Module 6

RNAseq

Goal: Learn how to use various tools to extract information from RNAseq reads.

Input(s):
```
magnaporthe_oryzae_70-15_8_supercontigs.fasta
Moryzae_70-15_liquid_culture{1-3}.fastq.gz
Moryzae_FR13_inPlanta{1-3}.fastq.gz
magnaporthe_oryzae-70-15_8_transcripts.gtf
```

Output(s):
```
MoryzaeHS{1-8}.ht2
Mo_70-15_LC{1-3}_accepted_hits.bam
Mo_FR13_IP{1-3}_accepted_hits.bam
```

6.1 Information to be gained from RNASeq analysis

For this exercise, we will analyze six RNASeq datasets. Three were generated from RNA isolated from fungal mycelium growing in a liquid culture broth (three biological replicates). Three were generated from excised lesions containing the fungus growing in rice leaves. For the first step of our analysis, we will use **HISAT2** to align the RNASeq reads to a genome assembly of one of the fungal strains from which the RNAseq reads were derived (strain 70-15).

Kim D, Langmead B and Salzberg SL. **HISAT: a fast spliced aligner with low memory requirements**. *Nature Methods* 2015

https://daehwankimlab.github.io/hisat2/

Mapping RNASeq reads to a reference genome can tell us several things:

i) It will identify genomic regions that code for RNAs (i.e., regions that are genes).

ii) The number of reads mapping to each gene will tell us how strongly they were expressed in the two conditions from which RNA was extracted (mycelium growing in culture, infectious hyphae growing *in planta*).

iii) Gaps in the mappings will reveal the presence and locations of introns in the genes.

iv) Differences in gap positions will identify variable splice junctions.

v) Nucleotide differences can be used for variant calling (when the reads come from a genome that is different to the reference—which is true in this case).

6.2 Create an indexed reference genome

- ☐ Make sure you are in the *rnaseq* directory.
- ☐ Start a screen called "rnaseq".

First, we need to create an index of the reference genome that will allow faster searching. We will accomplish this using **hisat2-build**.

Usage: `hisat2-build <sequence-to-be-indexed> <prefix-for-index-name>`

- ☐ Index the reference genome.

```
• hisat2-build ~/blast/magnaporthe_oryzae_70-15_8_supercontigs.fasta
  MoryzaeHS
```

6.3 Mapping RNASeq reads to a reference genome

HISAT2 is based on the **Bowtie2** alignment engine but uses a more efficient indexing system for faster searching of RNAseq read alignments to the genome assembly. HISAT2 outputs *.sam* format files that we will need to convert to *.bam* for downstream processing.

Usage:

```
hisat2 [options] -o <output_dir> -x <path-to-index> <input-file(s)>
```

- ☐ Create a new directory called *alignments* to receive the output files from **HISAT2**
- ☐ Use **HISAT2** to map the RNAseq reads for the *liquid_culture1* dataset to the HISAT index (note: the program won't give you any indication it is running unless you made an error during entry).

```
• hisat2 -p 2 -x MoryzaeHS -U
  Moryzae_70-15_liquid_culture1_R1.fastq.gz --max-intronlen 2000
  --summary-file alignments/LC1_summary.txt --dta-cufflinks |
  samtools sort - -@ 2 -O
  bam -o alignments/Mo_70-15_LC1_accepted_hits.bam
```

`-p`	number of processors to use. **Note: you only have two processors available to you for this exercise, but normally you would run as many as are available.**
`-x`	name of index to align against
`-U`	comma-separated list of files containing unpaired reads to be aligned to reference. (we provide only one file here.)
`--max-intronlen`	maximum length of intron (500000 if left blank)
`--summary-file`	write alignment summary to this file
`--o`	name of output file

Note that we pipe the output of **hisat2** straight into **samtools** so that we can sort the results by chromosome position and then convert the sorted alignments from *.sam* to *.bam* format.

By default, **HISAT2** generates a .sam alignment file, which needs to be converted to .bam format and then sorted according to genome coordinates. This can be conveniently accomplished using a single pipe into the **samtools sort** program.

```
samtools sort - -@ 2 -O bam -o output.bam
```

`-`	use the output of the prior program as input
`-@`	number of processors to use
`-O`	output format (bam/sam)
`-o`	output file path

6.4 Examine the RNASeq mapping results

- ☐ First, let's use our command line to look at the *LC1_summary.txt* alignment summary file to get a sense of how much useable data we have.
- ☐ How many total reads were present in the input file? _____
- ☐ How many reads mapped to the reference genome? _____
- ☐ Use **head** to take a quick look at the first 10 lines of results in the alignments file.

```
• head alignments/Mo_70-15_LC1_accepted_hits.bam
```

- ☐ Does the output make any sense? No? That's because the file is in a binary format. Let's use **samtools** to convert the *.bam* file into the human-readable *.sam* format:

```
• samtools view alignments/Mo_70-15_LC1_accepted_hits.bam
```

- ☐ Whoa! Did you catch all that? Quit the process (ctrl-c) and try piping the results through **head**.

  The main fields in sam/bam format are as follows:

1. QNAME	Query name (read ID)
2. FLAG	Bitwise flag describing alignments (e.g., strand, paired info)
3. RNAME	Reference sequence name
4. POS	1-based leftmost position of alignment on reference
5. MAPQ	Mapping quality (Phred value, 0–255)
6. CIGAR	Alignment representation
7. RNEXT	Mate reference name (= on same chromosome, * mate unmapped, or single end read)
8. PNEXT	Mate position
9. TLEN	Template length (inferred length of sequenced fragment based on paired read info)
10. SEQ	Sequence
11. QUAL	Base quality scores (ASCII)

  More info on the *sam/bam* format can be found by searching on Google or ChatGPT.
- ☐ Look at the first alignment record:

☐ What is the alignment length? _____

☐ What is the lowest quality value for the read represented by this alignment (in PHRED format)?

_____

☐ Does this read align on the forward or the reverse strand of the reference? Use the SAM flag tool decoder to find out (https://broadinstitute.github.io/picard/explain-flags.html).

_____

☐ Space bar through a few pages to view a few more alignments and then quit the program (ctrl-C) when you're done.

6.5 Align the remaining liquid_culture datasets

☐ Repeat the alignment commands process for the *liquid_culture2* and *liquid_culture3* datasets. You can use the up arrow to speed up the command line entry but remember to change both the input AND output file identifiers. (Remember, there are two output files.). For example,

```
• hisat2 -p 2 -x MoryzaeHS -U Moryzae_70-15_liquid_culture2_R1.fastq.gz
  --max-intronlen 2000 --summary-file alignments/LC2_summary.txt
  --dta-cufflinks | samtools sort - -@ 2 -O bam
  -o alignments/Mo_70-15_LC2_accepted_hits.bam
```

etc... After each run is complete, list the *alignments* directory contents to make sure that the expected files were created.

6.6 Align the inPlanta datasets

☐ Now you have probably discovered that sitting and waiting for long alignment jobs to finish can make repetitive tasks very frustrating. Let's learn to appreciate the power of the command line for automation. We will perform the alignments for all three *inPlanta* datasets using a single command to loop through the three files, one at a time:

```
• for f in {1..3}; do echo Working on dataset
  Moryzae_FR13_inPlanta${f}_R1.fastq.gz; hisat2 -p 2
  -x MoryzaeHS -U Moryzae_FR13_inPlanta${f}_R1.fastq.gz
  --max-intronlen 2000 --dta-cufflinks
  --summary-file alignments/IP${f}_summary.txt | samtools sort -
  -@ 2 -O bam -o alignments/Mo_FR13_IP${f}_accepted_hits.bam; done
```

Here the **for** loop sequentially assigns the values 1, 2, and 3 to the variable `f`. We then substitute the variable `${f}` for the numerical identifiers in our input and output files. The semicolons (";") allow us to run several commands in a single line sequentially without passing the input of one command to the next (unlike a pipe). For example, we first run **echo** and then **hisat2**.

☐ You should see a run time message saying "Working on dataset…". If instead, you see a command prompt ">," you made a mistake in typing the for loop. In that case, quit the command (ctrl-c) and re-type the command.

- ☐ When the three runs are completed, list the *alignments* directory contents to make sure that all of the expected files were created.

6.7 Assembling transcripts from RNAseq data

We will use **StringTie** to build complete transcripts from the individual RNAseq read mappings.

Pertea et al. (2015) StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. Nature Biotechnology 33(3):290-295.

http://ccb.jhu.edu/software/stringtie/index.shtml

The first step in differential gene expression analysis is to identify the gene from which each sequence read is derived. **StringTie** examines the RNA-seq read mapping results produced by a read aligner such as **TopHat** or **HISAT2** and attempts to reconstruct the complete transcripts, identify transcript isoforms, and estimate transcript abundance. **StringTie** does this by clustering reads and constructing a splice graph for each cluster from which it will identify transcripts. A flow network is then constructed for each identified transcript to estimate the abundance of the transcript via a maximum flow algorithm. StringTie can accept an existing gene annotation file as a guide for constructing transcripts.

Usage:

```
stringtie <path/to/accepted_hits.bam> [options]
```

- ☐ Take a quick peek at all the **StringTie** options that are available.

```
• stringtie --help
```

- ☐ Make sure you are in the *rnaseq* directory.
- ☐ First, take a look at the existing annotation file, *magnaporthe_oryzae_70-15_8_transcripts.gtf*, to be sure you understand its format.

Note that the GTF format is very similar to the GFF3 format used in the *.gff3* file that you produced in the gene finding exercise. Each feature entry has an MGG gene_id which identifies the gene to which the feature corresponds. Note also that each gene has one or more corresponding transcript_ids, with T0 denoting the first transcript isoform, T1, the second, etc.

- ☐ Create a directory named *transcripts*.
- ☐ Now we are ready to run **StringTie**, and we will provide the *.gtf* file as a reference transcriptome:

```
• stringtie alignments/Mo_70-15_LC1_accepted_hits.bam
  -p 2 -G magnaporthe_oryzae_70-15_8_transcripts.gtf
  -o transcripts/Mo_70-15_LC1_transcripts.gtf
```

`-p`	number of processors to use
`-G`	tells **StringTie** to use the provided reference annotation to guide transcript assembly but also to report novel transcripts/isoforms
`-o`	name of output directory

Notes:

A) Omitting the `-G` option (and accompanying *.gtf* file specification) from the above command would tell the program to generate a *de novo* transcript assembly. Alternatively, one can use `-G` with an additional `-e` flag, which will tell the program to assemble only those reads that correspond to previously identified genes/transcripts.

B) It is recommended that you assemble your replicates individually, i) to speed computation; and ii) to simplify junction identification. Therefore, you will need to run **StringTie** separately for each of your *.bam* files.

☐ Make sure that the *Mo_70-15_LC1_transcripts.gtf* output file was created indicating that the command completed successfully.

☐ Look at the first few lines of the *Mo_70-15_LC1_transcripts.gtf* in order to answer the questions below (use the information below to understand what the different fields mean).

The final column of a GTF file is a semicolon-separated list of tag-value pairs providing additional information about each feature. Depending on whether an instance is a transcript or an exon and on whether the transcript matches the reference annotation file provided by the user, the content of the attributes field will differ. The following list describes the possible attributes shown in this column.

a. gene_id: A unique identifier for a single gene and its child transcript and exons based on the alignments' file name.

b. transcript_id: A unique identifier for a single transcript and its child exons based on the alignments' file name.

c. exon_number: A unique identifier for a single exon, starting from 1, within a given transcript.

d. reference_id: The transcript_id in the reference annotation (optional) that the instance matched.

e. ref_gene_id: The gene_id in the reference annotation (optional) that the instance matched.

f. ref_gene_name: The gene_name in the reference annotation (optional) that the instance matched.

g. cov: The average per-base coverage for the transcript or exon.

h. FPKM: Fragments per kilobase of transcript per million read pairs. This is the number of pairs of reads aligning to this feature, normalized by the total number of fragments sequenced (in millions) and the length of the transcript (in kilobases).

i. TPM: Transcripts per million. This is the number of transcripts from this particular gene normalized first by gene length, and then by sequencing depth (in millions) in the sample.

☐ **For the first gene reported for chromosome 8.1:**

a. Does this gene correspond to a previously annotated *Magnaporthe* gene (does it have an MGG identifier), or is it novel? previously annotated: _____; novel: _____

b. How long is the gene? _____ bp

c. Does it contain any introns? yes: _____; no: _____.

d. How abundantly is it expressed in fragments per kilobase per million reads? _____

☐ Now we will complete the rest of the transcript assemblies by using another kind of **for** loop.

```
• for f in $(ls alignments/*bam | grep -v LC1); do
  echo Working on $f; g=${f/*\//}; stringtie $f -p 2 -G
  magnaporthe_oryzae_70-15_8_transcripts.gtf
  -o transcripts/${g/accepted*/}transcripts.gtf; done
```

Here we are running the loop a little differently. We list all bam files in the alignments directory, and, after omitting the file that we've already processed (using `grep -v LC1`), we assign the results to the variable **f**. Then, for each value (i.e., each file) of **f**, we strip off everything before and including the first forward slash (`${f/*\//}`) and save the result to the variable **g**. Then, we run **stringtie** on each value of `f` (*alignments/Mo_70-15_LC1_accepted_hits.bam*, etc.) and output the results to the transcripts directory under a filename that we create by cutting off the *accepted_hits.bam* suffix from what is stored in the **g** variable (`${g/accepted*/}`). We then place what's left (e.g., "Mo_70-15_LC2_") in front of "transcripts.gtf".

6.8 Merging transcript assemblies

We will use **StringTie** with the `-merge` option to generate a "super-assembly" of transcripts based on the mapping information from all six RNAseq datasets. This will allow **StringTie** to identify overlaps between alignment data for different datasets. In this way, it can assemble complete transcripts for genes whose expression levels are too low to allow full transcript reconstruction from a single sequencing lane.

Usage:

```
stringtie --merge [options] <list_of_gtf_files>
```

☐ **Make sure you are in the *rnaseq* directory.**

☐ First, we need to create a text file that lists the paths to the various transcript assemblies that we want *stringtie* to merge. This document should have the format:

```
  GNU nano 4.8  assemblies_gtf_list.txt
./transcripts/Mo_70-15_LC1_transcripts.gtf
./transcripts/Mo_70-15_LC2_transcripts.gtf
./transcripts/Mo_70-15_LC3_transcripts.gtf
./transcripts/Mo_FR13_IP1_transcripts.gtf
./transcripts/Mo_FR13_IP2_transcripts.gtf
./transcripts/Mo_FR13_IP3_transcripts.gtf


^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text
```

Here is an example of where a file created by a standard text editor such as **Word** will not be read properly by the **stringtie** program and would produce an error.

☐ We could open a test editor such as **nano** to create this document but a much more efficient way to do this (while avoiding typing errors) is to use another **for** loop:

```
• for f in $(ls transcripts/*gtf); do echo ./$f >>
  assemblies_gtf_list.txt; done
```

>> means redirect to a file and append new entries to the same file

☐ Now we can run **stringtie** with the **`--merge`** option.

```
• stringtie --merge -p 2 -o merged_asm/merged.gtf
  -G magnaporthe_oryzae_70-15_8_transcripts.gtf
  assemblies_gtf_list.txt
```

`-p` number of processors to use

`-o` output file name for the merged transcripts. Here, **stringtie** will create the *merged_asm* directory if it doesn't already exist (this is not the case for many programs)

`-G` include the reference annotation in the merging operation

☐ Examine the *merged.gtf* file produced by `stringtie --merge` inside of the *merged_asm* directory.

☐ Use command line tools to interrogate the *merged.gtf* file to identify novel transcripts that have not been previously identified. Note: novel transcripts will lack MGG identifiers.

You will note that the gene-id attribute for each transcript has a **stringtie**-specific prefix (MSTRG). This is a bit of a problem because, if we use this file for downstream analyses such as differential gene expression analysis, this will be the gene ID that is displayed, even if there is already an established MGG identifier. To get around this problem, we will use a custom perl script to rename the gene_IDs and save to a new file (*cufflinks.gtf*).

☐ Use the **Inherit_IDs.pl** script to convert the gene-id field to its corresponding MGG identifier (if available).

```
• perl Inherit_IDs.pl merged_asm/merged.gtf
  > merged_asm/cufflinks.gtf
```

☐ What is the total number of genes that show at least one alternative transcription start site and/or alternative splice junction? (Recall that the first transcript isoform is given a T0 identifier.)

_____

6.9 Differential gene expression analysis

The **HISAT2/StringTie** pipeline normally feeds into the **Ballgown** program for differential expression analysis (Frazee AC et al. 2015. Nat Biotechnol. 33:243-246). However, this would require many of us to learn the R programming language and environment, which itself could take a whole week. Therefore, we will use the legacy **cuffdiff** program to determine which genes are differentially expressed in one of the RNAseq datasets.

http://cole-trapnell-lab.github.io/cufflinks/cuffdiff/

For our purposes, we will have **cuffdiff** use the *cufflinks.gtf* file which combines the prior gene annotations with the new information (novel transcripts, isoforms, etc.) generated from our RNAseq data. **cuffdiff** then uses the alignment data (in the *.bam* files) to calculate and compare transcript abundances.

Usage:

```
cuffdiff [options] <transcripts.gtf>
      <condition1.repl1.bam,condition1.replicate2.bam…
      <condition2.replicate1.bam,condition2.replicate2.bam…>
```

Note: experimental replicates are separated with commas; datasets being compared are separated by a space (i.e.: Condition1_rep1,Condition21_rep2 (SPACE) Condition2_rep1,Condition2_rep2).

Since **bash** doesn't know about the way **cuffdiff** separates the experimental replicate files (and we have not provided any custom code to **bash** for tab-completing **cuffdiff**), tab completion will not work for an argument after you type the first comma. We can get around this limitation by initially separating with spaces and then replacing the appropriate spaces with commas.

For our experiment, we will compare transcript abundance in fungus grown in liquid culture (three replicates) versus fungus growing *in planta* (three replicates).

☐ Run **cuffdiff** as follows (**DO NOT PUT A SPACE BETWEEN THE COMMAS AND THE FLANKING DATASETS—SEE ABOVE COMMENT IN RED BEFORE RUNNING**):

```
• cuffdiff -o diff_out -p 2 -L culture,inPlanta
  -u merged_asm/cufflinks.gtf
  alignments/Mo_70-15_LC1_accepted_hits.bam,
  alignments/Mo_70-15_LC2_accepted_hits.bam,
  alignments/Mo_70-15_LC3_accepted_hits.bam   (include a space here)
  alignments/Mo_FR13_IP1_accepted_hits.bam,
  alignments/Mo_FR13_IP2_accepted_hits.bam,
  alignments/Mo_FR13_IP3_accepted_hits.bam
```

`-o` output directory where results will be deposited

`-p` number of processors to use

`-L` Labels to use for the two conditions being compared. These labels will appear at the top of the relevant columns in the various output files.

`-u` Tells **cufflinks** to do an initial estimation procedure to more accurately weight reads mapping to multiple locations in the genome

☐ This produces many output files in the *diff_out* output directory. Most of these are not really needed for us to identify the genes that have expression differences, so we will not go into them in detail. A brief explanation of files is provided in the appendix to this module.

☐ The main information about gene expression differences are written to the file named *gene_exp.diff* inside the defined output folder. View the header of this file and see if you can determine what information is contained in the various columns. If necessary, look at the description of the output columns in the appendix to this module, or you can find more information in the online **cufflinks** manual:

http://cole-trapnell-lab.github.io/cufflinks/manual/

- ☐ Practice using the command line to produce an output that shows only those genes that **cuffdiff** predicts to be differentially expressed between the two samples.
- ☐ Use **awk** to print the lines of results for genes that show a more than ten-fold higher expression *in planta* versus in liquid culture.
- ☐ Use the command line to produce a list that contains the identities of the genes that show significant differences in their expression levels (only the names of the genes and nothing else). Write this list to a file.

Hint: You will need to use **grep** and/or **awk**.

Appendix

Explanation of cuffdiff output files:

The main types of files produced are:

bias_params.info	The bias model learns how read coverage varies across transcript sequences, and stores those parameters here.
model.info	stores information about the empirical fragment length distribution model that Cuffdiff builds when estimating transcript abundance.
count_tracking	reports raw and normalized fragment counts for each feature across individual replicates.
fpkm_tracking	reports FPKM values for each feature (gene, isoform, etc.) across all samples and replicates.
read_group_tracking	contains per-replicate quantification data — including FPKM values, fragment counts, and confidence intervals

Several differential expression output files are produced:

gene_exp.diff	Gene-level differential expression. Tests differences in the summed FPKM of all transcripts sharing the same gene_id
isoform_exp.diff	Transcript-level differential expression. Tests differences in the summed FPKM of each transcript isoform for a given gene
tss_group_exp.diff	Primary transcript differential expression. Tests differences in the summed FPKM of transcripts sharing the same transcription start site (tss)
cds_exp.diff	Coding sequence differential expression. Tests differences in the summed FPKM of transcripts that result in the same coding sequence, regardless of tss.

Each of the differential expression output files has the following fields:

test_id	identifier of the feature being tested
gene_id	official gene/feature identifier (if it was previously annotated)
gene	official name of gene (e.g. BRCA2)
locus	genomic coordinates
sample_1	identifier for test group 1 (e.g. inCulture)
sample_2	identifier for test group 2 (e.g. inPlanta)
status	whether a statistical test was performed
value_1	mean expression value for group 1
value_2	mean expression value for group 2
log2(fold_change)	log2 fold change in expression for group 2 relative to group 1

test_stat	test statistic value
p_value	raw p-value for the test
q_value	adjusted p-value after Benjamini-Hochberg correction for false discovery rate
significant	whether the expression level is significantly different between group 1 and 2