# RNA-seq pipeline

**Connor Ludwig & Abby Thurm**

## PART I – Preparation and Alignment

For the following section, you will be using the Linux command line. Open Ubuntu and navigate to the folder where your RNA-seq scripts and data are/will be stored. You can change directories using 'cd' and list directory contents with 'ls' to help navigate. To go up a level, you can use '../', which can also precede a file or folder. The following section also uses Python 3, called from the command line.

Create a FASTA file in Sublime or another text editor software. FASTA files have two lines per entry, where the first line starts with '>' and contains the sequence identifier (e.g. gene name). Avoid spaces in these identifiers. The second line contains the DNA sequence, which you can copy from Benchling. You only need to include the sequence of the transgene of interest, not the promoter, 3xFLAG, or any other plasmid elements.

Create a custom genome with your transgenes appended as extra chromosomes to the human genome (i.e. GRCh38). First, generate a FASTA file for this custom genome by using concat_fastas.py:

```
python concat_fastas.py [path_to_GRCh38_fasta] [path_to_transgene_fasta]
[output_fasta_save_path]
```

Example:

```
python scripts/concat_fastas.py
../transcriptomes/Homo_sapiens.GRCh38.dna.primary_assembly.fa
../transcriptomes/20220506_HHV_RNA-seq_proteins.fasta
../transcriptomes/GRCh38_20220506_HHVs.fasta
```

Build the custom genome using hisat2-build. You'll need to specify the concatenated FASTA file from the previous step as well as a root name for your new genome in the directory where you want it saved. The easiest way to devise a root name for the genome is to use the concatenated FASTA name without the file extension. This script will produce six .ht2 files sharing the root name.

```
hisat2-build [path_to_concat_fasta]
[desired_save_dir/root_name_for_new_genome]
```

Example:

```
hisat2-build ../transcriptomes/GRCh38_20220506_HHVs.fasta
../transcriptomes/GRCh38_20220506_HHVs
```

Download raw data from your NextSeq run. Name your sample sheet ‘SampleSheet.csv’ and place in the run folder. Demultiplex your data to generate separate fastq.gz files for each sample and for each direction sequenced.

```
bcl2fastq --runfolder-dir [dir_in_path] -p 12 --output-dir [dir_out_path] --
no-lane-splitting
```

Example:

```
bcl2fastq --runfolder-dir 220505_NB551514_0768_AHWJCGBGXL/ -p 12 --output-dir
fastq_files/ --no-lane-splitting
```

Create a directory for each sample (named with the same root as the read files) and move the paired read fastq.gz files to the appropriate folder. Place these folders into a parent folder called ‘fastq’ or something similar (exact name does not matter).

Align your reads to your custom genome using hisat2. This script batch processes all files, iterating over the paired read fastq.gz files to produce SAM alignment files. The script then generates BAM files (binary versions of SAM) to be used in the downstream DESeq analysis.

```
python batch_hisat2_alignment_admera_20220506.py [path_to_fastq_directory]
[path_to_genome_dir/genome_root]
```

Example:

```
python scripts/batch_hisat2_alignment_admera_20220506.py fastq/
../transcriptomes/GRCh38_20220506_HHVs
```

You can perform a quick check to make sure your alignment worked by searching for a transgene in a sample where it was expressed versus one where it was not. You can use grep for this search.

```
grep [transgene_name_as_it_appears_in_the_fasta] [path_to_sam_file]
```

Example:

```
grep VIRF2 sam/K562_CL290_dox_BR1.sam
```

Prepare a sample table for downstream DESeq analysis. Check the script to make sure it has the appropriate fields (e.g. it makes an IFN column if this is one of the conditions tested). The script uses the names of the SAM files to populate and output a CSV table that contains information about cell type, plasmid, dox condition, and replicate.

```
python sampletable_prep.py [path_to_sam_dir]
```

Example:

```
python scripts/sampletable_prep.py sam/
```

Prepare a GTF (genome annotation) file for downstream DESeq analysis. This script takes a FASTA file as an input. It is important to use the same FASTA file as that which was used to build the custom genome so that the transgene names match.

```
python make_transgene_gtf.py [path_to_transgene_fasta] [path_to_GRCh38_gtf]
[path_to_new_gtf]
```

Example:

```
python scripts/make_transgene_gtf.py ../transcriptomes/20220506_HHV_RNA-
seq_proteins.fasta ../transcriptomes/Homo_sapiens.GRCh38.104.gtf
../transcriptomes/GRCh38_20220506_HHVs.gtf
```

PART II – DESeq

The following section will be dealing with scripts in R and packages from Bioconductor. We edit and run scripts using RStudio. Please note that we are less proficient at R, so it is very possible that many parts of the following pipeline could be streamlined or done another way. However, the most important pieces of the following workflow come from a Bioconductor DESeq tutorial that can be found here: http://bioconductor.org/help/course-materials/2016/CSAMA/lab-3-rnaseq/rnaseq_gene_CSAMA2016.pdf.

Create a copy of the DESeq script in R for your specific experiment. For instance, I'm using '20220500_DESeq_HHVs.R' as the name. Update the script with the paths to the appropriate files. You'll first need to specify your parent directory (e.g. 'F:/Connor/20220505_NextSeq_RNA-seq/') for where the sample table is located. You'll next need to specify the BAM file directory. Finally, you'll need to specify the directory where the GTF file is stored, as well as the GTF file name (e.g. 'F:/Connor/transcriptomes/' and 'GRCh38_20220506_HHVs.gtf).

Execute the first part of the script to create a SummarizedExperiment object using summarizeOverlaps. This is the most computationally and time-intensive part of this process (surprisingly, this only took about 10 minutes in the most recent run). Be sure to run the line that writes this data to a CSV (se_counts.csv), which can be loaded later to avoid having to re-run the above.

Run the functions stored in RNA-seq_functions.R. You should execute all lines in this script before using any of the functions. The scripts are well-annotated, so I will just put examples below. Start with some general, high-level analyses, including PCA and eigencore plots to get a sense of the largest sources of variance across samples:

make_PCA_plots(1:24, 'plasmid', 'Rep')

```
make_eigencorplot(1:24, ‘plasmid’, ‘Rep’)
```

Next, see how a dox vs nodox sample compares and get the log2FC values. Here’s an example with mCitrine, which we expect to have no effect on host gene expression:

```
get_DESeq(1:4, ‘dox’, ‘Rep’, ‘dox’, ‘nodox’)

get_ordered_results(1:4, ‘dox’, ‘Rep’, ‘dox’, ‘nodox’, ‘CL048_dox-nodox.csv’)

make_volcano_plot(1:4, ‘dox’, ‘Rep’, ‘dox’, ‘nodox’)
```

We can do the same as the above, but for comparing a viral transgene such as VIRF3 to mCitrine in the presence of dox.

```
get_DESeq(c(1:2, 5:6), 'plasmid', 'Rep', 'CL194', 'CL048')

get_ordered_results(c(1:2, 5:6), 'plasmid', 'Rep', 'CL194', 'CL048', 'CL194-CL048_dox.csv')

make_volcano_plot(c(1:2, 5:6), 'plasmid', 'Rep', 'CL194', 'CL048')
```