

RNA sequencing and ribosome profiling have revolutionized biology, allowing experimenters to measure gene expression at both the levels of transcription and translation genome-wide with very high levels of sensitivity and reproducibility. Other techniques, such as CHIP-seq and CLIP-seq, use next generation sequencing to identify interaction sites between DNA and RNA binding proteins and their targets.

The goal of this primer is to get you comfortable using the Linux command line and common bioinformatics tools. You will learn here how to analyze files containing raw sequencing data in order to identify differentially expressed genes between control vs. experimental RNA-seq and ribosome profiling datasets.

As you go through this primer, you can always see the contents of a file you've produced by using the `more` command. I recommend doing this on various files (as long as they aren't in a compressed/zipped format) so you have an idea of what they look like.

Before we begin here are a few more useful tips. The `cd` command switches directories. For example, the command `cd .` keeps you in the current directory, `cd ..` moves you to the parent directory, and `cd folder_a` moves you into a directory called "folder_a" (if it exists). Adding `&` after a command will allow it to run in the background allowing you to run multiple commands at once. Adding `nohup` at the beginning of a command will allow it to keep on running even if your connection to the server gets disconnected. Finally, the `ls -lh` command is very useful since it will allow you to see what files are in the current directory:

```
> ls -lh (outputs list of files in the directory)
> more example_file.fastq (outputs some of the file; press space to see more or q to quit to the command line)
```

Some common file types are `fastq` files, which contains raw sequencing reads and quality scores in PHRED format, `sam` files, which contain aligned reads, along with their chromosomal position and information such as whether the read is uniquely mapping and in the sense or antisense orientation, and `GTF` files, which contain genome annotation information. Files are often stored and downloaded in a zipped format (`.gz`) and must be unzipped using the command `gunzip file_name` (or the wildcard `*.gz` for all zipped files in a directory). When storing files long-term, it is recommended to zip them up using `gzip` which helps massively to save space.

(1) First create an account on Compute Canada

Link to apply:

<https://www.computecanada.ca/research-portal/account-management/apply-for-an-account/>

Your Compute Canada account has a `$HOME` folder which allows for 50gb storage (not a ton), and a `projects` folder which has 1TB of storage shared between the lab. In addition there is a `scratch` folder for running big projects which has 20TB available but files older than 60 days get purged once a month.

You will need our group CCRI:

Compute Canada Role Identifier (CCRI): ist-494-01

(2) Log on by using terminal and typing:

```
> ssh user_name@cedar.computecanada.ca
```

(3) Load up a Python module on Compute Canada

```
> module load python/3.9
```

(4) Create a virtual environment by typing:

```
> virtualenv --no-download ~/ENV
```


(5) Activate the virtual environment by typing (**run this command every time you log in**):

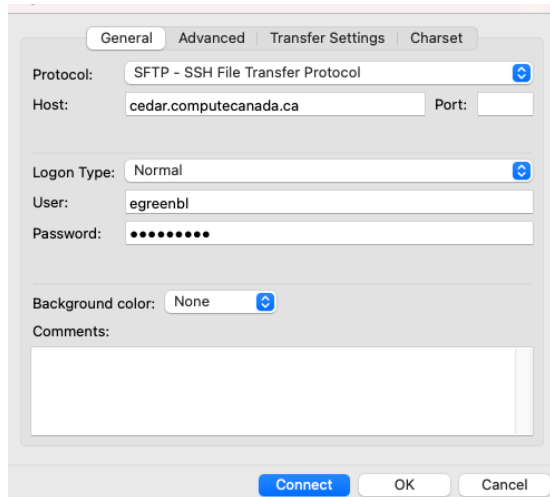
```
> source ~/ENV/bin/activate
```

(6) You will need to install miniconda3 for Linux. Do this by downloading the latest version here on to your personal computer:

<https://docs.conda.io/en/latest/miniconda.html>

(7) You will need to transfer the miniconda installation file you downloaded to your personal computer to your Compute Canada account. I did this using FileZilla, which is a free and secure file transfer protocol application <https://filezilla-project.org/>

You can connect to cedar.computeccanada.ca over an SFTP connection using FileZilla. Click the  button in FileZilla on the top left and use the options below with your user ID. Once it connects you can use FileZilla to drag files onto the server. Please add the miniconda installation files into your subfolder within the projects directory.



(8) After transferring, install conda using these commands (you will need to be in the folder that contains the installation file):

```
> bash Miniconda3-latest-Linux-x86_64.sh
```

(9) Use conda to install commonly used software:

```
> conda install -c bioconda star
```

```
> conda install -c bioconda rsem
```

```
> conda install -c bioconda subread
```

```

> conda install -c bioconda sra-tools
> conda create --name py2 python=2.7
> conda create --name py3 python=3.8
> conda activate py2
> conda install -c bioconda ribodiff
> conda deactivate py2

```

(10) Download the genome and annotation (GTF) files for each organism of interest. This guide will cover how to do it for humans, flies, mice, and E.coli. First make a directory in /scratch and then download using wget:

```

> cd scratch
> mkdir genomes
> cd genomes
> mkdir mouse
> mkdir fly
> mkdir human
> mkdir ecoli
> cd mouse
> wget --timestamping
ftp://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.fa.gz -O mm10.fa.gz

> wget --timestamping
ftp://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/genes/mm10.refGene.gtf.gz
-O mm10.refGene.gtf.gz
> gunzip *

> cd ..
> cd fly
> wget
ftp://ftp.ensembl.org/pub/release-103/fasta/drosophila_melanogaster/dna/Drosophila_melanogaster.BDGP6.32.dna.toplevel.fa.gz

> wget
ftp://ftp.ensembl.org/pub/release-103/gtf/drosophila_melanogaster/Drosophila_melanogaster.BDGP6.32.103.gtf.gz
> gunzip *

> cd ..
> cd human
> wget
https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.fa.masked.gz

> wget
ftp://hgdownload.cse.ucsc.edu/goldenPath/hg38/bigZips/genes/hg38.refGene.gtf.gz
> gunzip *

> cd ..
> cd ecoli

```

```

> wget
ftp://ftp.ensemblgenomes.org/pub/release-51/bacteria/fasta/bacteria_0_collection/escherichia_coli_str_k_12_substr_mg1655_gca_000005845/dna/Escherichia_coli_str_k_12_substr_mg1655_gca_000005845.ASM584v2.dna.toplevel.fa.gz

> wget
ftp://ftp.ensemblgenomes.org/pub/release-51/bacteria/gtf/bacteria_0_collection/escherichia_coli_str_k_12_substr_mg1655_gca_000005845/Escherichia_coli_str_k_12_substr_mg1655_gca_000005845.ASM584v2.51.gtf.gz

```

(11) Before we get into the analysis, we still need to load up a bunch of non-coding RNA (ncRNA) files so that we can use them to filter all of the reads that belong to non-coding RNA from our ribosome profiling data. Just like with the genome and annotation files, we are going to make some directories in scratch and then download everything using wget. We are also going to use bowtie2-build to create an index for our ncRNA:

```

> cd scratch/genomes
> mkdir mouse_ncrna
> mkdir fly_ncrna
> mkdir human_ncrna
> mkdir ecoli_ncrna

> module load bowtie2

> cd fly_ncrna
> wget
ftp://ftp.ensemblgenomes.org/pub/metazoa/release-51/fasta/drosophila_melanogaster/ncrna/Drosophila_melanogaster.BDGP6.32.ncrna.fa.gz
> gunzip *
> bowtie2-build -f Drosophila_melanogaster.BDGP6.32.ncrna.fa fly_ncrna

> cd ..
> cd ecoli_ncrna
> wget
ftp://ftp.ensemblgenomes.org/pub/release-51/bacteria/fasta/bacteria_0_collection/escherichia_coli_str_k_12_substr_mg1655_gca_000005845/ncrna/Escherichia_coli_str_k_12_substr_mg1655_gca_000005845.ASM584v2.ncrna.fa.gz
> gunzip *
> bowtie2-build -f
Escherichia_coli_str_k_12_substr_mg1655_gca_000005845.ASM584v2.ncrna.fa
ecoli_ncrna

> cd ..
> cd human_ncrna
> wget
ftp://ftp.ensembl.org/pub/release-104/fasta/homo_sapiens/ncrna/Homo_sapiens.GRC
h38.ncrna.fa.gz
> gunzip *
> bowtie2-build -f Homo_sapiens.GRCh38.ncrna.fa human_ncrna

```

```

> cd ..
> cd mmus_ncrna
> wget
ftp://ftp.ensembl.org/pub/release-105/fasta/mus_musculus/ncrna/Mus_musculus.GRCm39.ncrna.fa.gz
> gunzip *
> bowtie2-build -f Mus_musculus.GRCm39.ncrna.fa mouse_ncrna

```

(12) Generate STAR genome index for all of the genomes. It is not entirely necessary to do this within a bash file for smaller genomes like the fly and E.coli genomes, but for the sake of consistency we will generate all of the genome indecis by running the bash file like the one below through sbatch. Note that this script is for the fly genome and is made such that the generate genome command is able to run in parallel.

```

#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --mem=12G
#SBATCH --cpus-per-task=10
#SBATCH --account=def-egreenbl
#SBATCH --job-name=Index_genome
#SBATCH --output=%x-%j.out
#SBATCH --mail-user=keeganfl@student.ubc.ca
#SBATCH --mail-type=END
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=FAIL

```

```

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

```

```

STAR --runThreadN 8 --runMode genomeGenerate --genomeDir . --genomeFastaFiles
Drosophila_melanogaster.BDGP6.32.dna.toplevel.fa --sjdbGTFfile
Drosophila_melanogaster.BDGP6.32.103.gtf --sjdbOverhang 100
--genomeSAindexNbases 13

```

(13) Make a directory for storing sequence data (do this in /scratch so our 1TB group limit doesn't get filled up) and for storing the output text files from running jobs. NOTE: It is very important to always create an output folder in whatever directory you are running the analysis. Failing to do this prevent the scripts from running properly.

```

> cd $HOME/scratch
> mkdir seq
> mkdir greenblatt2018
> mkdir output
> cd greenblatt2018

```

(14) You will now need to download and unzip the raw data from whatever paper you are using the data from. As an example, this file will use the data from Greenblatt and Spradling, Science, 2018. Download the raw data by submitting an array job to sbatch (Compute Canada's Job scheduler). This is as easy as running:

```

> sbatch step_14_download_ssr.sh

```

While the appropriate command list file is present in your current directory. The command list file for the Greenblatt and Spradling dataset can be found on the [GitHub page](#). Since you may need to alter how you run one of these batch jobs it is important to understand how they work.

Let's start by looking at one of the command lists. The command list for step 14 looks like this.

```
> fastq-dump --gzip SRR7132273
fastq-dump --gzip SRR7132274
fastq-dump --gzip SRR7132275
fastq-dump --gzip SRR7132276
fastq-dump --gzip SRR7132277
fastq-dump --gzip SRR7132278
fastq-dump --gzip SRR7132279
fastq-dump --gzip SRR7132280
fastq-dump --gzip SRR7132281
fastq-dump --gzip SRR7132282
fastq-dump --gzip SRR7132283
fastq-dump --gzip SRR7132284
fastq-dump --gzip SRR7132285
fastq-dump --gzip SRR7132286
fastq-dump --gzip SRR7132287
fastq-dump --gzip SRR7132288
fastq-dump --gzip SRR7132289
fastq-dump --gzip SRR7132290
fastq-dump --gzip SRR7132291
fastq-dump --gzip SRR7132292
fastq-dump --gzip SRR7132293
fastq-dump --gzip SRR7132294
fastq-dump --gzip SRR7132295
```

When you run the bash script using sbatch, it is going to assign each of these commands to a different core and run all of them in parallel.

Now let's look at one of the scripts used to run all of the commands in the command list. The script for step 14 looks like this:

```
#!/bin/bash
#SBATCH --time=01:00:00
#SBATCH --mem=1G
#SBATCH --array=1-23
#SBATCH --account=def-egreenbl
#SBATCH --job-name=Download_SSR
#SBATCH --output=output/%x-%j.out
#SBATCH --mail-user=your.email@student.ubc.ca
#SBATCH --mail-type=END
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=FAIL
```

```
# File list should contain all of the
# Commands you wish to run.
```

```
echo "Starting task $SLURM_ARRAY_TASK_ID"
Download=$(sed -n "${SLURM_ARRAY_TASK_ID}p" file_list)

# Then start the download
eval $Download
```

The first section of this script includes information to be fed into the Compute Canada's job scheduler, Slurm. Let's review what all of these commands do:

--time controls how long the scheduler gives the job to be completed. Whatever job you are doing will be automatically canceled after it has run for this amount of time, so you may need to change this time if you are adapting this code to work with different data. Generally speaking, you should err on the side of caution when choosing the time for your job.

--mem controls how much memory is available for this project.

--array specifies the number of jobs you want to run simultaneously. This is the command that will allow us to run all of the commands in our command list in parallel. Make sure that the number used for the array argument is the same as the number of commands in our list (e.g. step 14 has 23 commands so --array=1-23).

--account specifies which Compute Canada allocation this job is being run under.

--job-name specifies the name of the job that will show up in your alert emails and your output names

--output specifies where the output text files will be saved and what they will be called (these files contain information that would normally be outputted to the terminal).

--mail any command that starts with --mail is meant to specify how you will receive email alerts when the jobs you are running get completed.

(15) Once you have downloaded all of the necessary files you will need to unzip them using `step_15_unzip_SRR_files.sh`.

If you are running this protocol on the fly data, then the files you just downloaded contain the following experiments (number at end is the replicate #). Unfortunately the SRA database gave these samples labels that are out of order. RNA = mRNA-seq (poly-A selected) and RPF = ribosome profiling.

```
control_RNA_1 SRR7132278
control_RNA_2 SRR7132275
control_RNA_3 SRR7132276
```

```
control_RPF_1 SRR7132290
control_RPF_2 SRR7132291
control_RPF_3 SRR7132288
control_RPF_4 SRR7132289
```

```
Fmr1_RNA_1 SRR7132281
Fmr1_RNA_2 SRR7132282
Fmr1_RNA_3 SRR7132279
Fmr1_RNA_4 SRR7132280
Fmr1_RNA_5 SRR7132273
Fmr1_RNA_6 SRR7132274
```

```
Fmr1_RPF_1 SRR7132294
Fmr1_RPF_2 SRR7132295
```

```
Fmr1_RPF_3 SRR7132292
Fmr1_RPF_4 SRR7132293
Fmr1_RPF_5 SRR7132286
Fmr1_RPF_6 SRR7132287
Fmr1_RPF_7 SRR7132277
```

(15.5) If you are running this on the data from Sharma et al., Cell, 2019, then you will need to run the `fix_mislabeled` commands in order to correct for some mislabeled files from the mouse dataset.

(16) Ribosome profiling reads contain adapter sequences which need to be trimmed using `fastx_clipper`. This is because ribosome footprints are very short (~30bp) so reads will always contain a 3' adapter sequence (you can see that a couple samples contain a different adapter sequence).

Run `step_16_fastx_clipper.sh` using `sbatch` with the appropriate command list present in the directory (`step_16_clip_list`).

(17) Ribosome profiling experiments generally contain a large amount of ribosomal RNA (rRNA) contamination. We will align the trimmed reads to rRNA in order to bioinformatically remove the contamination, creating a new file with the extension `.norrna.fastq`.

Run `step_17_align_rna_bowtie.sh` using `sbatch` while `step_17_bowtie_list` is present in the directory.

(18) Time to align the trimmed and filtered reads to the genome and the transcriptome (yay)! We can do this using STAR:

Run `step_18_align_genome` using `sbatch` while `step_18_genome_align_list` is in the directory.

(19) Now align the RNA-seq reads - no need to trim and/or filter these reads:

Run `step_19_align_rna_seq.sh` using `sbatch` while `step_19_rna_align_list` is in the directory.

(20) All Done! You now have completed the genome alignment and are ready to move on with your analysis! The following steps just cover some basic ribosome profiling/ RNA sequencing data analysis that you can complete if you want to analyze if there are significant changes in translation efficiency between your control and mutant samples.

We can use a program called `featureCounts` to count the individual number of reads that align to specific genomic features. We are interested in reads that align to exons for RNA-seq experiments and reads that align to CDS (protein coding) regions for ribosome profiling experiments.

This step is very fast and can be run on the login node by simply copy and pasting the contents of `step_20.txt` into the file command line.

(21) Almost there! Now that we have our count files, we will next identify gene-specific changes to translation efficiency between the control and Fmr1 RNAi samples using the program `RiboDiff`. To do this, we have to put the counts output from step 20 into the proper tab-delimited format, and create an experiment template file. I have created a python file that is able to do this automatically from the data found in our count files. You can run this file by running:

```
module load scipy-stack
```

```
python process_counts.py
```

Now we have to actually run `Ribodiff`. `Ribodiff` can only run on Python 2 (unfortunately) so we are going to have to activate our `python2` environment:


```
conda activate py2
```

Then we have to actually install Ribodiff. Clone the Ribodiff package from GitHub in your \$HOME directory <https://github.com/ratschlab/RiboDiff>

CD into your newly created Ribodiff directory and then run:

```
python setup.py build
python setup.py install --user
```

This will take some time.

Once RiboDiff has finished installing you will be able to run it by using python2 to run the TE script within the RiboDiff directory while specifying the path to your raw read counts text file and your experimental design csv file. The command should look something like this:

```
python2 ~/RiboDiff/scripts/TE.py -e experimental_design.csv -c
raw_read_count.txt -o ribo_output.txt -p 1
```

The output of Ribodiff will be a list of genes, calculated fold-change in translation efficiencies, and statistical significance (padj values). By specifying `-p 1` RiboDiff will also output several graphs that describe the data.