Tuxedo Suite Tutorial

Robert I. Colautti

February 14, 2017

Overview

In this tutorial you will learn about the Tuxedo Suite of tools for sequence alignment and RNA expression analysis. This involves several tools:

1. Bowtie2 – A very fast, memory-efficient short-read aligner. Use for aligning short DNA sequences to a reference genome. This is the first step in variant detection for GWAS, QTL mapping, and a host of population genetics analyses.
2. TopHat2 – Maps RNA sequences to a reference genome to assist with annotation and identify spliced sequences. Can also run de novo alignments when a reference genome is not available.
3. CuffLinks:
   3a. Cufflinks – assemble transcripts
   3b. Cuffcompare – compare assemblies to annotation
   3c. Cuffmerge – merge two or more transcript assemblies
   3d. Cuffdiff – identify loci with differential expression; detect alternative splicing and promoter regions

Bowtie2

Indexing a genome

Let's begin by mapping reads from our Illumina MiSeq short reads to our chloroplast reference genome. Start by loading the bowtie2 package and then taking a look at the help file.

```
$ use bowtie2
$ bowtie2 --help
```

Take a second to think about the computation required to align a few hundred million short reads, each requiring a match of a few hundred bases long, to a reference geneome that may be hundreds of millions to billions of bases long. To keep memory requirements from getting out of control (i.e. GB instead of TB), Bowtie uses the Burrow's Wheeler Transform link (https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform).

Bowtie also requires an indexed reference genome. References for well-studied organisms are available online, but in our case we'll have to create a set of indices ourselves. It can take a while for larger genomes but only has to be done once. Since several files are created, it is helpful to keep the genome and all the associated indices in a separate folder, which should already been created in the **.tar.gz** file:

```
$ ls -l ./Ap_genome
```

```
total 791
-rw-r--r-- 1 hpc3183 hpcg1540 400153 Feb 20 08:44 Ap_cpDNA_annotation.gff
-rw-r--r-- 1 hpc3183 hpcg1540 153259 Feb 20 08:44 Ap_cpDNA_genome.fa
```

Note the fasta file with the .fa extension, which contains the draft chloroplast (cp) sequence as a single contig.

To index the cp genome, run the *bowtie2-build* program. Since the genome is small, we could do this from the command line. However, for most genomes (100 Mb+) you would want to put this into a bash file, so let's do that.

Create a new bash file called **genindex.sh** text file with your favorite text editor:

```
nano genindex.sh
```

Now enter the usual header of the bash script:

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -q abaqus.q
#$ -l qname=abaqus.q
#$ -m e
```

Then choose the names of the output files and tell the scheduler to load the bowtie2 commands:

```
#$ -e genindex.err
#$ -o genindex.out

use bowtie2
```

Take a quick look at the *bowtie2-build -h* file. You'll see that, at a minimum, we need to specify the location of the genome file, and set a file prefix for the output (index) files. It's important to remember that the cp genome is in a folder called Ap_cpDNA_genome.fa. Since we set **-cwd** in the bash file, we have to be careful to either use the path that is relative to the directory that we will submit the script from, OR use an absolute path. Let's use the relative path in the bash script:

```
bowtie2-build ./Ap_genome/Ap_cpDNA_genome.fa Ap_cpDNA_genome
```

In this case, we want to be sure to submit the script from the directory that contains the Ap_genome directory. Be sure to exit and save your changes. Then, submit the script using qsub:

```
$ qsub genindex.sh
```

Now there should be our output files (**.err** and **.out**), which we can inspect to make sure that everything ran smoothly:

```
$ nano genindex.out
```

There should also be several new index files in our genome folder:

```
$ ls -l ./Ap_genome
```

Aligning short-reads

Now that we have an indexed reference genome, we can try to align our short-read data to it. We'll do this for the same MiSeq data that we used in our **de novo genome assembly** tutorial. Let's start with another bash script that we'll call *MiSeqAlign.sh*:

```
$ nano MiSeqAlign.sh
```

Enter the usual first few lines, with custom output file names, then load the bowtie2 commands:

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -q abaqus.q
#$ -l qname=abaqus.q
#$ -m e
#$ -e MiSeqAlign.err
#$ -o MiSeqAlign.out

use bowtie2
```

Now we add the bowtie2 command, specifying the index prefix (**-x**) and the MiSeq mate-pairs using **-1** and **-2** Note that it is the number '1', not the letter 'l'. We'll also specify a name for the output **.sam** file. Finally, we'll break up one long line into several lines using the backslash, to make the script more readable:

```
bowtie2 -x ./Ap_genome/Ap_cpDNA_genome.fa \
-1 ../deNovo/MiSeqF.fq -2 ../deNovo/MiSeqR.fq \
-S MiSeqAligned.sam
```

Note that we are using the short-read sequences from the de novo tutorial, which in a different directory. Using the '../' tells the scheduler to go to the parent of the current directory, then the 'deNovo/' tells it to go into the deNovo folder.

We could try to run this script, but it will take a little while to run, even with the short genome, because of the large number of sequences. It is usually possible to use parallel computing with bioinformatics programs that iterate (i.e. repeats) the same analysis over and over again (e.g. aligning a bunch of

sequences). One good tip is to check the help file for a parameter that allows you to specify the number of **threads** or processors. If you have a very long help file, a good trick is to search for the string using the grep command:

```
$ bowtie2 --help | grep 'thread'
```

Note the vertical line **|** or **pipe** character in the command above. This is a very useful **UNIX** command that says 'take whatever is output from the command on the left side of the pipe and use it as input for whatever is on the right side'. So in this case, it is taking the output from the **help** command and inputting it into the **grep** command to search for the string 'thread'.

grep uses **Regular Expressions** or **REGEX**, which is a very powerful set of syntax rules for searching text. We don't have time to go into this in detail, but there are many tutorials online. I've also written one for R: LINK (http://post.queensu.ca/~rc91/R/4_regex.html)

```
  -p/--threads <int> number of alignment threads to launch (1)
```

If you look at the output from the command above, you'll see a single line from the help file, telling us that we can use **-p** as a parameter in the **bowtie2** command to run in parallel. If you recall from the de novo assembly tutorial, we do this by adding a line to the header of the bash script (**#$ -pe shm.pe 8**), and then use **$NSLOTS** with the -p parameter in the actually script. Do this and then use qsub to submit the script. It may take a few minutes to run.

When it is done running, use **ls -l** to inspect the output. You'll see a very large **SAM** file (>1GB).

```
-rw-r--r-- 1 hpc3183 hpcg1540 1047343144 Feb 20 09:46 MiSeqAligned.sam
```

We looked at **SAM** files in the de novo assembly tutorial, but let's take a quick look at this file. Remember, if we try to use a standard text editor we are going to use up a lot of memory and may crash our computer. Instead, we should us **head**, **tail** or **less** to inspect the file. Let's use head to look at the first few lines.

```
$ less MiSeqAligned.sam
```

The **.sam** file is just like a large spreadsheet, with tabs delimiting the columns. You can use the right arrow to scroll across to get a better sense of this.

The first three lines begin with the **@** symbol and specify the header of the file:

- **@HD** – The header line
    - **VN:** Version number of the alignment (not the Bowtie2 program)
    - **SO:** – Sorting order (usorted)
- **@SQ** – Reference sequence dictionary
    - **SN:** Reference sequence name (taken from first line of **FASTA** file)
    - **LN:** Length of the reference sequence
- **@PG** – Program info
    - **ID:** – Program ID

- **PN:** – Program name
- **VN:** – Program version
- **CL:** – The command line used to generate the alignment (compare with MiSeqAlign.sh)

The fourth line shows the data for the first alignment:

1. **QNAME** – The name of the sequence; from the **FASTQ** file
2. **FLAG** – a bit-score FLAG; explained on Wikipedia page LINK (http://genome.sph.umich.edu/wiki/SAM)
3. **RNAME** – Name of the reference alignment (from **FASTA** file)
4. **POS** – Mapping position (location along reference)
5. **MAPQ** – Quality score for the mapped sequence (probability of correct match)
6. **CIGAR** – A code specifying things like mismatches, insertions and deletions. Note the **=** represents a perfect match
7. **RNEXT** – Name of the mate pair read
8. **PNEXT** – Position of the mate pair read
9. **TLEN** – Length of the template
10. **SEQ** – Actual sequence of the mapped read
11. **QUAL** – Quality the original sequence (same Q-score from from **FASTQ** file)

The last set of columns are optional **TAGS**. For more detail, see the official SAM specification info LINK (http://samtools.github.io/hts-specs/SAMv1.pdf)

In the de novo assembly tutorial, we talked about the difference between the **SAM** file and the compressed **BAM** version. While **SAM** files are human-readable, **BAM** files are much smaller and faster for the computer to work with. We can convert them using the **SAMTOOLS** program:

```
$ use samtools
$ samtools sort MiSeqAligned.sam > MiSeqAligned.bam
```

It will take a few minutes to run. After it is complete, compare the file sizes.

> How much smaller is the **BAM** file compared to the **SAM** file?

The **SAMTOOLS** program is actually a collection of several very useful programs (aka tools), just like the **PICARD** program we looked at briefly in the de novo assembly tutorial. We can use these to inspect our alignment. The **flagstat** tool gives us a report based on the **FLAG** column in the **SAM** file (Col #2)

```
$ samtools flagstat MiSeqAligned.bam
```

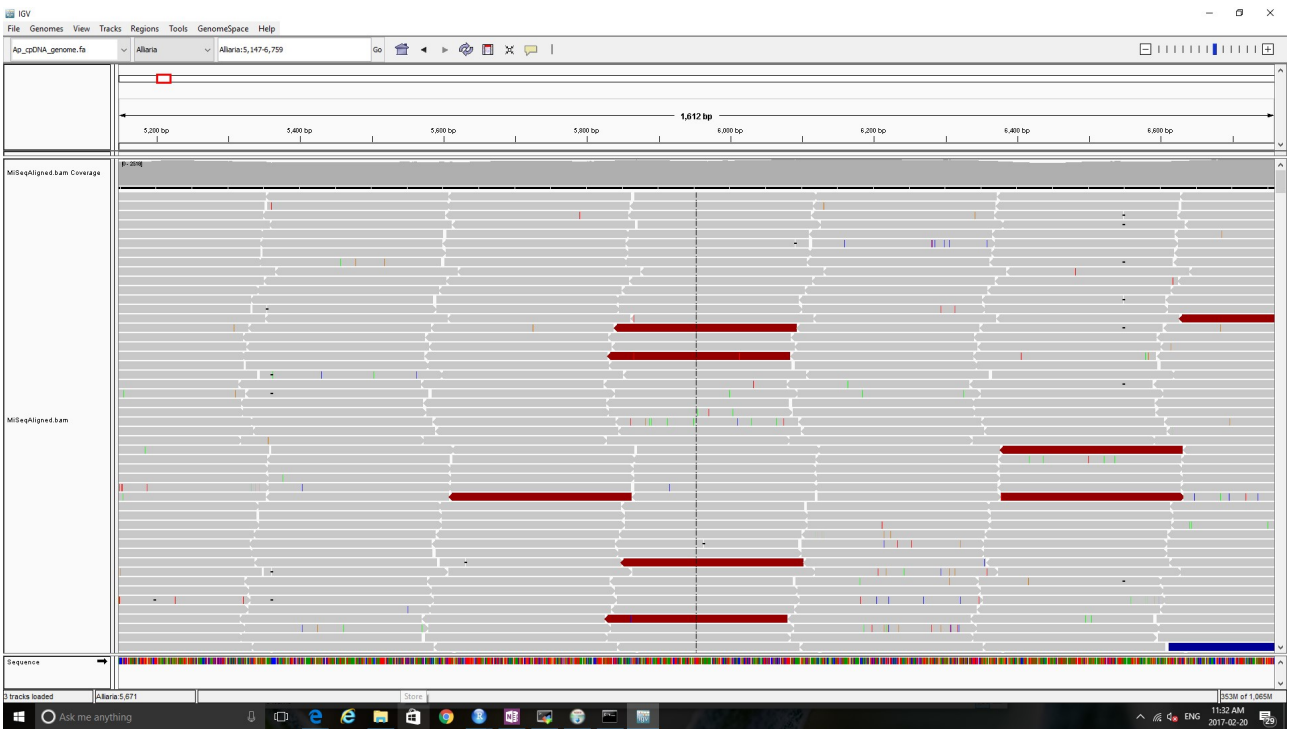
```
1599134 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
0 + 0 supplementary
0 + 0 duplicates
1552016 + 0 mapped (97.05% : N/A)
1599134 + 0 paired in sequencing
799567 + 0 read1
799567 + 0 read2
1520076 + 0 properly paired (95.06% : N/A)
1537634 + 0 with itself and mate mapped
14382 + 0 singletons (0.90% : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

Note that we have no real errors since we were aligning sequences that were already filtered for quality and alignment to the chloroplast genome.

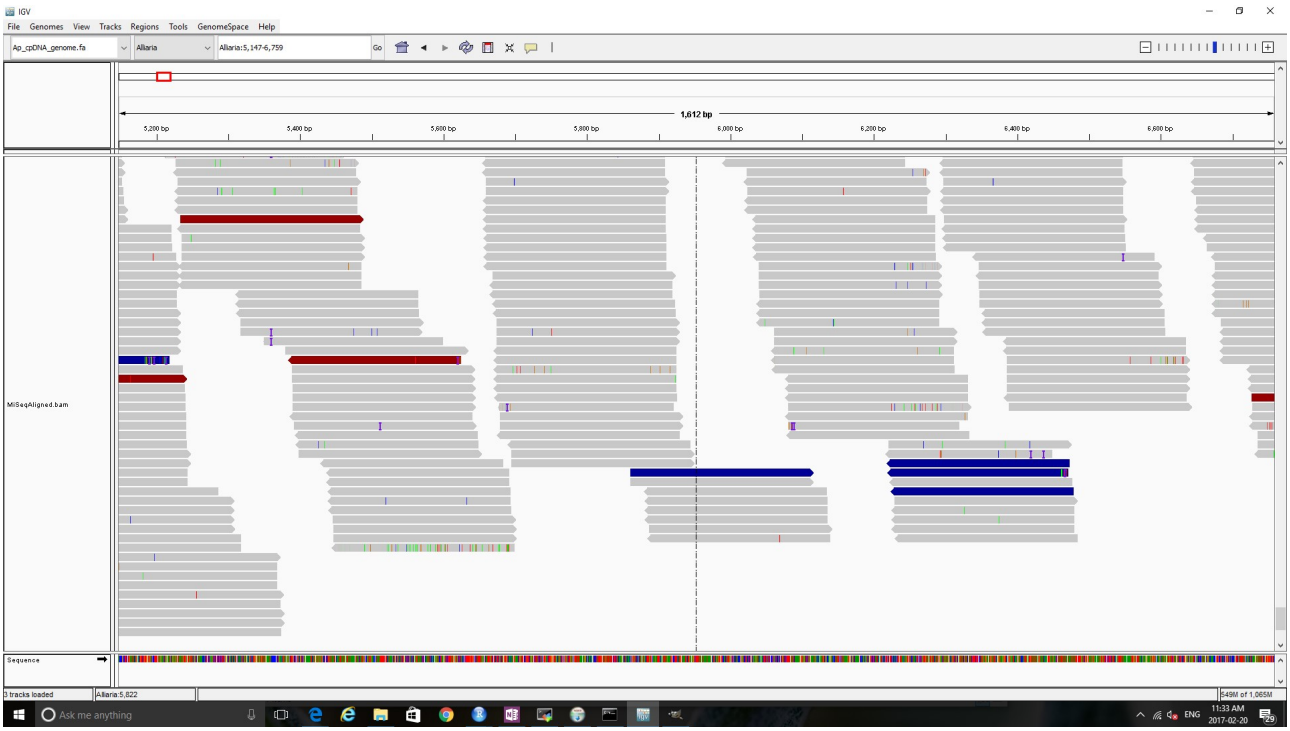
Another thing we might want to do is to index our **BAM** file so that it can be used with a genome browswer like IGV LINK (http://software.broadinstitute.org/software/igv/). This creates a 'bam index' file with the .bai extension that is used by these programs.

```
$ samtools flagstat MiSeqAligned.bam
```

As far as I know, the alignments can't be viewed directly on the cluster, but the **BAM** and **BAI** files can be copied to a computer with a visual interface. When opened with IGV, it looks something like this:



Our coverage is very high so it's easier to see if we scroll down to the end of our aligned sequences:



Each little grey bar shows the alignment and orientation of one of the short reads. Coloured vertical bars show SNP variants relative to the reference genome.

TopHat2

Now that we have some experience aligning sequences, let's get some RNA sequence data from the NCBI sequence read archive (SRA) database. Searching for 'chloroplast RNA sequence'yields a few hits, one of which is an experiment on enriched transcriptome sequences in *Arabidopsis thaliana*. This species is a well-studied 'model organism' that also happens to be in the same taxonomic family as *Alliaria petiolata* – the source of our cpDNA genome! Here is a link to the **BioProject** info: LINK (https://www.ncbi.nlm.nih.gov/bioproject/PRJNA268035)

The word **enriched** usually implies some sort of lab method(s) that increases the representation of something of interest. For example, whole-RNA extractions would contain transcripts from the nucleus and the mitochondrion as well as the chloroplast. In this case, it looks like the authors used centrifugation with a glucose gradient to isolate chloroplast organelles. A second step uses streptavidin-coated magnetic beads used to remove ribosomal RNA (rRNA) from the RNA sample.

The chloroplast transcripts come from two genotypes, the wild-type col0 line and an rnc3/4 double mutant. In this mutant, the RNC3 and RNC4 genes are a disfunctional and apparently affects transcription regulation, so we should see a difference in the transcript profiles.

These have been downloaded already into the setup file. They begin with the prefix **At** followed by the genotype code **col0** or **rnc34** and the SRA ID code. Let's align them to our chloroplast genome. Take a look at the *alignRNA.sh* script:

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -q abaqus.q
#$ -l qname=abaqus.q
#$ -m e
#$ -e alignRNA.err
#$ -o alignRNA.out
#$ -pe shm.pe 8

use tophat2

tophat2 -p $NSLOTS -o col0Aligned \
./Ap_genome/Ap_cpDNA_genome ./At_col0_SRR1722713.fastq.gz

tophat2 -p $NSLOTS -o rnc34Aligned \
./Ap_genome/Ap_cpDNA_genome ./At_rnc34_SRR1724089.fastq.gz
```

We won't run it because it takes a while. The key is to note in the first **tophat2** command: (i) the folder **col0Aligned**, which contains the output based on the input genome and index files (which all start with **Ap_cpDNA_genome**) and (ii) the gzipped fastq file, which contains the sequenced RNA reads from the *col0* genotype. The second **tophat2** command repeats but for the *rnc3/4* genotypewith the *rnc34* prefix for the folder and **FASTQ** data.

The output creates a **BAM** file called **accepted_hits.bam** inside each folder for each of the RNA-Seq experiments. To avoid confusion, let's rename these:

```
$ mv ./col0Aligned/accepted_hits.bam ./col0Aligned/col0Aligned.bam
$ mv ./rnc34Aligned/accepted_hits.bam ./rnc34Aligned/rnc34Aligned.bam
```

Next we should should index the files, just like we did for the DNA sequences earlier with the *genindex.sh* script. In fact, we can simply copy and then modify this script, rather than writing a new one from scratch:

```
$ cp genindex.sh RNAindex.sh
$ nano RNAindex.sh
```

Inside the text editor, we need to change the error (-e) and output (-o) files in the bash header:

```
#$ -e RNAindex.err
#$ -o RNAindex.out
```

Then we change the bowtie2 command to samtools to index the RNA **BAM** file, and repeat for each genotype:

```
use samtools

samtools index ./col0Aligned/col0Aligned.bam

samtools index ./rnc34Aligned/rnc34Aligned.bam
```

Go ahead and submit the script using qsub.

Cufflinks

Now that we have aligned the RNASeq data to the genome and indexed the reads, we need to reconstruct the transcript and estimate abundances so that we can compare expression.

As usual, we should start by looking at the help file

```
use cufflinks
cufflinks --help
```

As you can see, there are a lot of options here. Some of the options are very important and specific to the details of the RNASeq library prep methods. Different methods have different biases in representation, which are corrected for by many of these parameters.

We need to do this separately for both of the genotypes, then compare the abundances as a measure of transcription changes. Check the file *cufflinks.sh*

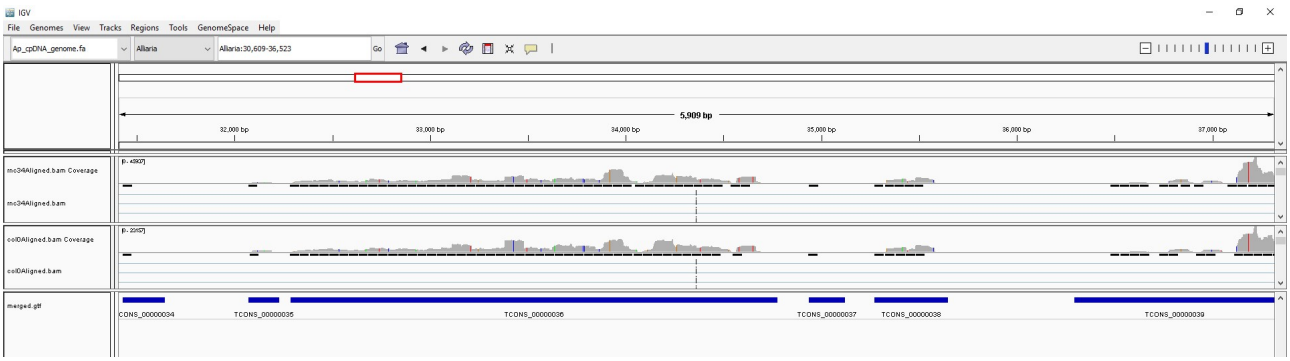
```
cufflinks -p $NSLOTS -o cuffcol0 ./col0Aligned/col0Aligned.bam
mv ./cuffcol0/transcripts.gtf cuffcol0/col0.gtf
```

Notice that the cufflinks command occurs twice; once for each genotype. The mv command again renames the output into something more descriptive

This is followed by two lines that create a text file containing the paths to the two **gtf** files. This is used as input in the final command, which annotates the transcript data with the genome:

```
cuffmerge -s ./Ap_genome/Ap_cpDNA_genome.fa RNAseq.txt
```

This script takes a while to run but produces an annotated genome that be viewed in a genome browser along with coverage variation from the RNASeq data.



Explore further:

- Identify differentially expressed genes with **cuffdiff**
- Analyze expression differences using the **cummeRbund** package in R