NORWEGIAN SEQUENCING CENTRE

Norwegian Sequencing Centre
de novo assembly course October 26-27th 2011

Course schedule

DAY 1 - Wednesday October 26th
09:00-09:15 Introduction: Who is how (All)
09:15-10:00 Lecture 1: Principles and problems of *de novo* genome assembly (Lex)
10:00-10:30 Lecture 2: Planning a *de novo* assembly project (Nick)

10:30-10:45 Coffee/tea break

10:45-12:00 Practical 1: Understanding reads, preparing and QC of sequence data (Lex)
- logging in, ftp, data and working folders, how to run commands
- fastQC and Prinseq
- trimming
- introduction to the E coli O104 outbreak data available

12:00-13:00 Lunch break (cantina of the new informatics building across the street)

13:00-14:30 Practical 2 - *de novo* assembly of 454 reads with Newbler (Lex)
- setting up your first E coli O104 (strain 280) assembly
- coverage
- following newbler progress
- newbler, newbler options

14:30-15:45 Practical 3 - Newbler output (Lex)
- introduction to basic assembly metrics
- evaluation of Newbler output files
- understanding scaffolds
- assembly viewing in Tablet

15:45-16:00 Coffee/tea break

16:00-17:00 Practical 4: *de novo* assembly of Illumina reads - Part 1 (Nick)
- Importance of k-mers
- Setting up a velvet assembly

NORWEGIAN SEQUENCING CENTRE

DAY 2 - Thursday October 27th

09:00-09:30 Practical 5: hybrid *de novo* assembly (Lex)

- Using newbler
- Start the assembly (it takes a long time)

09:30-10:45 Practical 6: *de novo* assembly of Illumina reads - Part 2

- optimizing k-mer lengths
- optimizing other velvet parameters
- adding paired end and mate-pair information

10:45-11:00 Coffee/tea break

11:00-12:00 Practical 6: *de novo* assembly of Illumina reads - Part 2 (continued)

12:00-13:00 Lunch break (cantina of the new informatics building across the street)

13:00-14:00 Practical 5 and 6: evaluation of the results

14:00-15:00 Practical 7: What can our assemblies tell us about *E. coli* O104? (Nick)

- MAUVE to visually check for mis-assembly
- align to reference genome, find difference(s)

15:00-15:45 Discussion: what have we learnt

- QC and filtering
- 454 only assembly
- Illumina only assembly
- hybrid assembly
- how to determine which assembly is ‘best’

15:45-16:00 Coffee/tea break

16:00-16:30 Lecture 3: what’s next (Lex)

- other popular assembly programs
- resources for annotation
- challenges for *de novo* assembly of large (eukaryotic) genomes

16:30-17:00 Discussion: rounding up

- short evaluation of the course
- Q&A

NORWEGIAN SEQUENCING CENTRE

Practicals hand-out

Conventions in this document

This is normal text

This is text describing a unix command, e.g. `grep` - the command will then be in italics.

```
This is a command you need to enter on the command line
```

```
This command has one word in bold that you need to change
```

For example, this might be the name of the folder that will contain the output of the command

Below is a table where you can fill in the answers to the question(s)

Question 1	your answer here
Question 2	your answer here

BEFORE practical 1: preparing

Logging in

First, make sure you know whether you will be using the cod1 or cod2 server.
Open your terminal program and log in:

```
ssh your_username@cod1.uio.no
```

OR

```
ssh your_username@cod2.uio.no
```

Enjoy the ASCII art (courtesy of the IT people)...

ftp access

Open your ftp client (fetch, filezilla, winscp, …) and log in to the same server using the same username as described above. Select the folder `/work`

Working area
The area to do your 'work' and save your files is here:

```
/work
```

To move into this area, write

```
cd /work
```

`cd` stands for change directory. Make a folder for your results and analyses, using the command `mkdir` ('make a directory')

```
mkdir yourname
```

move into that folder

```
cd yourname
```

Running commands/programs

There are three ways to run a command/program:

1) Many commands come 'for free', they are accessible right after you have logged in. Built-in system commands like `cd mv ls` etc are accessed in this way.

2) We have installed a few programs and commands especially for the course in this folder:

```
/data/bin
```

This means that for these to run, you will have to type each time:

```
/data/bin/command_name
```

Luckily, there is a trick allowing you to skip the '/data/bin/' part. After logging in to the server, just *once*, type (exactly as typed here):

```
export PATH=/data/bin/:$PATH
```

Test if this works by typing

```
jellyfish --version
```

You should now see

```
jellyfish 1.1.2
```

If you don't see this, ask one of the teachers!
Unix geeks might want to put this command into a (new?) file called:

```
~/.bash_login
```

3) On the Univ. of Oslo titan cluster (and the cod1 and cod2 servers), we use public-domain Unix package called "Modules" as a tool to allow access to programs. We will for example be able to run the newbler program by typing:

```
module load newbler
```

We will start using this later today.

Practical 1 - Understanding reads, preparing and QC of sequence data

Learning points:

- Understand read file formats
- How to judge a QC report
- How to filter/trim Illumina reads

Quality Control of 454 reads using PRINSEQ

If you want to have a look at the demo data, go to the prinseq website:

http://edwards.sdsu.edu/prinseq_beta/#

choose 'Access Data' and enter the following data ID: 31333139313939303238

A peak into the fastq files

The illumina reads for sample 280 are located in this folder:

```
/data/illumina/
```

Fastq files are very big. In order to be able to view them in a 'page-by-page' way, we will use the `less` command:

```
less /data/illumina/Sample280_1.fastq
```

This file contains read 1 of a MiSeq run for the sample. Use the space bar to browse through the file. Use '`q`' to go out of the `less` program. Make sure you recognize the fastq format, if needed use the slides from the presentation.

Repeat this for the read 2 file:

```
less /data/illumina/Sample280_2.fastq
```

Do you see whether the reads in the same order in both files?

Quality control of Illumina reads

We will be using a program called **FastQC**. The program is available with a graphical user interface, or as a command-line only version. We will use the latter one. It takes a single fastq file (the file can be compressed) as input, and produces a web page with the results of a number of analyses.

The program, as many of the other programs/scripts we are going to be using, is located in /data/bin.

For this practical, we will be using two options:

`--noextract`

This makes the program only give a compressed (zipped) file

`-o` **`foldername`**

This tells the program where to put the output file(s). If you leave out this option, the output is placed in the same folder as where the input file is located.

To run it on the first MiSeq file, run the command below; **`yourname`** should be the name you used for your folder in /work. Note that the command should be written on a *single line*.

```
fastqc --noextract -o /work/yourname
   /data/illumina/Sample280_1.fastq
```

The program will tell you how far it has come, and should finish in a minute or two. Check that it finished without error messages.

In the folder you specified after `-o`, you should now see a new file called Sample280_1_fastqc.zip. Using your ftp program, locate this file and copy it over to your laptop (local hard disk). Next, you will need to unzip it, usually this can be done by double clicking on the file. You should now see two folders and three files:

`Icons` → folder
`Images` → folder
`fastqc_data.txt` → file
`fastqc_report.html` → file
`summary.txt` → file

Open the file called fastqc_report.html in a web browser (Firefox, Safari, Internet Explorer, Google Chrome, …), use the 'open file' option in the menu, or double click on it. Study the results.
The plot called "Per base sequence quality" shows an overview of the range of quality scores across all based at each position in the fastq file. The y-axis shows quality scores and the x-axis shows the read position. For each read position, a boxplot is used to show the

distribution of quality scores for all reads. The yellow boxes represent quality scores within the inter-quartile range (25% - 75%). The upper and lower whiskers represent 10% and 90% point. The central red line shows the median of the quality values and the blue line shows the mean of the quality values.

A rule of thumb is that a quality score of 30 indicates a 1 in 1000 probability of error and a quality score of 20 indicates a 1 in 100 probability of error (see the wikipedia page on the fastq format at http://en.wikipedia.org/wiki/Fastq). The higher the score the better the base call. You will see from the plots that the quality of the base calling deteriorates along the read (as is always the case with Illumina sequencing). The NSC has as a minimum requirement for Per Base Sequence Quality that the first 36 bases should have a median and mean quality score over 20.

Now, answer these questions:

Question	**Your answer**
How many reads were there in total in the `Sample280_1.fastq` file ?	
How many bases were there in total in the file?	
Which part(s) of the reads would you say are of low quality?	
Would the run have passed the NSCs minimum requirement for Per Base Sequence Quality?	

Repeat the fastqc analysis for the file

```
/data/illumina/Sample280_2.fastq
```

Copy the zipped report file created by fastqc to your laptop again, unzip it and open it in your webbrowser.

Question	Your answer
How many bases were there in total in the `Sample280_2.fastq` file? Is this surprising?	
Are there part(s) of the reads that have a lower quality compared to the `Sample280_1.fastq` file?	
Would the run have passed the NSCs minimum requirement for Per Base Sequence Quality?	

NB. You can get more information about the use of the fastqc program by writing

```
fastqc -h
```

Filtering and trimming

We want to do the following with the raw reads:

- reads may contain adaptor sequences left from library preparation, these we want to remove
- remove bad-quality reads
- remove reads that are likely to be artifacts
- remove the end of reads that are given the quality score 2 (encoded in the fastq file with a 'B'), indicating these bases should not be used for further analysis. See http://en.wikipedia.org/wiki/Fastq#Encoding

To see these trailing 'B's, do the following.

Type

```
less /data/illumina/Sample280_1.fastq
```

Now, type the forward slash '`/`'.This allows you to do text search in the file you are viewing. Type

```
BBBBBB
```

The first string of six or more B's is indicated. Type '`N`' to see more of these. Where are these string of B's located? Type `q` to get out of `less`.

For the actual filtering, we are going to use parts of the **fastx toolkit** for filtering. The following commands will be used, with the options we will give to them:

Program	**Options**	**Explanation**
fastx_clipper		Removal of adaptor sequences
	-v	tells the program to report on what it is doing
	-l 20	discard sequences shorter than 20 nucleotides
	-M 15	require minimum adapter alignment length of 15 bases
	-a	the sequence following this option is the adaptor to remove
fastq_quality_filter		filters sequences based on quality
	-q 30	minimum quality score to keep is 30
	-p 50	minimum 50% of bases that must have quality 30 or better
fastx_artifacts_filter		removes reads that likely are artifacts

In addition, we will use a script written by one of us:

Program	**Options**	**Explanation**
fastq_trim_trailing_Bs.pl	none	Removal of sequences at the end with quality score 2 ('B')

Piping, saving to file

Those who are familiar with unix can skip this section.

One could run each of these programs individually and save the results in intermediate files, to be used as input for the next step. However, it is much more practical to have the output of one command used directly by the next (without saving to disk). This is possible using the unix 'pipe'. The pipe symbol '`|`' allows for sending the output from one command to the next, thereby allowing for stringing commands together. For example, using the `cat` command that list the contents of the file(s) mentioned after it, we can send (pipe) it's output to the `less` command, which allows for page-by-page viewing:

```
cat file.txt | less
```

Note that a shorter way to do this would be

```
less file.txt
```

Using the pipe symbol, we can be creating very long command lines, involving many commands and pipes between them to do the filtering:

```
command1 -option1 -option2 | command2 -option1 | command3
    -option1 -option2 | command4 -option1 -option2 | command5
    -option1 -option2
```

Another trick we are going to us is to split up this long command over several lines to make it much more readable. This can be done in unix by typing a space, the backslash symbol '\' and pressing the return key:

```
command1 -option1 -option2 | \
command2 -option1 -option2 | \
command3 -option1 -option2 | \
command4 -option1 -option2 | \
command5 -option1 -option2
```

One final thing to know is that many programs send their output to the screen (technically, to 'standard output') for you to read. You can save this output to a file instead by writing the '>' symbol (called 'redirect') and a filename, e.g.

```
command 1 | command 2 >output.txt
```

Tab completion

One of the most useful short-cuts to know when using command-line is *tab completion*. When typing in the name of a command or file, you can hit the tab key part-way through to try and get the system to finish it off for you. When it is unambiguous what you are trying to do, the system will auto complete for you. Otherwise it will partially complete. You can hit tab twice to see a list of all the available options. Another useful trick is the up-arrow, which lets you find commands you have previously entered and re-run them. This goes in hand with the `history` command which shows you all of the commands you have entered recently.

Trimming and filtering - the command

This is the command we will use:

```
cat /data/illumina/Sample280_1.fastq | \
fastx_clipper -v -l 20 -M 15 -a GATCGGAAGAGCGGTTCAGCAGGAATGCCGAG | \
fastq_quality_filter -v -q 30 -p 50 | \
fastx_artifacts_filter  -v | \
fastq_trim_trailing_Bs.pl >Sample280_1_filtered.fastq
```

This process will take around 5 minutes.

The fastx_toolkit commands will report among other things the number of reads for each step.

Question	**Your answer**
How many reads were removed from `Sample280_1.fastq` during filtering?	
What was the biggest cause for removal?	

Now, run the same filtering on the file `Sample280_2.fastq`

Question	**Your answer**
How many reads were removed from `Sample280_2.fastq` during filtering?	
What was the biggest cause for removal?	

NB each of the fastx_toolkit commands has a help text available with the `-h` option, for example

```
fastx_clipper -h
```

Redoing the fastqc on the filtered reads

Run fastqc on the filtered files (as before), and compare the results with the those from the unfiltered reads.

Question	**Your answer**
When comparing the “Per base sequence quality” graphs, what is the biggest difference between the filtered and unfiltered data sets?	
Explain the difference in the “Sequence length distribution” plots between the filtered and unfiltered data sets?	
What did we achieve by filtering this way?	

Practical 2 - *de novo* assembly of 454 reads with Newbler

Learning points:

- How to start a newbler assembly
- Basic understanding of parameters

Where is what

All 454 files are located in the folder `/data/454/`.

You can 'have a look' at what other files are there by typing

`ls /data/454` [ls stands for list]

A short description of the files:

454_sg_all.sff	GS FLX+ shotgun reads	218 000 reads, 155 Mbp
454_sg_one_3rd.sff	one third of the reads above	73 000 reads, 52 Mbp
454_gsflx+_two_thirds.sff	two thirds of the reads above	146 000 reads, 103 Mbp
454_sg_two_3rd.sff	GS FLX Titanium 8kb PE reads	74 000 reads, 28 Mbp
G310ZZ001.sff	GS Junior shotgun run 1	136 000 reads, 63 Mbp
G32DMQC01.sff	GS Junior shotgun run 2	138 000 reads, 63 Mbp

As of today, GS Junior reads have the same length as GS FLX Titanium, peak length around 500 bases. GS FLX+ reads peak at 800 bases.

A first assembly

We will perform an assembly using newbler and the 454 GS FLX+ shotgun reads, together with 8kb Paired End (better called Mate Pair) reads. These files will be used for this first assembly:

```
/data/454/454_sg_all.sff
/data/454/454_8kb_pe.sff
```

Remember the 'module' package (see the beginning of this document)? By typing the command below, your system is set up to run newbler version 2.6:

```
module load newbler
```

You can test whether you now have access to newbler by typing:

```
runAssembly --help
```

You should now see something like this:

```
Usage:  runAssembly [-o projdir] [-nrm] [-p (sfffile | [regionlist:]analysisDir)
]... (sfffile | [regionlist:]analysisDir)...


(The full documentation for "runAssembly" command can be found in the user manua
l).
```

If you get and error message, contact one of the teachers!

To run an assembly, use the `runAssembly` command described below. Please note:

- `runAssembly` is a program that sets up a newbler assembly and starts newbler
- the command should be written on one line
- replace **`projectname`** (see below) with something of your own choosing; this will be the name of the folder the assembly will appear in ('`-o`' stands for output folder)
- the rest of the command is the path to one or more files to be used as input; in this case, we use two files
- although one can use multiple cpus (through the '`-cpu`' flag) to speed up the assembly, do not to this today (otherwise we don't have enough cpus for everybody...).
- *do not close the terminal window* until the assembly is done, as you will cancel it
- the whole process will, for this dataset, take just over 45 minutes

Use this command

```
runAssembly -o projectname /data/454/454_sg_all.sff
    /data/454/454_8kb_pe.sff
```

The output that newbler sends to the screen during assembly is explained in Lex's blog at http://contig.wordpress.com/2010/02/09/how-newbler-works/
It can be found in the file called 454NewblerProgress.txt after the assembly is done.

While the assembly is running, follow the output to the screen and answer the following questions:

Question	**Your answer**
How many bases were there in total available for the assembly?	
How much coverage is that (taking a total genome size of 5.43 Mbp)?	

→ We'll discuss the answers in plenum, as well as some of the options you can use with newbler.

Practical 3 - Newbler output

Learning points

- How to locate an obtain relevant metrics
- How to find contigs and scaffold
- How to inspect alignments

Assembly metrics

Have a look at the 454NewblerMetrics.txt file.

```
cd projectname
```
(use the projectname you chose)

This will move you into the assembly folder

```
cat 454NewblerMetrics.txt
```

(Remember that `cat` command will list the contents of the file(s) mentioned after it).
For more details of what the different parts of this file mean, check
http://contig.wordpress.com/2010/03/11/newbler-output-i-the-454newblermetrics-txt-file.
When newbler reports the number of reads (or bases) for a file, it writes this as:

numberOfReads = xxx[raw], yyy[after filtering];

Question	**Your answer**
How many reads did newbler remove during filtration for both input files?	
How much coverage was left after filtering?	
What did newbler determine to be the coverage? And the genome size? Look in the section labeled 'Alignment depths' further down in the file.	
What did newbler determine to be the insert size ('average pair distance') of the 8kb library?	
What do you think 'Q40PlusBases" means?	

To have a look at the sequences newbler produced, check out the scaffolds and contig files. 'LargeContigs' are those that are 500 bp and more, 'AllContigs' those from 100 bp (including all the 'Large' ones). As these files are very large, using `cat` will send a lot of text to the screen, which is not very practical. Instead, we will use the `less` command to view them page-by-page:

```
less 454Scaffolds.fna
```

Use the space bar to browse through the file. Do this until you find the first gap. Use '`q`' to go out of the `less` program.

To have a quick look at the lengths of the different scaffolds, write:

```
grep 'scaffold' 454Scaffolds.fna
```

`grep` is a really useful command. It will show the lines from a file that match a certain pattern. The pattern can, but does not have to be, in between 'quotation marks' (single or double). CAREFUL, if you want to select lines beginning with the '>' symbol, you HAVE to use quotation marks:

```
grep '>' 454Scaffolds.fna
```

You will see that the sequences are no longer shown, just the fasta headers (sequence descriptors)

You can also do this for the contig files. However, you might want to 'pipe' to output to the `less` command for easy viewing:

```
grep contig 454LargeContigs.fna | less
```

The pipe symbol '`|`' allows for sending the output from one command to the next, thereby allowing for stringing commands together.

Note that the largest scaffolds and contigs appear first.

Understanding N50

What is the N50 of an assembly with 7 contigs of sizes: 20, 9, 9, 6, 3, 2 and 1 long?

The assemblathon metrics

The assemblathon (www.assemblathon.org) used a script to obtain standardized metrics for the assemblies that were submitted. Here we use (a slightly modified version of) this script. It takes one (scaffold) file as input and breaks the sequences into contigs when there are 20 or more N's.

To make this script work, you need to start another module:

```
module load perl
```

This will give the script access to a perl module called FALite (and a suite of other perl modules, including bio-perl).

Write

```
assemblathon_stats.pl 454Scaffolds.fna
```

OR, save the output to a file with

```
assemblathon_stats.pl 454Scaffolds.fna > metrics.txt
```

Here, '>' (redirect) symbol is again used to 'redirect' what is written to the screen to a file.

Study the output of this script (use `cat` or `less`). We will use it to compare different assemblies.

Assembly viewing

For your first assembly, transfer the 454Contigs.ace file to your harddisk, e.g. using an ftp client. The ace file represents the assembly with contigs, and all read alignments (but not the scaffolding information), which is why it usually is quite large. We can view the alignments using the program Tablet. More information on the ace file can be found at http://bozeman.mbt.washington.edu/consed/distributions/README.19.0.txt.

Open the 454Contigs.ace file in Tablet (choose 'Open Assembly'). Sort the contigs on length (shortest on top), select contig00169

- can you spot a homopolymer difference between one or more reads and the consensus? Look for both undercalls (one base too few) and overcalls (one too many)
- play around with the different elements of this program (slider bars etc), also have a look at the largest scaffold

More assemblies

If you have time, you can run another assembly from the table below. Please note on the board which assembly you will do. We will compare the results tomorrow. *Unless otherwise noted*, use default newbler parameters.

For those assemblies using IonTorrent data, see below the table. NOTE: perhaps you might want to run fastqc on these runs?

Number	**Description**	Files
1	assembling the shotgun data alone	454_sg_all.sff
2	the (shorter) Titanium GS Junior reads	G310ZZ001.sff G32DMQC01.sff
3	the (shorter) Titanium GS Junior reads with the 8kb paired ends	G310ZZ001.sff G32DMQC01.sff 454_8kb_pe.sff
4	lower shotgun coverage	454_sg_one_3rd.sff 454_8kb_pe.sff
5	intermediate shotgun coverage	454_sg_two_3rd.sff 454_8kb_pe.sff
6	scaffold gap filling, add the `-scaffold` flag	454_sg_all.sff 454_8kb_pe.sff
7	more stringent alignment, use `-ml 80 -mi 95`	454_sg_all.sff 454_8kb_pe.sff
8	less stringent alignment, use `-ml 30 -mi 80`	454_sg_all.sff 454_8kb_pe.sff
9	IonTorrent 314 chip	Run1_4.sff BH1-4_5.sff Run3_6.sff Run4_7.sff
10	IonTorrent 314 chip with 454 paired ends	Run1_4.sff BH1-4_5.sff Run3_6.sff Run4_7.sff 454_8kb_pe.sff

11	IonTorrent 316 chip I	BH1-7_9.sff and fastq
12	IonTorrent 316 chip II	BH1-8_10.sff and fastq
13	You are lucky and get to try something not yet mentioned	Pick your file(s)

IonTorrent reads

We have the IonTorrent run data from the same 280 strain. The files are in this folder:

`/data/iontorrent/`

All runs are shotgun reads, with a peak length at 122 bases

314 chip data

Run1_4.sff and fastq	153 000 reads, 12.9 Mbp
BH1-4_5.sff and fastq	51 000 reads, 4.1 Mbp
Run3_6.sff and fastq	208 000 reads, 17.3 Mbp
Run4_7.sff and fastq	193 000 reads, 16.1 Mbp

316 chip data

BH1-7_9.sff and fastq	2.25 million reads, 251 Mbp
BH1-8_10.sff and fastq	1.95 million reads, 209 Mbp

Practical 4: *de novo* assembly of Illumina reads - Part 1

Learning points

- understand k-mers and how to estimate genome size from short-read data
- understand how to estimate read error, sequence diversity, repeats from k-mers
- understand how to run Velvet
- understand the concept of k-mer coverage
- understand the effect of k-mer settings on assemblies
- understand how to use paired-end information in Velvet
- understand the significance of setting expected coverage values

Calculate genome size using k-mer counting

We can count the number of unique k-mers in a sample to give us an estimate of genome size. For this we will use the program Jellyfish:

Program	**Options**	**Explanation**
jellyfish count		Count unique and total k-mers from a FASTA or FASTQ file
	-m 22	Use a *k*-mer size of 22
t	-o	The output file prefix
	-s 10000000	The hash table size (influences memory usage)
	-c	Length of counting field in bits (don't need to change)
jellyfish stats		Provide statistics on total and distinct k-mers
	-L	Lower frequency bound to count from
	-U	Upper frequency bound to count from
jellyfish histo		Produce a histogram from k-mer counting data

Perform the following:

```
jellyfish count -m 22 -o your_prefix -c 3 \
     -s 10000000 \
     /data/illumina/Sample280_interleaved.fastq
```

This will produce 2 or more files starting with your_prefix and then an integer. Make sure you supply all the generated files to the next step:

```
jellyfish merge your_prefix_0 your_prefix_1
     your_prefix_2 your_prefix_3 \
     -o your_database.jf
```

This will merge together your files.

Now we can generate some k-mer statistics from the database.

```
jellyfish stats your_database.jf
```

How many unique (singleton) k-mers are there? How many distinct (different) k-mers are there? What is the total number of k-mers in the database?

To determine genome size accurately, we must determine which k-mers are real and which represent sequencing error or variation. This is best done by plotting the k-mer frequency. We can do this easily through jellyfish:

```
jellyfish histo your_database.jf > histogram.txt
```

There are two ways of using k-mers to estimate genome size.

Ignoring the singletons, where is the modal coverage? Where do you think the boundary of erroneous k-mers is? Write down the numbers and consider plotting the histogram (e.g. in Excel).

What is the modal coverage (excluding singletons?)	
Where is the boundary between erroneous k-mers?	
Where is the upper bound of non-unique parts of the genome?	

Estimating genome size

Now we can estimate the genome size. Re-run the count, setting a lower bound as follows:

```
jellyfish stats -L lower_bound -U upper_bound your_database.jf
```

Two simple methods for genome size estimation

The first is simple; you simply divide the number of distinct (different) k-mers in your dataset by 2.

The second way is to divide the total number of good k-mers by 2 and then by the mean genome coverage. As an approximation you can use the modal coverage for the non-unique region of the genome.

Assembling short-reads with Velvet

We will use Velvet to assemble Illumina reads.

To initialise your environment:

```
module load velvet
module load python
```

We will assemble the *E. coli* 280 strain which was sequenced on an Illumina MiSeq. The sequencing centre read 150 bases from each direction. We know from the sequencing centre that the fragment size they chose was approximately 450bp.

Although we can identify pairs of reads by looking at their identifiers (they end in either `/1` or `/2`), Velvet requires paired-end data to be *interleaved*, e.g. read 2 should always *follow* read 1.

You should have two files from the filtering practical named `Sample280_1_filtered.fastq` and `Sample280_2_filtered.fastq`. We will use a simple Python script to interleave the two files, ensuring that the proper pairing is preserved. "Orphan" reads which have no pair will be sent to a separate file which we will not use in this practical.

```
pair_up_reads.py Sample280_1_filtered.fastq \
     Sample280_2_filtered.fastq \
     Sample280_1_good.fastq \
     Sample280_2_good.fastq \
     Sample280_orphans.fastq
```

Examine this file (Compare with `/data/illumina/Sample280_1.fastq` and `/data/illumina/Sample280_2.fastq`) and check you understand the format.

How many orphan sequences were produced?

Now interleave the files:

```
shuffleSequences_fastq.pl \
     Sample280_1_good.fastq Sample280_2_good.fastq \
     Sample280_interleaved.fastq
```

If you have problems with these steps, we have provided a Velvet-compatible file for you here:

```
/data/illumina/Sample280_interleaved.fastq
```

Building the Velvet Index File

Velvet requires an index file to be built before the assembly takes place. Just like in the previous practical we must choose a *k*-mer value. Longer *k*-mers result in a more stringent assembly, at the expense of coverage. There is no definitive value of *k* for any given project. However, there are several rules - *k* must be less than the read length and it should be an odd number.

Firstly we are going to run Velvet in single-end mode, ignoring the pairing information. Later on we will incorporate this information.

Pick a value of *k* between 21 and 99 as a starting point and run `velveth` to build the hash index.

```
velveth my_assembly_directory value_of_k -short -fastq \
     Sample280_interleaved.fastq
```

Now look in **my_assembly_directory**, you should see the following files:

```
Log
Roadmaps
Sequences
```

Have a look at the file `Log` - this is a useful reminder of what commands you typed to get this assembly result, useful for reproducing results later on.

Now we will run the assembly:

```
velvetg my_assembly_directory
```

Look again at my_assembly_directory, you should see the following extra files;

```
contigs.fa
Graph
LastGraph
PreGraph
stats.txt
```

The important files are:

`contigs.fa` - the assembly itself
`Graph` - a textual representation of the contig graph
`stats.txt` - a file containing statistics on each contig

What *k*-mer did you use?	
What is the N50 of your assembly?	
What is the largest contig size?	
What is the total assembly size?	