Tools for Genome Analysis on Bio-Linux 7
A Taster

To demonstrate some of the capabilities of Bio-Linux, this demonstration takes you through a very simple assembly of some reads from a mitochondrial genome. This is in no way supposed to be a tutorial on genome assembly, but rather a way to see various Bio-Linux tools in action on a small dataset.

Setup

Open up the ***Bio-Linux Documentation*** icon on the desktop, then the ***Introductory Tutorial*** folder. You should see several tar files. Select ***assembly_taster.tar.xz*** and right click it. Select ***Extract To...*** from the pop-up menu. Extract to your home directory, which on the Live USB system is listed as ***live*** in the list on the left.

Open a terminal, then change into the new directory and list the files:

```
$ cd assembly_taster
# -lh options to ls show human-readable file size
$ ls -lh
```

Note: In all the commands given in this tutorial, $ represents your terminal prompt. This is a common convention. The real prompt will be something like "live@biolinux[live]". Lines beginning with # are comments and not to be typed.

To get a quick look at the input data, you can view it in the **less** text file viewer:

```
$ less mt_reads.fastq
```

Press Q to return to the terminal.

Make a directory to store your results:

```
$ mkdir results
```

Quality Checking

Firstly, in receiving a set of sequence data it is paramount to assess the quality of the dataset. A useful tool is **fastQC** which gives a quick graphical overview of the dataset.

```
# Run FastQC on the dataset
$ fastqc -o results mt_reads.fastq

# Open the HTML report file.
# The ampersand (&) will put the process in the background so you can still use the terminal
$ firefox results/mt_reads_fastqc/fastqc_report.html &
```

Split Barcodes

The sequencing data may be barcoded, depending on the experimental set up. Here, two mitochondria have been sequenced together, with differing 10bp barcodes at the 5' end. This allows us to split the data into two sets whilst only performing one sequencing run. Here we use a standard script from the the fastx toolkit (http://hannonlab.cshl.edu/fastx_toolkit/index.html)

```
# fastx splitter splits mt_reads.fastq by barcode.
# --bol indicates that the barcodes are at the 5' end.
# Note the following command should be typed on a single line:
$ fastx_barcode_splitter.pl <mt_reads.fastq --bcfile mt_barcodes.txt
      --bol --suffix .fastq --prefix results/
```

There are now two .fastq files in the results directory; one for each barcode. There is also an unmatched.fasta file which should be empty. We will be focusing on the first mitochondrion, ie. the one now in results/mt1.fastq.

Clean Up

To remove artefacts and improve the assembly we will do two steps:

Trim barcodes

This removes the barcode sequences from the beginning of each read. The -Q33 is required due to differences in sanger and illumina encoding

```
$ cd results
$ fastx_trimmer -i mt1.fastq -f 8 -o trimmed_mt1.fastq -Q33
```

Quality Filter

Removing low quality sequences increases the accuracy of the assembly.

```
$ fastq_quality_filter -i trimmed_mt1.fastq -q 25 -p 80
      -o qual_trim_mt1.fastq -Q33 -v
```

Or you could have run both commands in one shot, combined as a pipeline.

```
$ fastx_trimmer -i mt2.fastq -f 8 -Q33 |
      fastq_quality_filter -q 25 -p 80 -Q33 -o qual_trim_mt2.fastq
```

Assembly With Velvet

Velvet (https://www.ebi.ac.uk/~zerbino/velvet/) is a highly popular short-read assembler which is available on Bio-Linux. There are countless parameters and combinations to achieve the best assembly, but we will run close to default here. We will assess the quality of the assemblies in the next step.

Running velvet in single-end mode with k=21

'k' signifies the Kmer length i.e. the length of sub sequences that the data is being broken up into, and is often one of the most important parameters to manipulate. Full parameters can be seen by typing either command with no flags.

```
# You should still be in the results directory at this point
$ velveth velvet_k21 21 -short -fastq qual_trim_mt1.fastq
$ velvetg velvet_k21 -read_trkg yes -amos_file yes
```

We can inspect the results in the Tablet graphical viewer (not ideal - we have 139 contigs):

```
$ tablet velvet_k21/velvet_asm.afg &
```

Quick 'cheat'

VelvetOptimiser is a script which automatically tries multiple parameter combinations and returns the best assembly it can find. Can be good to point you in the right direction.

```
$ velvetoptimiser -s 27 -e 31 -f
      '-short -fastq qual_trim_mt1.fastq' -a 1
$ tablet auto_data_31/velvet_asm.afg &
```

Assembly With Abyss

Abyss (http://www.bcgsc.ca/platform/bioinfo/software/abyss) is another popular assembler which we will run to give a comparison. Again, multitudes of parameters are available, but here we will run almost default.

Running abyss in single-end mode with k=21

```
$ abyss -k21 qual_trim_mt1.fastq -o abyss_contigs.fa
```

Side note: Trying abyss with multiple kmer values

A major benefit of working in a unix environment is the ability to loop easily through multiple values. Without an 'optimiser' type program, a shell loop can be used:

```
#Type the first line and press return.  The prompt will change to "for>"
$ for k in {15..20}
for>  abyss -k$k qual_trim_mt1.fastq -o abyss_k$k.fa
```

Assessing The Assemblies

We used tablet to view the output from Velvet assemblies. This isn't possible with the Abyss output as the program does not provide a full assembly, just the consensus contigs. We can obtain some simple statistics on all the assembly results on the command line.

Comparing assemblies with gnx-tools

The gnx-tools command will output basic statistics on the multi-fasta file produced by the assembler.

```
$ for f in velvet_k21/contigs.fa auto_data_31/contigs.fa abyss_contigs.fa
for> gnx-tools $f
```

Adding Some Annotation

If sequence assembly is a tricky process to master then sequence annotation is a bona fide black art. There are various approaches that one can use and several pipelines available that will help. But in this case, we just want to get something to look at in Artemis. We'll quickly scan the assembled genome for likely open reading frames. We'll use the Abyss output as this has (hopefully!) produced a single contig.

Glimmer3 (http://ccb.jhu.edu/software/glimmer/index.shtml) is an application for predicting open reading frames in prokaryotic genomes. As with the assemblers above, it should generally be tuned for the specific organism that you are working with and also provided with an appropriate training data set. But in this case we will just run it quickly with the default options.

A Perl script is provided to convert the output from Glimmer into something that Artemis can view. You don't need to be a Perl programmer to find and use useful scripts like this.

```
$ g3-from-scratch abyss_contigs.fa glimmer

$ perl ../glimmer_to_gbk.perl <glimmer.predict >glimmer.gbk

$ artemis abyss_contigs.fa &
```

You should now be looking at a view of the contig in Artemis. From the ***File*** menu select ***Read An Entry…*** and choose the file glimmer.gbk.

To conclude this taster session, load the file human_mitochondrial.gbk into Artemis for comparison. This is not the same as the mitochondrial data you've just assembled (which is from L. Rubellus) but it is fully annotated. This annotation will have been achieved using a combination of automated tools and manual editing in Artemis. You can find more on Artemis, and on how to identify genes using BLAST, in the main tutorial booklet.