraph

What do you suppose they might represent? PreGraph, Graph, LastGraph: Velvet internal graph representation at different stages (see manual for more details about the file format)

stats.txt: tab-delimited description of the nodes of the graph incl. coverage information

contigs.fa: assembly output file

In the Log file in `run_25`, what is the N50? 4409 bp

Hopefully, we will have discussed what the N50 statistic is by this point. Broadly, it is the median (not average) of a sorted data set using the length of a set of sequences. Usually it is the length of the contig whose length, when added to the length of all longer contigs, makes a total greater that half the sum of the lengths of all contigs. Easy, but messy - a more formal definition can be found here:

http://www.broadinstitute.org/crd/wiki/index.php/N50

Backup the `contigs.fa` file and calculate the N50 (and the N25,N75) value with the command:

```
1 cp contigs.fa contigs.fa.0
2 gnx -min 100 -nx 25,50,75 contigs.fa
```

Does the value of N50 agree with the value stored in the Log file? No

If not, why do you think this might be? K-mer N50 vs bp N50; contig length cut-off value, estimated genome length

In order to improve our results, take a closer look at the standard options of `velvetg` by typing `velvetg` without parameters. For the moment focus on the two options `-cov_cutoff` and `-exp_cov`. Clearly `-cov_cutoff` will allow you to exclude contigs for which the k-mer coverage is low, implying unacceptably poor quality. The `-exp_cov` switch is used to give `velvetg` an idea of the coverage to expect.

If the expected coverage of any contig is substantially in excess of the suggested expected value, maybe this would indicate a repeat. For further details of how to choose the parameters, go to "Choice of a coverage cutoff":

http://wiki.github.com/dzerbino/velvet/

Briefly, the k-mer coverage (and much more information) for each contig is stored in the file `stats.txt` and can be used with R to visualize the k-mer coverage distribution. Take a look at the `stats.txt` file, start R, load and visualize the data using the following commands:

```
R --no-save --no-restore
library(plotrix)
data <- read.table("stats.txt", header=TRUE)
weighted.hist(data$short1_cov, data$lgth, breaks=0:50)
```

A weighted histogram is a better way of visualizing the coverage information, because of noise (lots of very short contigs). You can see an example output below:

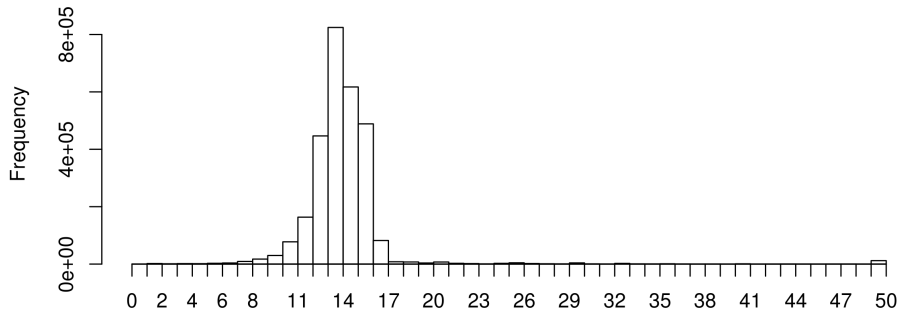

Figure 4: A weighted k-mer coverage histogram of the single-end reads.

After choosing the expected coverage and the coverage cut-off, you can exit R by typing:

```
q()
```

The weighted histogram suggests to me that the expected coverage is around 14 and that everything below 6 is likely to be noise. Some coverage is also represented at around 20, 30 and greater 50, which might be contamination or repeats (depending on the dataset), but at the moment this should not worry you. To see the improvements, rerun `velvetg` first with `-cov_cutoff` 6 and after checking the N50 use only / add `-exp_cov` 14 to the command line option. Also keep a copy of the contigs file for comparison:

```
cd ~/NGS/velvet/part1/SRS004748
time velvetg run_25 -cov_cutoff 6

# Make a copy of the run
cp run_25/contigs.fa run_25/contigs.fa.1

time velvetg run_25 -exp_cov 14
cp run_25/contigs.fa run_25/contigs.fa.2

time velvetg run_25 -cov_cutoff 6 -exp_cov 14
cp run_25/contigs.fa run_25/contigs.fa.3
```

What is the N50 with no parameter: 4,447 bp

What is the N50 with `-cov_cutoff` 6: 5,168 bp

What is the N50 with `-exp_cov` 14: 4,903 bp

What is the N50 with `-cov_cutoff` 6 `-exp_cov` 14: 5,417 bp

Did you notice a variation in the time `velvetg` took to run? If so, can you explain why that might be? Velvet reuses already calculated results (from PreGraph,Graph)

You were running `velvetg` with the `-exp_cov` and `-cov_cutoff` parameters. Now try to experiment using different cut-offs, expected parameters and also explore other settings (e.g. `-max_coverage`, `-max_branch_length`, `-unused_reads`, `-amos_file`, `-read_trkg` or see `velvetg` help menu).

Make some notes about the parameters you've played with and the results you obtained. -max_coverage: cut-off value for the upper range (like cov_cutoff for the lower range)
-max_branch_length: length of branch to look for bubble
-unused_reads: write unused reads into file
-amos_file: write AMOS message file
-read_trkg: tracking read (more memory usage) - automatically on for certain operations

AMOS Hawkeye

The `-amos_file` argument tells `velvetg` to output the assembly as an AMOS message file (`*.afg`) which can then be used by tools like Hawkeye from the AMOS suite of tools.

Lets create the AMOS message file by running `velvetg` with some appropriate parameters:

```
velvetg run_25 -cov_cutoff 6 -exp_cov 14 -amos_file yes
```

The `-exp_cov` argument to enable read-tracking `-read_trkg yes` in Velvet. Without read tracking enabled, very little read-level information can be output to the AMOS message file. This results in a pretty useless visualisation in Hawkeye! However, since reads are being tracked, the analysis takes longer and uses more memory.

Now convert the AMOS message file `velvet_asm.afg` into an AMOS bank using `bank-transact` and view the assembly with AMOS Hawkeye.

```
bank-transact -c -b run_25/velvet_asm.bnk -m run_25/velvet_asm.afg
hawkeye run_25/velvet_asm.bnk
```

Have a look around the interface, in particular try to look at the "Scaffold View" and "Contig View" of the larges scaffold. You should see something like this:

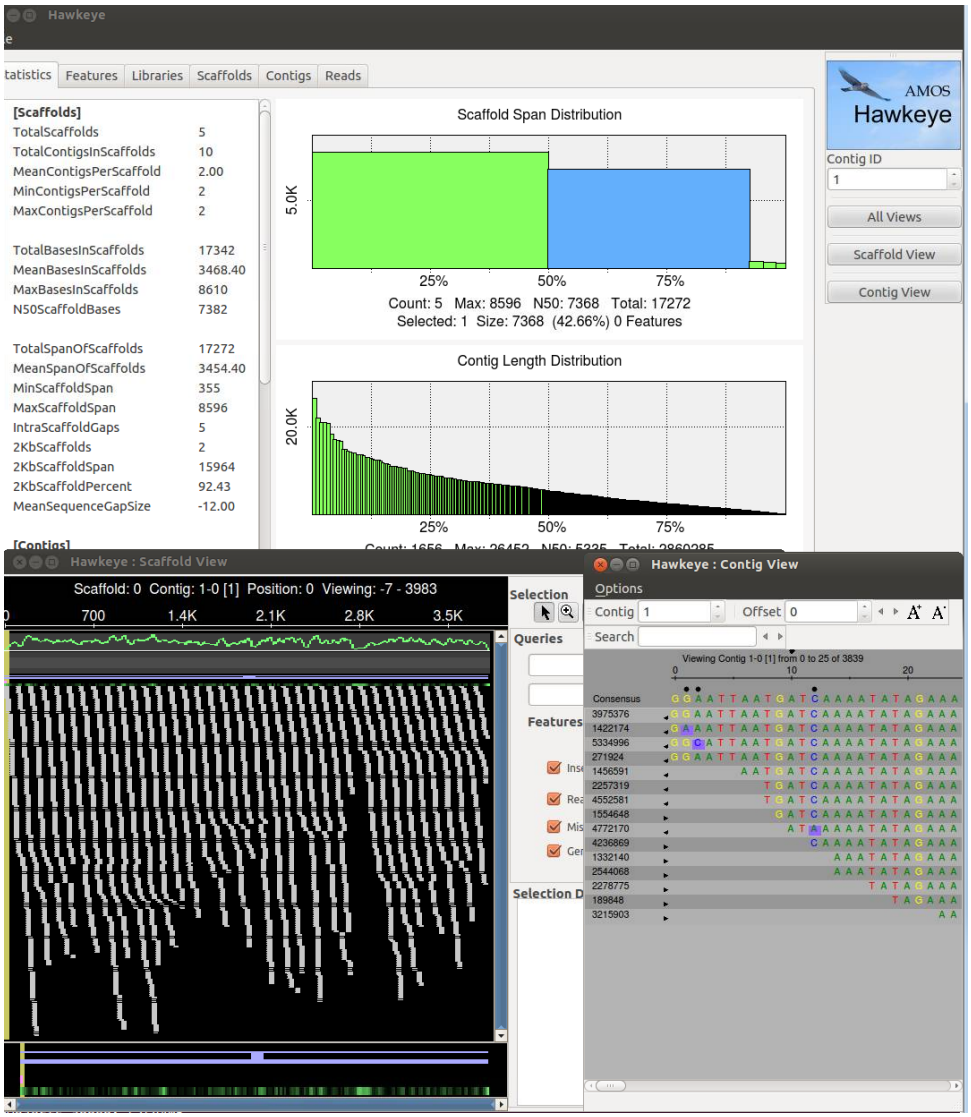

Figure 5:

If you have time, try running the `velvetg` command without the `-exp_cov` argument, create the AMOS bank and see how the assemblies look different in Hawkeye. Here's a hint:

```
velvetg run_25 -cov_cutoff 6 -amos_file yes
bank-transact -c -b run_25/velvet_asm.bnk -m run_25/velvet_asm.afg
hawkeye run_25/velvet_asm.bnk
```

Simple Assembly Simulation

The data for this section is from Staphylococcus aureus MRSA252, a genome closely related to the genome that provided the short read data in the earlier sections of this exercise. The sequence data this time is the fully assembled genome. The genome size is therefore known exactly and is 2,902,619 bp.

In this exercise you will process the single whole genome sequence with `velveth` and `velvetg`, look at the output only and go no further. The main intent of processing this whole genome is to compute its N50 value. This must clearly be very close to the ideal N50 value for the short reads assembly and so aid evaluation of that assembly.

To begin, move back to the main directory for this exercise, make a sub-directory for the processing of this data and move into it. All in one go, this would be:

```
cd ~/NGS/velvet/part1/
mkdir MRSA252
cd MRSA252
```

Next, you need to download the genome sequence from http://www.ensemblgenomes.org/ which holds five new sites, for bacteria, protists, fungi, plants and invertebrate metazoa. You could browse for the data you require or use the file which we have downloaded for you. For the easier of these options, make and check a symlink to the local file and with the commands:

```
ln -s ~\
   /NGS/Data/s_aureus_mrsa252.EB1_s_aureus_mrsa252.dna.chromosome.Chromosome.fa.gz \
   ./
ls -l
```

Usually Velvet expects relatively short sequence entries and for this reason has a read limit of 32,767 bp per sequence entry. As the genome size is 2,902,619 bp - longer as the allowed limit and does not fit with the standard settings into velvet. But like the maximum k-mer size option, you can tell Velvet during compile time, using `LONGSEQUENCES=Y`, to expect longer input sequences than usual. I already prepared the executable which you can use by typing `velveth_long` and `velvetg_long`.

Now, run `velveth_long`, using the file you either just downloaded or created a symlink to as the input:

```
velveth_long run_25 25 -fasta.gz -long \
    s_aureus_mrsa252.EB1_s_aureus_mrsa252.dna.chromosome.Chromosome.fa.gz
velvetg_long run_25
```

What is the N50? 24,142 bp

How does the N50 compare to the previous single end run (SRS004748)? Big difference

Does the total length differ from the input sequence length? 2,817,181 (stats) vs 2,902,619 (input)

What happens when you rerun velvet with a different k-mer length? K-mer 31: N50: 30,669 bp, total 2,822,878

Assembling Paired-end Reads

The use of paired-end data in *de novo* genome assembly results in better quality assemblies, particularly for larger, more complex genomes. In addition, paired-end constraint violation (expected distance and orientation of paired reads) can be used to identify misassemblies.

If you are doing *de novo* assembly, pay the extra and get paired-ends: they're worth it!

The data you will examine in this exercise is again from Staphylococcus aureus which has a genome of around 3MBases. The reads are Illumina paired end with an insert size of 350 bp.

The required data can be downloaded from the SRA. Specifically, the run data (SRR022852) from the SRA Sample SRS004748.

http://www.ebi.ac.uk/ena/data/view/SRS004748

The following exercise focuses on preparing the paired-end FASTQ files ready for Velvet, using Velvet in paired-end mode and comparing results with Velvet's 'auto' option.

First move to the directory you made for this exercise and make a suitable named directory for the exercise:

```
cd ~/NGS/velvet/part2
mkdir SRS004748
cd SRS004748
```

There is no need to download the read files, as they are already stored locally. You will simply create a symlink to this pre-downloaded data using the following commands:

```
ln -s ~/NGS/Data/SRR022852_?.fastq.gz ./
```

It is interesting to monitor the computer's resource utilisation, particularly memory. A simple way to do this is to open a second terminal and in it type:

```
top
```

`top` is a program that continually monitors all the processes running on your computer, showing the resources used by each. Leave this running and refer to it periodically throughout your Velvet analyses. Particularly if they are taking a long time or whenever your curiosity gets the better of you. You should find that as this practical progresses, memory usage will increase significantly.

Now, back to the first terminal, you are ready to run `velveth` and `velvetg`. The reads are `-shortPaired` and for the first run you should not use any parameters for `velvetg`.

From this point on, where it will be informative to time your runs. This is very easy to do, just prefix the command to run the program with the command `time`. This will cause UNIX to report how long the program took to complete its task.

Set the two stages of velvet running, whilst you watch the memory usage as reported by `top`. Time the `velvetg` stage:

```
velveth run_25 25 -fmtAuto -create_binary -shortPaired -separate \
    SRR022852_1.fastq.gz SRR022852_2.fastq.gz
time velvetg run_25
```

What does `-fmtAuto` and `-create_binary` do? (see help menu) `-fmtAuto` tries to detect the correct format of the input files e.g. FASTA, FASTQ and whether they are compressed or not.

`-create_binary` outputs sequences as a binary file. That means that `velvetg` can read the sequences from the binary file more quickly that from the original sequence files.

Comment on the use of memory and CPU for `velveth` and `velvetg`? `velveth` uses only one CPU while `velvetg` uses all possible CPUs for some parts of the calculation.

How long did `velvetg` take? My own measurements are:
`real 1m8.877s; user 4m15.324s; sys 0m4.716s`

Next, after saving your `contigs.fa` file from being overwritten, set the cut-off parameters that you investigated in the previous exercise and rerun `velvetg`. time and monitor the use of resources as previously. Start with `-cov_cutoff 16` thus:

```
1 mv run_25/contigs.fa run_25/contigs.fa.0
2 time velvetg run_25 -cov_cutoff 16
```

Up until now, `velvetg` has ignored the paired-end information. Now try running `velvetg` with both `-cov_cutoff 16` and `-exp_cov 26`, but first save your `contigs.fa` file. By using `-cov_cutoff` and `-exp_cov`, `velvetg` tries to estimate the insert length, which you will see in the `velvetg` output. The command is, of course:

```
1 mv run_25/contigs.fa run_25/contigs.fa.1
2 time velvetg run_25 -cov_cutoff 16 -exp_cov 26
```

Comment on the time required, use of memory and CPU for `velvetg`? Runtime is lower when velvet can reuse previously calculated data. By using `-exp_cov`, the memory usage increases.

Which insert length does Velvet estimate? Paired-end library 1 has length: 228, sample standard deviation: 26

Next try running `velvetg` in 'paired-end mode'. This entails running `velvetg` specifying the insert length with the parameter `-ins_length` set to 350. Even though velvet estimates the insert length it is always advisable to check / provide the insert length manually as velvet can get the statistics wrong due to noise. Just in case, save your last version of `contigs.fa`. The commands are:

```
1 mv run_25/contigs.fa run_25/contigs.fa.2
```

```
time velvetg run_25 -cov_cutoff 16 -exp_cov 26 -ins_length 350
mv run_25/contigs.fa run_25/contigs.fa.3
```

How fast was this run? My own measurements are:
`real 0m29.792s; user 1m4.372s; sys 0m3.880s`

Take a look into the Log file.

What is the N50 value for the `velvetg` runs using the switches:
Base run: 19,510 bp `-cov_cutoff 16` 24,739 bp

`-cov_cutoff 16 -exp_cov 26` 61,793 bp

`-cov_cutoff 16 -exp_cov 26 -ins_length 350` n50 of 62,740 bp; max 194,649 bp; total 2,871,093 bp

Try giving the `-cov_cutoff` and/or `-exp_cov` parameters the value `auto`. See the `velvetg` help to show you how. The information Velvet prints during running includes information about the values used (coverage cut-off or insert length) when using the `auto` option.

What coverage values does Velvet choose (hint: look at the output that Velvet produces while running)? Median coverage depth = 26.021837
Removing contigs with coverage < 13.010918 ...

How does the N50 value change? n50 of 68,843 bp; max 194,645 bp; total 2,872,678 bp

Run `gnx` on all the `contig.fa` files you have generated in the course of this exercise. The command will be:

```
gnx -min 100 -nx 25,50,75 run_25/contigs.fa*
```

For which runs are there Ns in the `contigs.fa` file and why? contigs.fa.2, contigs.fa.3, contigs.fa
Velvet tries to use the provided (or infers) the insert length and fills ambiguous regions with Ns.

Comment on the number of contigs and total length generated for each run.

Filename Total length	# contigs # Ns
Contigs.fa.0 2,830,659	631 0
Contigs.fa.1 2,832,670	580 0
Contigs.fa.2 2,849,919	166 4,847
Contigs.fa.3 2,856,795	166 11,713
Contigs.fa 2,857,439	163 11,526

Table 6:

AMOS Hawkeye

We will now output the assembly in the AMOS massage format and visualise the assembly using AMOS Hawkeye.

Run `velvetg` with appropriate arguments and output the AMOS message file, then convert it to an AMOS bank and open it in Hawkeye:

```
1 time velvetg run_25 -cov_cutoff 16 -exp_cov 26 -ins_length 350 \
      -amos_file yes -read_trkg yes
2 time bank-transact -c -b run_25/velvet_asm.bnk -m run_25/velvet_asm.afg
3 hawkeye run_25/velvet_asm.bnk
```

Looking at the scaffold view of a contig, comment on the proportion of "happy mates" to "compressed mates" and "stretched mates". Nearly all mates are compressed with no stretched mates and very few happy mates.

What is the mean and standard deviation of the insert size reported under the Libraries tab? Mean: 350 bp SD: 35 bp

Look at the actual distribution of insert sizes for this library. Can you explain where there is a difference between the mean and SD reported in those two places? We specified `-ins_length 350` to the `velvetg` command. Velvet uses this value, in the AMOS message file that it outputs, rather than its own estimate.

You can get AMOS to re-estimate the mean and SD of insert sizes using intra-contig pairs. First, close Hawkeye and then run the following commands before reopening the AMOS bank to see what has changed.

```
1 asmQC -b run_25/velvet_asm.bnk -scaff -recompute -update -numsd 2
2 hawkeye run_25/velvet_asm.bnk
```

Looking at the scaffold view of a contig, comment on the proportion of "happy mates" to "compressed mates" and "stretched mates". There are only a few compressed and stretched mates compared to happy mates. There are similar numbers of stretched and compressed mates.

What is the mean and standard deviation of the insert size reported under the Libraries tab? TODO Mean: 226 bp SD: 25 bp

Look at the actual distribution of insert sizes for this library. Does the mean and SD reported in both places now match? Yes

Can you find a region with an unusually high proportion of stretched, compressed, incorrectly orientated or linking mates? What might this situation indicate? This would indicate a possible misassembly and worthy of further investigation.

Look at the largest scaffold, there are stacks of stretched pairs which span contig boundaries. This indicates that the gap size has been underestimated during the scaffolding phase.

Velvet and Data Quality

So far we have used the raw read data without performing any quality control or read trimming prior to doing our velvet assemblies.

Velvet does not use quality information present in FASTQ files.

For this reason, it is vitally important to perform read QC and quality trimming. In doing so, we remove errors/noise from the dataset which in turn means velvet will run faster, will use less memory and will produce a better assembly. Assuming we haven't compromised too much on coverage.

To investigate the effect of data quality, we will use the run data (SRR023408) from the SRA experiment SRX008042. The reads are Illumina paired end with an insert size of 92 bp.

Go back to the main directory for this exercise and create and enter a new directory dedicated to this phase of the exercise. The commands are:

```
cd ~/NGS/velvet/part2
mkdir SRX008042
cd SRX008042
```

Create symlinks to the read data files that we downloaded for you from the SRA:

```
ln -s ~/NGS/Data/SRR023408_?.fastq.gz ./
```

We will use FastQC, a tool you should be familiar with, to visualise the quality of our data. We will use FastQC in the Graphical User Interface (GUI) mode.

Start FastQC and set the process running in the background, by using a trailing `&`, so we get control of our terminal back for entering more commands:

```
fastqc &
```

Open the two compressed FASTQ files (File $->$ Open) by selecting them both and clicking OK). Look at tabs for both files:

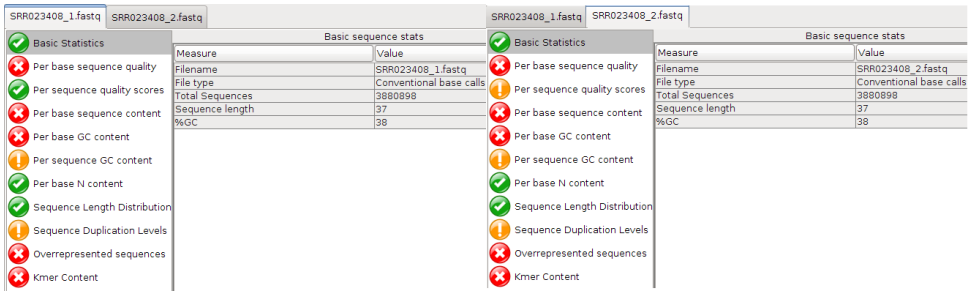

Figure 6:

Are the quality scores the same for both files? Overall yes

Which value varies? Per sequence quality scores

Take a look at the Per base sequence quality for both files. Did you note that it is not good for either file? The quality score of both files drop very fast. Qualities of the REV strand drop faster than the FWD strand. This is because the template has been sat around while the FWD strand was sequenced.

At which positions would you cut the reads if we did "fixed length trimming"? Looking at the "Per base quality" and "Per base sequence content", I would choose around 27

Why does the quality deteriorate towards the end of the read? Errors more likely for later cycles

Does it make sense to trim the 5' start of reads? Looking at the "Per base sequence content", yes - there is a clear signal at the beginning.

Have a look at the other options that FastQC offers.

Which other statistics could you use to support your trimming strategy? "Per base sequence content", "Per base GC content", "Kmer content", "Per base sequence quality"

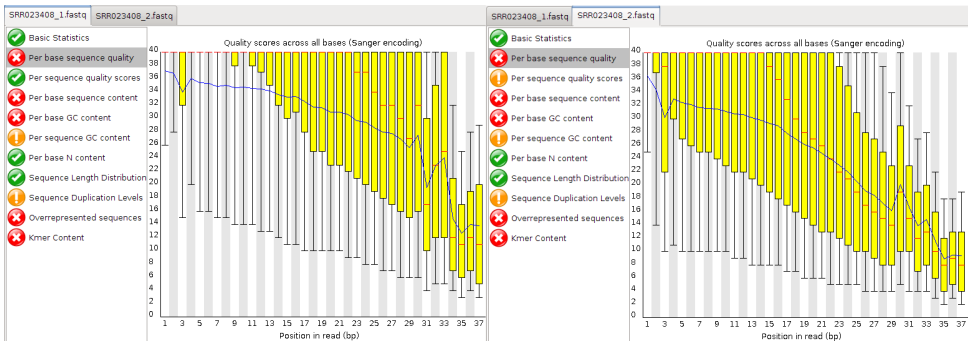

Figure 7:

Once you have decided what your trim points will be, close FastQC. We will use `fastx_trimmer` from the FASTX-Toolkit to perform fixed-length trimming. For usage information see the help:

```
1 fastx_trimmer -h
```

`fastx_trimmer` is not able to read compressed FASTQ files, so we first need to decompress the files ready for input.

The suggestion (hopefully not far from your own thoughts?) is that you trim your reads as follows:

```
1 gunzip < SRR023408_1.fastq.gz > SRR023408_1.fastq
2 gunzip < SRR023408_2.fastq.gz > SRR023408_2.fastq
3 fastx_trimmer -Q 33 -f 4 -l 32 -i SRR023408_1.fastq -o \
    SRR023408_trim1.fastq
4 fastx_trimmer -Q 33 -f 3 -l 29 -i SRR023408_2.fastq -o \
    SRR023408_trim2.fastq
```

Many NGS read files are large. This means that simply reading and writing files can become the bottleneck, also known as I/O bound. Therefore, it is often good practice to avoid unnecessary disk read/write.

We could do what is called pipelining to send a stream of data from one command to another, using the pipe (|) character, without the need for intermediary files. The following command would achieve this:

```
1 gunzip --to-stdout < SRR023408_1.fastq.gz | fastx_trimmer -Q 33 -f 4 \
    -l 32 -o SRR023408_trim1.fastq
2 gunzip --to-stdout < SRR023408_2.fastq.gz | fastx_trimmer -Q 33 -f 3 \
    -l 29 -o SRR023408_trim2.fastq
```

Now run `velveth` with a k-mer value of 21 for both the untrimmed and trimmed read files in `-shortPaired` mode. Separate the output of the two executions of `velveth` into suitably named directories, followed by `velvetg`:

```
# untrimmed reads
velveth run_21 21 -fmtAuto -create_binary -shortPaired -separate \
    SRR023408_1.fastq SRR023408_2.fastq
time velvetg run_21

# trimmed reads
velveth run_21trim 21 -fmtAuto -create_binary -shortPaired -separate \
    SRR023408_trim1.fastq SRR023408_trim2.fastq
time velvetg run_21trim
```

How long did the two `velvetg` runs take? run_25: `real 3m16.132s; user 8m18.261s; sys 0m7.317s`
run_25trim: `real 1m18.611s; user 3m53.140s; sys 0m4.962s`

What N50 scores did you achieve? Untrimmed: 11
Trimmed: 15

What were the overall effects of trimming? Time saving, increased N50, reduced coverage

The evidence is that trimming improved the assembly. The thing to do surely, is to run `velvetg` with the `-cov_cutoff` and `-exp_cov`. In order to use `-cov_cutoff` and `-exp_cov` sensibly, you need to investigate with R, as you did in the previous exercise, what parameter values to use. Start up R and produce the weighted histograms:

```
R --no-save
library(plotrix)
data <- read.table("run_21/stats.txt", header=TRUE)
data2 <- read.table("run_21trim/stats.txt", header=TRUE)
par(mfrow=c(1,2))
weighted.hist(data$short1_cov, data$lgth, breaks=0:50)
weighted.hist(data2$short1_cov, data2$lgth, breaks=0:50)
```

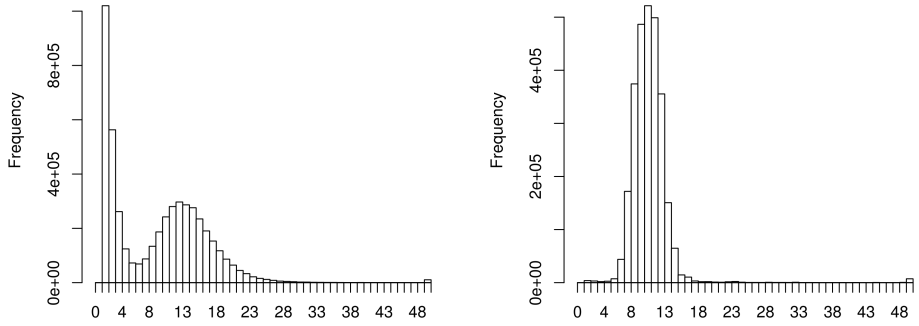

Figure 8: Weighted k-mer coverage histograms of the paired-end reads pre-trimmed (left) and post-trimmed (right).

For the untrimmed read histogram (left) there is an expected coverage of around 13 with a coverage cut-off of around 7. For the trimmed read histogram (right) there is an expected coverage of around 9 with a coverage cut-off of around 5.

If you disagree, feel free to try different settings, but first quit R before running `velvetg`:

```
q()
```

```
time velvetg run_21 -cov_cutoff 7 -exp_cov 13 -ins_length 92
time velvetg run_21trim -cov_cutoff 5 -exp_cov 9 -ins_length 92
```

How good does it look now?
Still not great Comment on:
Runtime Reduced runtime

Memory Lower memory usage

k-mer choice (Can you use k-mer 31 for a read of length 30 bp?) K-mer has to be lower than the read length and the K-mer coverage should be sufficient to produce results.

Does less data mean "worse" results? Not necessarily. If you have lots of data you can safely remove poor data without too much impact on overall coverage.

How would a smaller/larger k-mer size behave?

Compare the results, produced during the last exercises, with each other:

Metric	SRR022852	SRR023408	SRR023408.trimmed
Overall Quality (1-5)			
bp Coverage			
k-mer Coverage			
N50 (k-mer used)			

Table 7:

Metric	SRR022852	SRR023408	SRR023408.trimmed
Overall Quality (1-5)	2	5	4
bp Coverage	136 x (36 bp;11,374,488)	95x (37bp; 7761796)	82x (32bp; 7761796)
k-mer Coverage	45x	43x (21); 33x (25)	30x (21); 20.5x (25)
N50 (k-mer used)	68,843 (25)	2,803 (21)	2,914 (21)

Table 8:

What would you consider as the "best" assembly? SRR022852

If you found a candidate, why do you consider it as "best" assembly? Overall data quality and coverage

How else might you assess the the quality of an assembly? Hint: Hawkeye. By trying to identify paired-end constraint violations using AMOS Hawkeye.

Hybrid Assembly

Like the previous examples, the data you will examine in this exercise is again from Staphylococcus aureus which has a genome of around 3MB. The reads are 454 single end and Illumina paired end with an insert size of 170 bp. You already downloaded the required reads from the SRA in previous exercises. Specifically, the run data (SRR022863, SRR000892, SRR000893) from the SRA experiments SRX007709 and SRX000181.

The following exercise focuses on handing 454 long reads and paired-end reads with velvet and the differences in setting parameters.

First move to the directory you made for this exercise, make a suitable named directory for the exercise and check if all the three files are in place:

```
cd ~/NGS/velvet/part3
mkdir SRR000892-SRR022863
cd SRR000892-SRR022863
# 454 single end data
ln -s ~/NGS/Data/SRR00089[2-3].fastq.gz ./
# illumina paired end data
ln -s ~/NGS/Data/SRR022863_?.fastq.gz ./
```

The following command will run for a LONG time. This indicated the amount of calculations being preformed by Velvet to reach a conclusion. To wait for velvet to finish would exceed the time available in this workshop, but it is up to you to either let it run over night or kill the process by using the key combination `CTRL+c`.

```
velveth run_25 25 -fmtAuto -create_binary -long \
    SRR00089?.fastq.gz -shortPaired -separate \
    SRR022863_1.fastq.gz SRR022863_2.fastq.gz
time velvetg run_25
```

If you have decided to continue, we already inspected the weighted histograms for the short and long read library separately, you can reuse this for the cut-off values:

```
time velvetg run_25 -cov_cutoff 7 -long_cov_cutoff 9
```

What are your conclusions using Velvet in an hybrid assembly? 17 min: time velvetg run_25

Post-Workshop Information

Access to Computational Resources

By the end of the workshop we hope you're thinking one or more of the following:

- I'm interested in dabbling some more during my day-job!
- How do I access a Linux box like the one I've been using in the workshop - I *really* don't want the hassle of setting this all up myself!
- I'm hooked! I *really* want to get down and dirty with NGS data! What computational resources do I need, what do I have access to and how do I access them?

We're ecstatic you're thinking this way and want to help guide you! However, lets take this one step at a time.

The quickest way to dabble is to use a clone of the operating system (OS) you've been using during this workshop. That means you'll have hassle-free access to a plethora of pre-installed, pre-configured bioinformatics tools. You could even set it up to contain a copy of all the workshop data and handouts etc to go through the hands-on practicals in your own time!

We have created an image file (approx. 10 GBytes in size) of the NGS Training OS for you to use as you wish:

https://swift.rc.nectar.org.au:8888/v1/AUTH_33065ff5c34a4652aa2fefb292b3195a/VMs/NGSTrainingV1.2.vdi

We would advise one of the following two approaches for making use of it:

- Import it into VirtualBox to setup a virtual machine (VM) on your own computer.
- Instantiate a VM on the NeCTAR Research Cloud.

Setting up a VM using VirtualBox

This approach requires the least amount of mind-bending to get up and running. However, you will need to install some software. If you do not have administrator access or your system administrator is slow or unwilling to install the software, you may find using the NeCTAR Research Cloud to be viable alternative.

This approach will use, at most, the computational resources available on your own computer. If you are analysing non-microbial organisms or performing *de novo* assemblies, you may find these resources are insufficient. If this is the case, you really should speak to someone from IT support at your institution or get in touch with a bioinformatician for advise.

The software you need is VirtualBox, a freely available, Open Source virtualisation product from Oracle (https://www.virtualbox.org/). This software essentially allows you to

run an operating system (the guest OS) within another (the host OS). VirtualBox is available for several different host OSes including MS Windows, OS X, Linux and Solaris (https://www.virtualbox.org/wiki/Downloads). Once VirtualBox is installed on your host OS, you can then install a guest OS inside VirtualBox. VirtualBox supports a lot of different OSes (https://www.virtualbox.org/wiki/Guest_OSes).

Here are the steps to setting up a VM in VirtualBox with our image file:

1. Download and install VirtualBox for your OS: https://www.virtualbox.org/wiki/Downloads
2. Start VirtualBox and click New to start the Create New Virtual Machine wizard
3. Give the VM a useful name like "NGS Training" and choose Linux and either Ubuntu or Ubuntu (64-bit) as the OS Type
4. Give the VM access to a reasonable amount of the host Oses memory. i.e. somewhere near the top of the green. If this value is < 2000 MB, you are likely to have insufficient memory for your NGS data analysis needs.
5. For the virtual hard disk, select "Use existing hard disk" and browse to and select the `NGSTrainingV1.2.vdi` file you downloaded.
6. Confirm remaining settings
7. Select the "NGS Training" VM and click Start to boot he machine.
8. Once booted, log into the VM as either `ubuntu` (a sudoer user; i.e. has admin rights) or as `ngstrainee` (a regular unprivileged user). See table below for passwords.

Setting up a VM using the NeCTAR Research Cloud

All Australian researchers, who are members of an institution which subscribes to the Australian Access Federation (AAF; http://www.aaf.edu.au/), have access to a small amount of computing resources (2 CPU's and 8 GBytes RAM) on the NeCTAR Research Cloud (http://nectar.org.au/research-cloud).

Login to the NeCTAR Research Cloud Dashboard

The online dashboard is a graphical interface for administering (creating, deleting, rebooting etc) your virtual machines (VMs) on the NeCTAR research cloud.

1. Go to the dashboard: `http://dashboard.rc.nectar.org.au`
2. When you see the following page, click the "Log In" button:

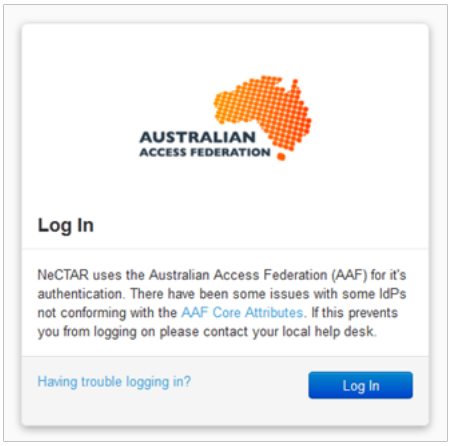

Figure 9:

3. At the following screen, simply choose your institution from the dropdown box and click "Select". Now follow the on screen prompts and enter your regular institutional login details.

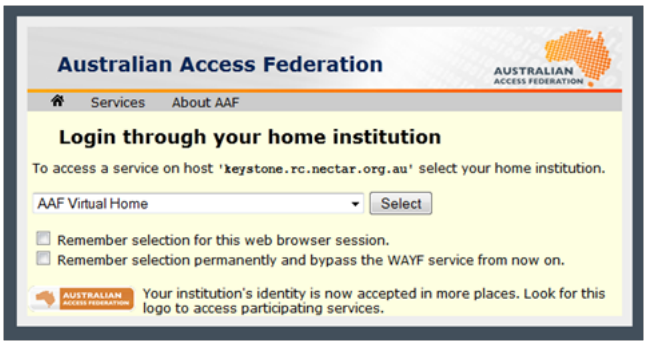

Figure 10:

4. If you see the following screen, congratulations, you have successfully logged into the NeCTAR Research Cloud dashboard!

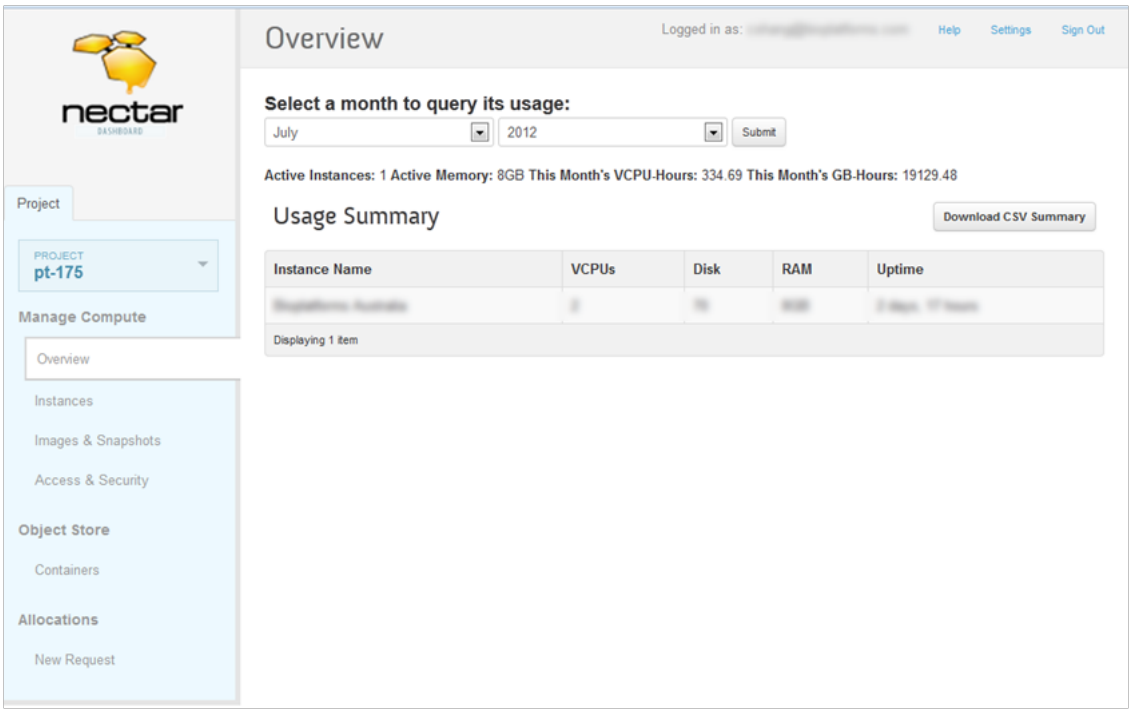

Figure 11:

Instantiating Your Own VM

We will now show you how to instantiate the "NGS Training" image using your own personal cloud allocation.

1. In the NeCTAR Research Cloud dashboard, click "Images & Snapshots" to list all the publicly available images from which you can instantiate a VM. Under "Snapshots" Click the "Launch" button for the latest version of the "NGSTraining" snapshot:

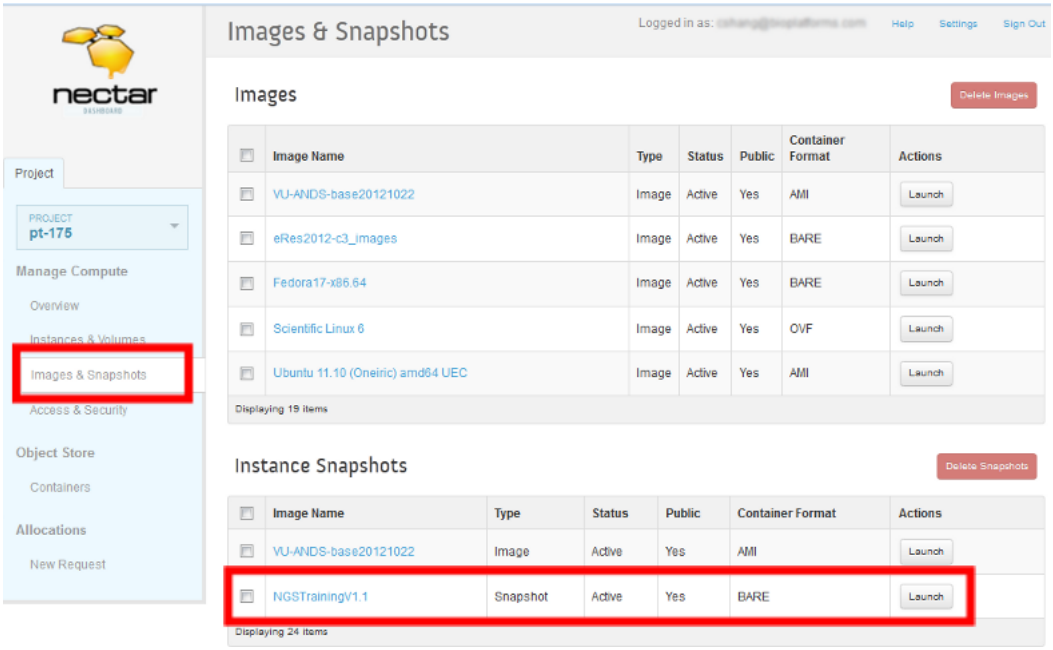

Figure 12:

2. You will now see a "Launch Instances" window where you are required to enter some details about how you want the VM to be setup before clicking "Launch Instance". In the "Launch Instances" pop-up frame choose the following settings:

   Server Name A human readable name for your convenience. e.g. "My NGS VM"

   Flavor The resources you want to allocate to this VM. I suggest a Medium sized VM (2 CPUs and 8 GBytes RAM). This will use all your personal allocation, but anything less will probably be insufficient. You could request a new allocation of resources if you want to instantiate a larger VM with more memory.

   Security Groups Select SSH.

3. Click the "Launch Instance" button

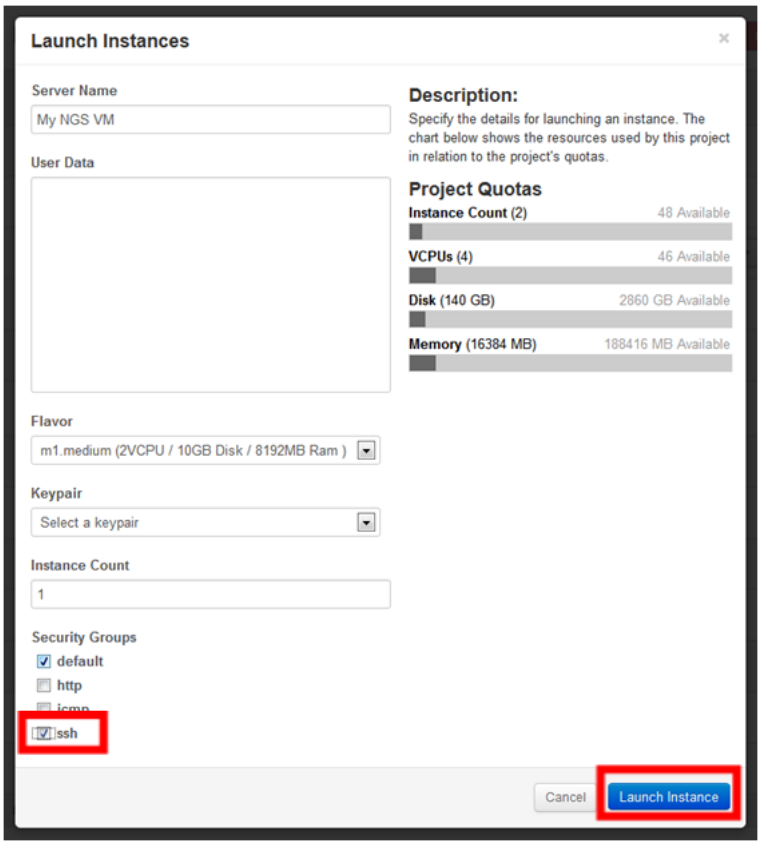

Figure 13:

4. You will be taken to the "Instances" page and you will see the "Status" and "Task" column for your new VM is "Building" and "Spawning". Once the "IP Address" cell is populated, take a note of it as you will need it for configuring the NX Client later on.

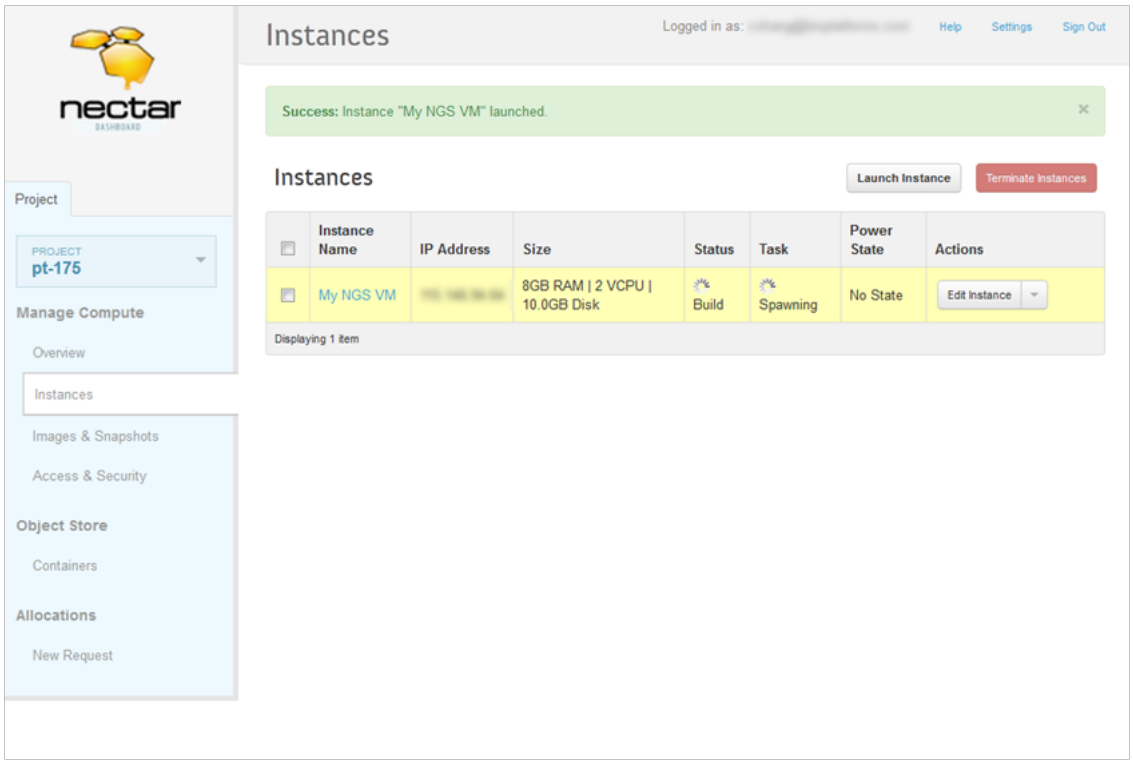

Figure 14:

5. Once the Status and Task for the VM change to "Active and "None" respectively, your VM is powered up and is configuring itself. Congratulations, you have now instantiated a Virtual Machine! If you try to connect to the VM too quickly, you might not be successful. The OS may still be configuring itself, so give it a few minutes to finish before continuing.

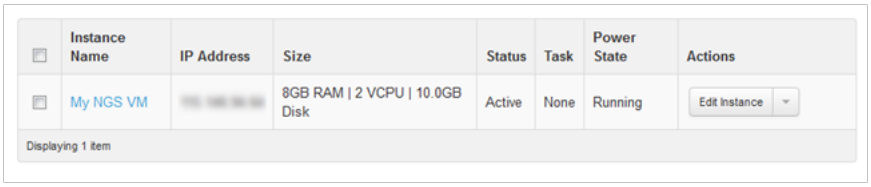

Figure 15:

VM Stuck Building and Spawning

Sometimes, the cloud experiences a "hiccup" and a newly instantiated VM will get stuck in the "Build" and "Spawning" state (step 3) for more than a few minutes. This can be rectified by terminating the instance and creating a new VM from scratch:

1. Selecting "Terminate Instance" under the "Edit Instance" dropdown box:

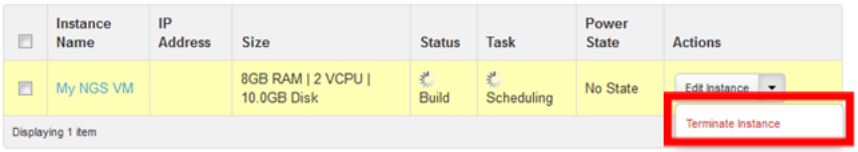

Figure 16:

2. Go back to step 1 of "Instantiating Your Own VM" and create the VM from scratch:

Remote Desktop with the NoMachine NX Client

During the workshop you were using the free NX client from NoMachine (http://www.nomachine.com/) to provide a remote desktop-like connection to VMs running on the NeCTAR Research Cloud. Therefore, we provide information on how to setup your local computer to connect to the VM you just instantiated in the steps above.

We assume that:

- You have administrator rights on your local computer for installing software.

NoMachine NX Client Installing

We show you instructions below for the MS Windows version of the NX Client, but procedures for other supported OSes (Linux, Mac OSX and Solaris) should be very similar.

1. Go to the NoMachine download page: http://www.nomachine.com/download.php
2. Click the download icon next to the NX Client for Windows:

Figure 17:

3. On the ”NX Client for Windows” page, click the ”Download package” button:

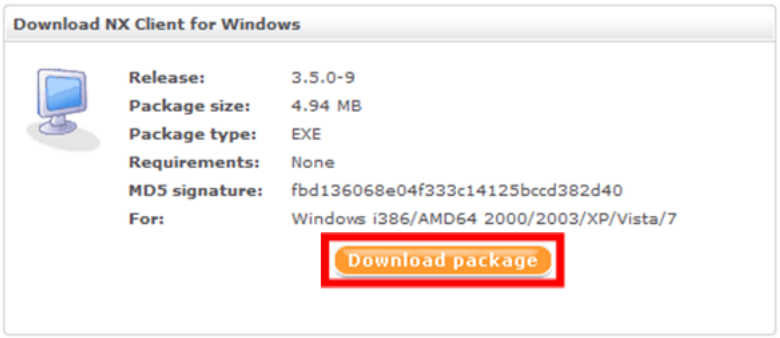

Figure 18:

4. Run the file you just downloaded (accepting defaults is fine)
5. Congratulations, you just installed the NoMachine NX Client!

NoMachine NX Client Configuration

Now we have the NoMachine NX Client installed, we need to configure a new NX ”session” which will point to the VM we instantiated in the NeCTAR Research Cloud.

We assume that:

- You know the IP address of the VM you want to remote desktop into.

1. Start the NX Connection Wizard and click ”Next” to advance to the ”Session” settings page.

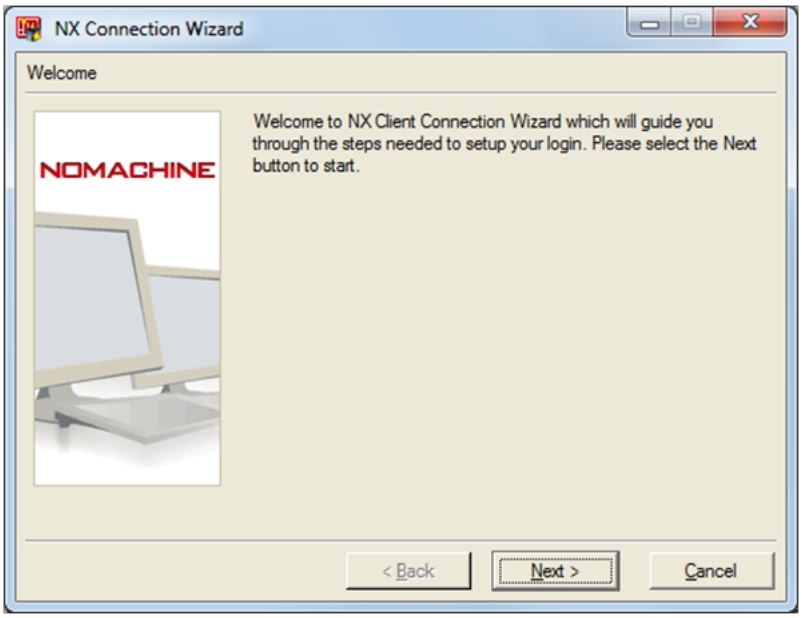

Figure 19:

2. On the "Sessions" settings page enter the following details:

   Session A memorable name so you know which VM this session is pointing at. You could use the same name you chose for the VM you instantiated earlier e.g. "NGS Training".

   Host Enter the IP address of the VM you instantiated on the NeCTAR Research Cloud.

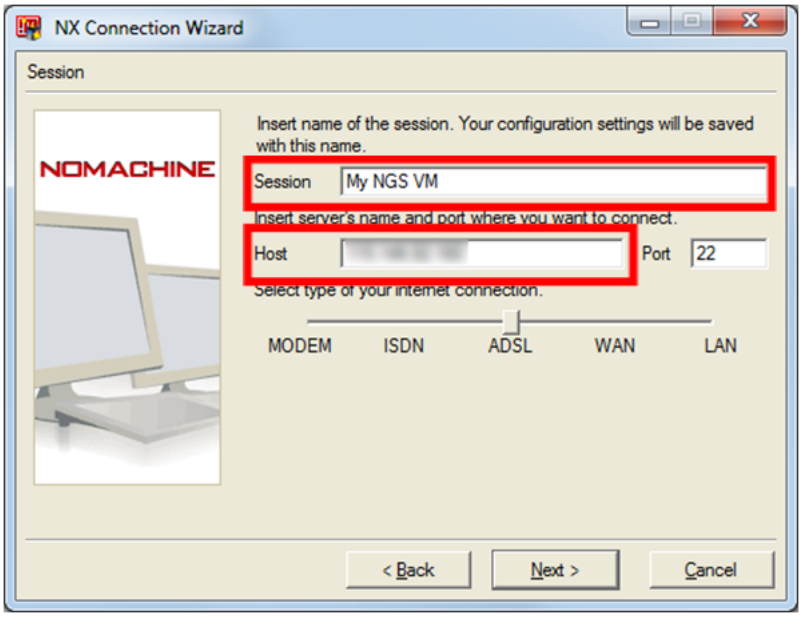

Figure 20:

3. Click "Next" to advance to the "Desktop" settings page. You should use the "Unix GNOME" setting.

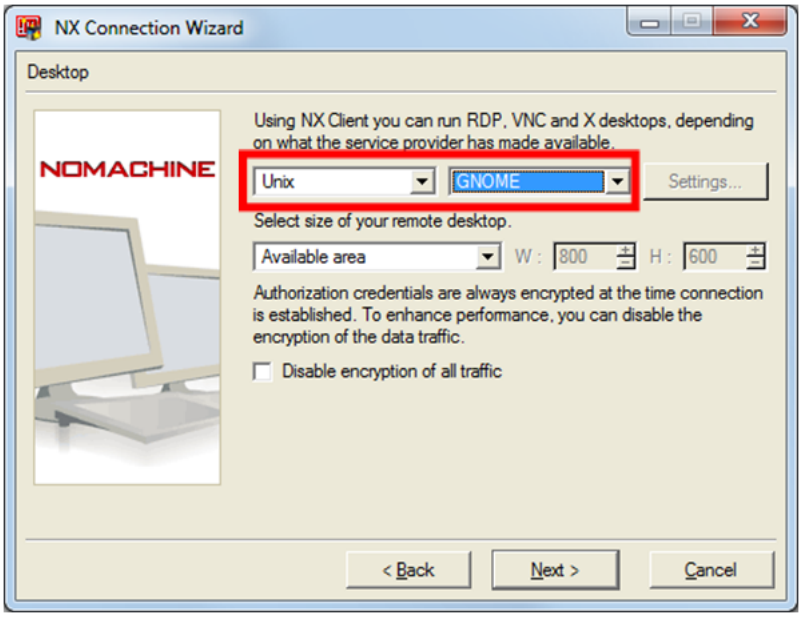

Figure 21:

4. Click "Next" and "Finish" to complete the wizard.

Connecting to a VM

If you just completed the NX Connection Wizard described above, the wizard should have opened the NX Client window. If not, run the "NX Client". You will be presented with a window like this:

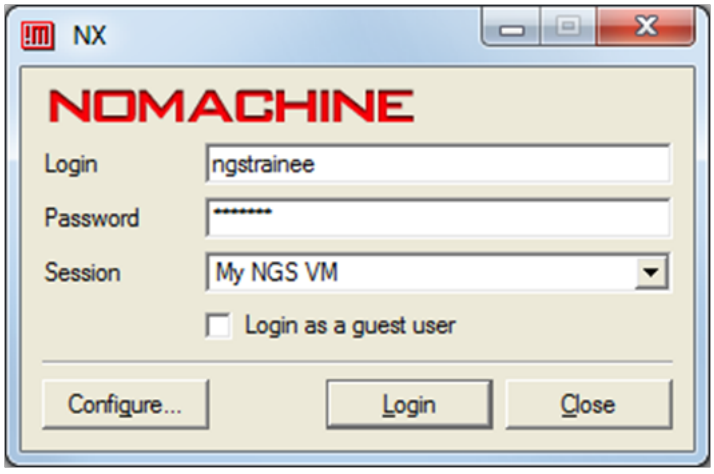

Figure 22:

The ”Login” and ”Password” boxes in the NX Client are for user accounts setup on the VM. By default our image, from which you instantiated your VM, has two preconfigured users:

Username	Password	Notes
ngstrainee	trainee	The user you used during the workshop
ubuntu	bioubuntu	Has “sudo” privileges for performing admin tasks

Figure 23:

Unless you know what you are doing, we suggest you use the `ngstrainee` user account details to initiate an NX connection to your VM. In less than a minute, you should see an NX Window showing the desktop of your VM:

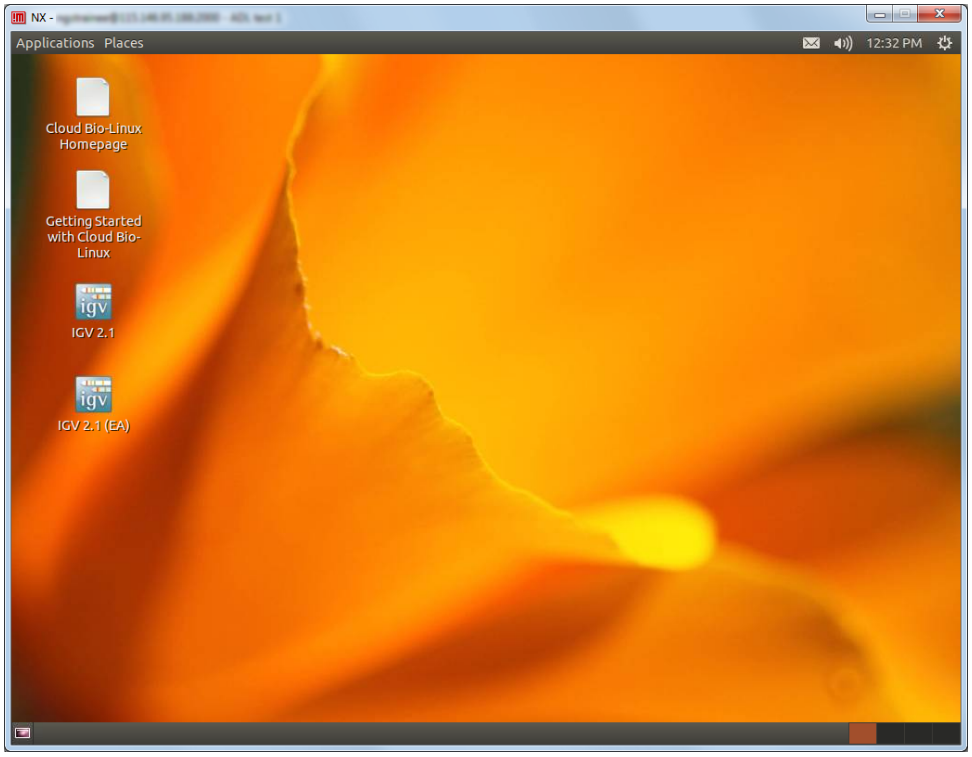

Figure 24:

NX Connection Failure

In the event that you don't get the NX Window with your VM's desktop displaying inside it. The most common errors are:

- You failed to select the "ssh" security group when instantiating the VM. You'll need to terminate the instance and create a new VM from scratch
- You failed to select "Unix GNOME" when you configured the NX Client session. You'll need to reconfigure the session using the NX Client
- Your institutions firewall blocks TCP port 22. You may need to request this port to be opened by your local network team or configure the NX client to use a proxy server.

Advanced Configuration

In the session configuration, you can configure the size of the NX Window in which the desktop of the VM is drawn:

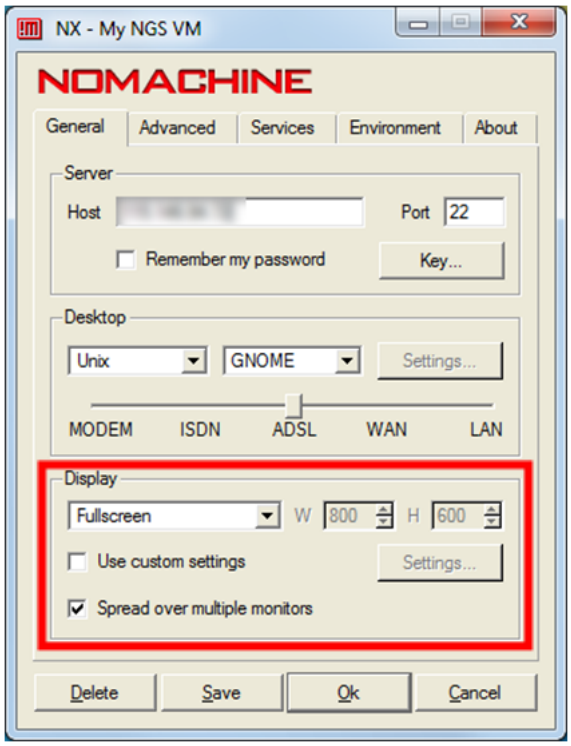

Figure 25:

This can be useful if you want to:

- Have the NX Window occupy the entire screen, without window decorations. This is often desirable if you wish to ”hide” the host OS from the person sitting at the computer running the NX Client.
- Have the NX Window spread over multiple monitors.

Access to Workshop Documents

This document has been written in LaTeX and deposited in a public github repository (https://github.com/nathanhaigh/ngs_workshop). The documentation has been released under a Creative Commons Attribution 3.0 Unported License (see the Licence page at the beginning of this handout).

For convienience, you can access up-to-date PDF versions of the LaTeX documents at:

Trainee Handout
https://github.com/downloads/nathanhaigh/ngs_workshop/trainee_handout_latest.pdf

Trainer Handout
https://github.com/downloads/nathanhaigh/ngs_workshop/trainer_handout_latest.pdf

Access to Workshop Data

Once you have created a VM from our image file, either locally using VirtualBox or on the NeCTAR Research Cloud, you can configure the system with the workshop documents and data. This way you can revisit and work through this workshop in your own time.

In order to do this, we have provided you with access to a shell script which should be executed on your NGS Training VM by the `ubuntu` user. This pulls approx. 3.3 GBytes of data from the NeCTAR Cloud storage and configures the system for running this workshop:

```
# As the ubuntu user run the following commands:
cd /tmp
wget https://github.com/nathanhaigh/ngs_workshop/raw/master/\
workshop_deployment/NGS_workshop_deployment.sh
bash NGS_workshop_deployment.sh
```

While you're at it, you may also like to change the timezone of your VM to match that of your own. To do this simply run the following commands as the `ubuntu` user:

```
TZ="Australia/Adelaide"
echo "$TZ" | sudo tee /etc/timezone
sudo dpkg-reconfigure --frontend noninteractive tzdata
```

For further information about what this script does and possible command line arguments, see the script's help:

```
bash NGS_workshop_deployment.sh -h
```

For further information about setting up the VM for the workshop, please see:

https://github.com/nathanhaigh/ngs_workshop/blob/master/workshop_deployment/README.md

Space for Personal Notes or Feedback

Space for Personal Notes or Feedback

Space for Personal Notes or Feedback