Chapter 5: A Comprehensive Tutorial for Identification of lncRNA from RNA-Seq Data

5.1 Introduction

At one time, the belief was the genome was made up of genes and junk sequence, a very black and white way of thinking. As further investigation has been conducted, the complexities and grey areas of the genome have started to be revealed. The areas not directly made into gene products have been associated with encouraging gene expression, suppressing gene expression, creating complexes influencing what and how much of gene product is made, and many other roles. A key part of the genome involved in these gene regulation interactions is the long non-coding RNA (lncRNA) sequences: portions of the genome that are converted from DNA to RNA but never from RNA into a protein. Previous research has hypothesized these RNA function as antisense transcripts, promoter-assistance transcripts, RNA enhancers, and may control functions like X chromosome inactivation, allelic imprinting, cancer cell cycle regulation, and likely much more. [14]

Attempts have been made to generate lncRNA identification software and web tools, however, they tend to be limited to model organisms. The process can be reverse engineered from publications, but this requires extensive reading, experience with bioinformatics, and a heavy amount of trial and error. This guide intends to use plain language to create an accessible, easy-to-follow, useful tutorial for a bare bones approach

to lncRNA identification. It includes what programs are required, how to install them, the exact steps and commands needed to generate a list of candidate lncRNA transcripts, pitfalls to avoid, and helpful tips for a novice user. Although a number of tools are required, they are all publicly available for free and all steps can be run from a command line interface.

5.2 Materials

The only requirement to start this analysis is a computer running an Ubuntu Linux operating system. The steps in this tutorial can be followed on a machine running another operating system or Linux distro, however examples provided are the Ubuntu Linux based commands.

Although not required, it is recommended to obtain and use an external hard drive to contain the data, tools, and outputs of this tutorial. Using an external hard drive allows all the storage to be allocated to the analysis, as well as providing a single space for all results to be held.

It should be noted, these files are quite large. Notes will be made within this tutorial where files can be removed to save space.

5.2.1 Introduction

In this section, we will establish a working environment and install and prepare all the tools utilized within this tutorial. Each tool section is broken into 3 sub-sections. The first, a very brief explanation of the function of the tool/package to introduce the program and its role within the tutorial. This is followed by directions to download the software and move it, if applicable, to the established working area. The third and final section details any commands needed to prepare the tool for use. All information and/or commands

provided can be found on the software's site and are re-iterated within this tutorial in an attempt to be as comprehensive and useful as possible.
The provided links and instructions are intended to be as recent as possible at the time of release, however, this cannot be guaranteed. Be sure to check the website for updated links and releases.

5.2.2 Set Up

Before embarking on this endeavor, the first step is to set up a working environment free of other files to avoid confusion. If you are using an external hard drive, navigate to the drive using the following command:

```
cd [file path]
```

If you are not using an external hard drive, create a folder to contain the project:

```
mkdir [project name of choice]
cd [project name of choice]
```

Once you've successfully created your working environment, create a directory (you may have heard of this colloquially as a folder) to house your software and move into the directory.

```
mkdir software
cd software/
```

Now you're ready to begin installing the tools you'll need for this project!

5.2.3 Software Installation

Hisat2

Function: This tool functions to align reads to a reference genome.

Download:

1. Follow this link: `http://daehwankimlab.github.io/hisat2/download/#version-hisat2-221`
2. Scroll to the **Binaries** subheadings
3. Click the link following the "Linux_x86_64" label

Prepare: Use the following commands to ready the software for use:

```
mv /home/[User]/Downloads/hisat* .
unzip hisat*
```

[13]

Samtools

Function: This tool functions in multiple ways to interact with sequencing data. In this tutorial, it will function to change between file formats.

Download:

1. Follow this link: `http://www.htslib.org/download/`
2. Click the download tool for "samtools-Index*" to download the most recent version

Prepare:

1. Use the following commands to ready to software for use:

```
mv /home/[User]/Downloads/samtools* .
tar -xvf samtools*
cd samtools*
./configure
make
sudo make install
cd ..
```

[20]

ChromToUcsc

Function: This tool functions to rename chromosomes to a consistent name between reference files.

Download:

1. Use the following command to obtain the software:

```
rsync -aP \rsync://hgdownload.soe.ucsc.edu/genome/admin/exe/linux
   ↪ .x86_64/chromToUcsc ./
```

Prepare:

1. N/A

[11]

Stringtie

Function: This tool acts to assemble reads into potential transcripts.

Download:

1. Follow this link: `http://ccb.jhu.edu/software/stringtie/#install`
2. Click the "stringtie-***.Linux_x86_64.tar.gz"

Prepare:

1. Use the following commands to ready to software for use:

```
mv /home/[User]/Downloads/stringtie* .
gunzip stringtie*
tar -xvf stringtie*
```

[21]

Cufflinks

Function: This tool functions mainly for transcript assembly and differential expression analysis. In this tutorial, it will function to annotate mapped transcripts based on a reference genome.

Download:

1. Follow this link:
   `http://cole-trapnell-lab.github.io/cufflinks/install/`
2. Click the "Linux" link in the most row representing the most recent release

Prepare:

1. Use the following commands to ready to software for use:

```
mv /home/[User]/Downlods/cufflinks* .
gunzip cufflinks*
tar -xvf cufflinks*
```

[4]

lncRNA Identification Pipeline

Function: This package contains scripts written by various people and compiled in a previous project. It will function in several steps to isolate intergenic transcripts, as well as convert nucleotide sequences to protein sequences.

Download:

Use the following command to obtain the software:

```
wget https://github.com/amarceau-998/lncRNA-Identification-Pipeline/
   ↪ archive/refs/heads/main.zip
```

Prepare:

1. Use the following commands to ready to software for use:

```
unzip main.zip
```

[1]

Bedtools

Function: This package holds many functions for genomic analysis. This tutorial will only use the "getfasta" function, which will convert a bed file, a file housing genetic coordinates, into a fasta file, a file containing the sequence(s) of interest.

Download:

1. Use the following command:

```
wget https://github.com/arq5x/bedtools2/releases/download/v2
   ↪ .31.0/bedtools-2.31.0.tar.gz
```

Prepare:

1. Use the following commands to ready to software for use:

```
gunzip bedtools*
tar -xvf bedtools*
cd bedtools*
make
sudo cp bin/* /usr/local/bin/
cd ..
```

[19]

Seqtk

Function: This package mainly functions to process fasta files. The only tool used in this tutorial will retrieve the reverse complement to fasta sequences.

Download:

1. Use the following command:

```
git clone https://github.com/lh3/seqtk.git
```

Prepare:

1. Use the following commands to ready to software for use:

```
cd seqtk
make
cd ..
```

[8]

CPC2

Function: This tool assesses the coding potential of given sequences and categorizes them as either coding or non-coding.

Download:

1. Follow this link: `http://cpc2.gao-lab.org/download.php`
2. Click the “Click here” link within the ”For the version supports Python3 please click here” text
3. Click the ”source code (tar.gz)” link

Prepare:

1. Use the following commands to ready to software for use:

```
mv /home/[User]/Downloads/CPC* .
gunzip CPC*
tar -xvf CPC*
cd CPC*
cd libs/libsvm/
gunzip libsvam*
tar -xvf libsvam*
cd libsvm-*
make clean && make
cd .. -- 4 times
```

[3]

Hmmer

Function: This tool functions to compare protein sequences to a protein family database and find similarities.

Download:

1. Use the following commands:

```
wget http://eddylab.org/software/hmmer/hmmer.tar.gz
```

Prepare:

1. Use the following commands to ready to software for use:

```
tar -xvf hmmer*
cd hmmer*
./configure
make
cd ..
```

[10]

BLAST

Function: This tool functions to compare nucleotide sequences to a sequence database to find similarities.

Download:

1. Follow this link:

   `https://ftp.ncbi.nlm.nih.gov/blast/executables/LATEST/`

2. Click the link ending in "-linux.tar.gz"

Prepare:

1. Use the following commands to ready to software for use:

```
mv /home/[User]/Downloads/ncbi-blast* .
gunzip ncbi*
tar -xvf ncbi2* .
```

[16]

SRA-Toolkit

Function: This tool kit allows you to interact with the SRA database. We will only be using fastq-dump to download data from the SRA database. If you are not obtaining data from the SRA database, you do not need this tool.

Download:

1. Follow this link:

   `https://github.com/ref2/sra-tools/wiki/02.-Installing-SRA-Toolkit`

2. Click the link in the "Ubuntu" row

Prepare:

1. Use the following commands to ready to software for use:

```
mv /home/[User]/Downloads/sra* .
gunzip sra*
tar -xvf sra*
```

[9]

5.2.4 Reference Files

Introduction

In this section, we will acquire the necessary reference files for our analysis. Many files are used multiple times in several steps. This tutorial will provide a source where the necessary file(s) are likely be found; however, if you are working with a more rare species, the files may not be found at the provided link and further digging may be required to obtain the files.

Similarly to the software section, there will be a brief description of the file, a link to where you are likely to find it, instructions as to how to download the file, and if needed, any pertinent information notes.
Before beginning this section, be sure you are in your project directory.

Reference Genome(s)

Description: A general transfer format (gtf) file containing lines of information for known genetic features.

Location: `https://hgdownload.soe.ucsc.edu/downloads.html`

Instructions:

1. Select your species of interest from the drop down menus
2. Click "Genome sequence files and select annotations (2bit, GTF, GC-content, etc)" under the first subheading, this will be the most recent release for the species
3. Scroll to the bottom and click "genes/"
4. Click the links for 2 of the reference genomes to begin their downloading
5. Move the files to your current directory

```
mv /home/[User]/Downloads/[Reference genome 1]* .
mv /home/[User]/Downloads/[Reference genome 2]* .
```

Notes:

- If there is only one refence genome, that is fine! Follow the same above steps for the single genome available.

[23]

Reference Fasta

Description: A file containing nucleotide sequences and sequence identifiers for the organism of interest.

Location: `https://hgdownload.soe.ucsc.edu/downloads.html`

Instructions:

1. Select your species of interest from the drop down menus
2. Click "Genome sequence files and select annotations (2bit, GTF, GC-content, etc)" under the first subheading, this will be the most recent release for the species
3. Scroll to the bottom and click the link following this format: [species name].fa.gz
4. Move the file to your current directory

```
mv /home/[User]/Downloads/[species name]* .
gunzip *.fa
```

Notes:

- There are several files ending in variations of ".fa.gz", be sure you are downloading the correct fasta file

[23]

Chromosome Alias Files

Description: A text file containing the various chromosome naming conventions allowing for simple conversions.

Location: `https://hgdownload.soe.ucsc.edu/downloads.html`

Instructions:

1. Select your species of interest from the drop down menus
2. Click “Genome sequence files and select annotations (2bit, GTF, GC-content, etc)” under the first subheading, this will be the most recent release for the species
3. Scroll to the bottom and click the link following this format: [species name].chromAlias.txt
4. Move the file to your current directory

```
mv /home/[User]/Downloads/[species name].chromAlias.txt .
```

Notes:

- There may be a similarly named file with the “.bb” file extension, be sure you are downloading that “.txt” file
- If when clicking this link, the file does not download and instead opens as a text webpage, the information can be copied into a .txt document in your project folder

[23]

Unscaffolded Chromosome File

Description: Fasta file containing sequence information for transcripts not assigned to a chromosome.

Location: `https://hgdownload.soe.ucsc.edu/downloads.html`

Instructions:

1. Select your species of interest from the drop down menus
2. Select “Sequence data by chromosome”, it may be in a less recent release

3. Scroll down to the “chrUn*” link and click the link

4. Move the file to your current directory

```
mv /home/[User]/Downloads/chrUn* .
```

Notes:

- This can be very tricky to find

[23]

Pfam Database

Description: A database containing protein domains.

Location: `https://www.ebi.ac.uk/interpro/download/Pfam/`

Instructions:

1. Click either the “Pfam-A models” link or the download icon in the Pfam-A models row

2. Move the file to your current directory

```
mv /home/[User]/Downloads/Pfam* .
```

Notes:

- This is a large file so do not panic if it takes several minutes to download

[18]

BLAST Database

Description: A database containing genetic sequences of known genes.

Location: `https://useast.ensembl.org/index.html`

Instructions:

1. Under the ‘All genomes” header, click the drop down menu and select your species
2. Under the “Gene annotation” header, click the “Download fasta” link
3. Click the “cds” link
4. Click the file with the “.cds.all.fa.gz” file extension
5. Move the file to your current directory

```
mv /home/[User]/Downloads/*.cds.all.fa.gz .
```

Notes:

- Be sure this is the “cds” file as it contains only known genes, if you use “cdna” file, your final filtering step will not work!

[6]

5.3 Methods

5.3.1 Introduction

Now that we have installed and prepared all the necessary tools and collected the necessary reference files, we are ready to begin our analysis. To be as digestible as possible, this tutorial is broken into many small steps that are generally a single command or a small set of commands intended to complete a single step in the analysis process. Some early steps may be considered common practice when working with genetic data and may be skipped by more experienced users; however, as previously stated, this tutorial is intended to be as comprehensive as possible for beginner users.
Within each step, there will be a description of purpose of the step, a command with place holders, an example command, a description of the output files, and any applicable notes about the command and/or the files associated with it. The notes section is where you will find notes about files that can be removed to save space.

5.3.2 Tutorial

Obtain data

Purpose: To identify lncRNA, we first must have genetic data to analyze! If you already have genetic information in a .fasta or a .fastq format, this step can be skipped. If you do not, a database like the SRA database can be used to obtain publically available data. The following command(s) will download data from the SRA database.

Command:

```
software/sratoolkit*/bin/fastq-dump* --gzip --split-3 [SRR_ID]
```

Example:

```
software/sratoolkit*/bin/fastq-dump* --gzip --split-3 SRR24066292
software/sratoolkit*/bin/fastq-dump* --gzip --split-3 SRR24066293
software/sratoolkit*/bin/fastq-dump* --gzip --split-3 SRR18899314
```

Output: This command will yield the fastq file for an entry in the SRA databases based on the SRR ID that was provided. The file will be in a condensed format.

Notes:

- It is possible your data will need pre-analysis processing. Information on pre-processing can be found here: link, using the Phase 1 tutorial. This step should be handled on a case-by-case basis.
- It may be tempting to unzip the files, but this is not necessary and should not be done to preserve space.

Build index

Purpose: Index files are required for a future step aligning reads to the genome. This command will generate the necessary index files.

Command:

```
software/hisat*/hisat2-build [referencence fasta] Index
```

Example:

```
software/hisat*/hisat2-build canFam6.fa Index
```

Output: This command will create 8 files with the .ht2 suffix, these will be used in the next step.

Notes:

- From this point forward, commands will need to be ran on each sample
- Many commands can be ran in a loop manner, however that is not beginner friendly. If you are comfortable with writing and running loops, feel free to modify commands as needed for this function

Align data

Purpose: Using the previously generated index information, this step will align your data to the reference information.

Command:

- If your data is made up of pair-end sequences; that is, your files end in _.1.fq.gz and _.2.fq.gz:

```
software/hisat*/hisat2 -q -x [index file name] -1 [data name]_1.
   ↪ fq.gz -2 [data name]_2.fq.gz -S [data name]_hisat2-1.sam --
   ↪ novel-splicesite-outfile [data name].junctions --summary-
   ↪ file [data name]_hisat2-1_summary.txt
```

- If your data is made up of unpaired sequence data; that is, your file ends in .fq.gz only:

```
software/hisat*/hisat2 -q x [index file name] -U [data name].fq.
   ↪ gz -S [data name]_hisat2-1.sam --novel-splicesite-outfile [
```

```
    ↪ data name].junctions --summary-file [data name]_hisat2-1
    ↪ _summary.txt
```

Example:

```
software/hisat*/hisat2 -q -x Index -U SRR24066292.fastq.gz -S
    ↪ Sample1_hisat2-1.sam --novel-splicesite-outfile Sample1.
    ↪ junctions --summary-file Sample1_hisat2-1_summary.txt
software/hisat*/hisat2 -q -x Index -U SRR24066293.fastq.gz -S
    ↪ Sample2_hisat2-1.sam --novel-splicesite-outfile Sample2.
    ↪ junctions --summary-file Sample2_hisat2-1_summary.txt
software/hisat*/hisat2 -q -x Index -1 SRR18899314_1.fastq.gz -2
    ↪ SRR18899314_2.fastq.gz -S Sample3_hisat2-1.sam --novel-
    ↪ splicesite-outfile Sample3.junctions --summary-file
    ↪ Sample3_hisat2-1_summary.txt
```

Output:

- A ".sam" file that contains sequence alignment information
- A ".junctions" file that contains the junctions between aligned transcripts
- A ".txt" file containing the alignment statistics for the sample

Notes:

- This aligner is able to process data in several formats: fastq, fasta, and SRA database enteries. If your data is not in the fastq format, visit this page to determine the appropriate command: `http://daehwankimlab.github.io/hisat2/manual/`

- There are other alignment tools available, if you have a preferred alignment tool, ensure you have a ".junctions" file output before continuing.
- To save space, the ".sam" file generated by this step can be removed.

Secondary alignment

Purpose: Using the previously generated index information and the junctions found in step 3, this step will align your data to the reference information with more accuracy.

Command:

If your data is made up of pair-end sequences; that is, your files end in _.1.fq.gz and _.2.fq.gz:

```
software/hisat*/hisat2 -q -x [index file name] -1 [data name]_1.fq.gz
   ↪ -2 [data name]_2.fq.gz -S [data name]_hisat2-2.sam --novel-
   ↪ splicesite-infile [data name].junctions --summary-file [data
   ↪ name]_hisat2-2_summary.txt
```

If your data is made up of unpaired sequence data; that is, your file ends in .fq.gz only:

```
software/hisat*/hisat2 -q -x [index file name] -U [data name].fq.gz -S
   ↪  [data name]_hisat2-2.sam --novel-splicesite-infile [data name
   ↪ ].junctions --summary-file [data name]_hisat2-2_summary.txt
```

Example:

```
software/hisat*/hisat2 -q -x Index -U SRR24066292.fastq.gz -S
   ↪ Sample1_hisat2-2.sam --novel-splicesite-infile Sample1.
   ↪ junctions --summary-file Sample1_hisat2-2_summary.txt
```

```
software/hisat*/hisat2 -q -x Index -U SRR24066293.fastq.gz -S
    ↪ Sample2_hisat2-2.sam --novel-splicesite-infile Sample2.
    ↪ junctions --summary-file Sample2_hisat2-2_summary.txt
software/hisat*/hisat2 -q -x Index -1 SRR18899314_1.fastq.gz -2
    ↪ SRR18899314_2.fastq.gz -S Sample3_hisat2-2.sam --novel-
    ↪ splicesite-infile Sample3.junctions --summary-file
    ↪ Sample3_hisat2-2_summary.txt
```

Output:

- A ".sam" file that contains sequence alignment information
- A ".txt" file containing the alignment statistics for the sample

Notes:

- This command looks and operates in a very similar way to step 3; however, it is not the same command. The output name has been changed to reflect that it is the second iteration of alignment and the "–novel-splicesite-outfile" option has been changed to "–novel-splicesite-infile"
- As previously mentioned, you are free to use any preferred aligner tool if you have one. In this step, be sure the tool outputs a ".sam" file
- After this step, you can remove the ".junction" files, they are not large files but space can be precious

Convert SAM file to BAM file

Purpose: This step functions to convert the sequence alignment files to a compressed version for further analysis.

Command:

```
software/samtools*/samtools view -S -b [data name]_hisat2-2.sam > [
    ↪ data name].bam
```

Example:

```
software/samtools*/samtools view -S -b Sample1_hisat2-2.sam > Sample1.
    ↪ bam
software/samtools*/samtools view -S -b Sample2_hisat2-2.sam > Sample2.
    ↪ bam
software/samtools*/samtools view -S -b Sample3_hisat2-2.sam > Sample3.
    ↪ bam
```

Output:

- A ".bam" file containing the same information as the ".sam" file in a more condensed format

Notes:

- The ".sam" file can now be removed
- You will not be able to natively open the ".bam" file

Sort BAM file

Purpose: This step sorts the ".bam" file by genomic coordinates to avoid errors going forward.

Command:

```
software/samtools*/samtools sort [data name].bam > [data name]_sorted.
   ↪ bam
```

Example:

```
software/samtools*/samtools sort Sample1.bam > Sample1_sorted.bam
software/samtools*/samtools sort Sample2.bam > Sample2_sorted.bam
software/samtools*/samtools sort Sample3.bam > Sample3_sorted.bam
```

Output:

- A ".bam" file containing the same information as the input file, sorted by location

Notes:

- You can remove the original ".bam" file
- As previously stated, you cannot natively open the ".bam" file

Standardize reference genome naming conventions

Purpose: Different databases use different chromosomal naming conventions; this step functions to standardize those names to avoid errors. If you only have 1 reference genome, perform only one of the commands.

Command:

```
software/chromToUcsc -a [chromAlias.txt] -i [reference 1].gtf -o ref1.
   ↪ gtf
software/chromToUcsc -a [chromAlias.txt] -i [reference 2].gtf -o ref2.
   ↪ gtf
```

Example:

```
software/chromToUcsc -a canFam6.chromAlias.txt -i refGene.gtf.gz -o
    ↪ ref1.gtf
software/chromToUcsc -a canFam6.chromAlias.txt -i ncbiRefSeq.gtf.gz.
    ↪ gtf -o ref2.gtf
```

Output:

- 2 modified ".gtf" files with a consistent chromosome naming convention

Notes:

- You can remove the original reference files to avoid confusion
- From this point forward, if you only have one reference, follow the commands for one of the reference genomes and disregard the other

Transcript Assembly

Purpose: This step will take the previously generated alignment data and create a transcript assembly. This tutorial uses 2 reference genomes to improve accuracy.

Command:

```
software/stringtie*/stringtie [data name]_sorted.bam -o [data name]
    ↪ _ref1.gtf -G [standardized reference genome 1].gtf -A [data
    ↪ name]_ref1.tab
software/stringtie*/stringtie [data name]_sorted.bam -o [data name]
    ↪ _ref2.gtf -G [standardized reference genome 2].gtf -A [data
    ↪ name]_ref2.tab
```

Example:

```
software/stringtie*/stringtie Sample1_sorted.bam -o Sample1_ref1.gtf -
   ↪ G ref1.gtf -A Sample1_ref1.tab
software/stringtie*/stringtie Sample1_sorted.bam -o Sample1_ref2.gtf -
   ↪ G ref2.gtf -A Sample1_ref2.tab
software/stringtie*/stringtie Sample2_sorted.bam -o Sample2_ref1.gtf -
   ↪ G ref1.gtf -A Sample2_ref1.tab
software/stringtie*/stringtie Sample2_sorted.bam -o Sample2_ref2.gtf -
   ↪ G ref2.gtf -A Sample2_ref2.tab
software/stringtie*/stringtie Sample3_sorted.bam -o Sample3_ref1.gtf -
   ↪ G ref1.gtf -A Sample3_ref1.tab
software/stringtie*/stringtie Sample3_sorted.bam -o Sample3_ref2.gtf -
   ↪ G ref2.gtf -A Sample3_ref2.tab
```

Output:

- 2 ".gtf" files containing the genomic features of your data based on the reference genome
- 2 ".tab" files containing expression data for transcripts within your data

Notes:

- If there is a naming convention for your reference genome(s) that you prefer, feel free to use that in place of "_ref1" and/or "_ref2"
- You can remove the "_sorted.bam" files now if you choose

Split files

Purpose: In order to avoid errors and loss of information, files should be split to represent known chromosomes and unknown chromosomes.

Command:

```
grep 'chrUn*' [data name]_ref1.gtf > [data name]_ChrUn_ref1.gtf
grep -v 'chrUn*' [data name]_ref1.gtf > [data name]_ChrKnown_ref1.gtf
grep 'chrUn*' [data name]_ref2.gtf > [data name]_ChrUn_ref2.gtf
grep -v 'chrUn*' [data name]_ref2.gtf > [data name]_ChrKnown_ref2.gtf
```

Example:

```
grep 'chrUn' Sample1_ref1.gtf > Sample1_ChrUn_ref1.gtf
grep -v 'chrUn*' Sample1_ref1.gtf > Sample1_ChrKnown_ref1.gtf
grep 'chrUn*' Sample1_ref2.gtf > Sample1_ChrUn_ref2.gtf
grep -v 'chrUn*' Sample1_ref2.gtf > Sample1_ChrKnown_ref2.gtf
grep 'chrUn*' Sample2_ref1.gtf > Sample2_ChrUn_ref1.gtf
grep -v 'chrUn*' Sample2_ref1.gtf > Sample2_ChrKnown_ref1.gtf
grep 'chrUn*' Sample2_ref2.gtf > Sample2_ChrUn_ref2.gtf
grep -v 'chrUn*' Sample2_ref2.gtf > Sample2_ChrKnown_ref2.gtf
grep 'chrUn*' Sample3_ref1.gtf > Sample3_ChrUn_ref1.gtf
grep -v 'chrUn*' Sample3_ref1.gtf > Sample3_ChrKnown_ref1.gtf
grep 'chrUn*' Sample3_ref2.gtf > Sample3_ChrUn_ref2.gtf
grep -v 'chrUn*' Sample3_ref2.gtf > Sample3_ChrKnown_ref2.gtf
```

Output:

- A ".gtf" file containing transcript information for unknown chromosomes based on the ensembl reference genome
- A ".gtf" file containing transcript information for known chromosomes based on the ensembl reference genome
- A ".gtf" file containing transcript information for unknown chromosomes based on the ref2 reference genome
- A ".gtf" file containing transcript information for known chromosomes based on the ref2 reference genome

Notes:

- This set of commands will be used several times

Cuffcompare

Purpose: Cuffcompare will function to annotate identified transcripts and assign a class code that can be utilized for filtering.

Command:

```
software/cufflink*/cuffcompare [data name]_ChrUn_ref1.gtf -s [unknown
    ↪ chromosome].fa -r ref1.gtf -o [data name]
    ↪ _ChrUn_ref1_Cuffcompare
software/cufflink*/cuffcompare [data name]_ChrKnown_ref1.gtf -s [
    ↪ reference fasta file].fa -r ref1.gtf -o [data name]
    ↪ _ChrKnown_ref1_Cuffcompare
software/cufflink*/cuffcompare [data name]_ChrUn_ref2.gtf -s [unknown
    ↪ chromosome].fa -r ref2.gtf -o [data name]
    ↪ _ChrUn_ref2_Cuffcompare
```

```
software/cufflink*/cuffcompare [data name]_ChrKnown_ref2.gtf -s [
   ↪ reference fasta file].fa -r ref2.gtf -o [data name]
   ↪ _ChrKnown_ref2_Cuffcompare
```

Example:

```
software/cufflink*/cuffcompare Sample1_ChrUn_ref1.gtf -s chrUn.fa -r
   ↪ ref1.gtf -o Sample1_ChrUn_ref1_Cuffcompare
software/cufflink*/cuffcompare Sample1_ChrKnown_ref1.gtf -s canFam6.fa
   ↪  -r ref1.gtf -o Sample1_ChrKnown_ref1_Cuffcompare
software/cufflink*/cuffcompare Sample1_ChrUn_ref2.gtf -s chrUn.fa -r
   ↪ ref2.gtf -o Sample1_ChrUn_ref2_Cuffcompare
software/cufflink*/cuffcompare Sample1_ChrKnown_ref2.gtf -s canFam6.fa
   ↪  -r ref2.gtf -o Sample1_ChrKnown_ref2_Cuffcompare
software/cufflink*/cuffcompare Sample2_ChrUn_ref1.gtf -s chrUn.fa -r
   ↪ ref1.gtf -o Sample2_ChrUn_ref1_Cuffcompare
software/cufflink*/cuffcompare Sample2_ChrKnown_ref1.gtf -s canFam6.fa
   ↪  -r ref1.gtf -o Sample2_ChrKnown_ref1_Cuffcompare
software/cufflink*/cuffcompare Sample2_ChrUn_ref2.gtf -s chrUn.fa -r
   ↪ ref2.gtf -o Sample2_ChrUn_ref2_Cuffcompare
software/cufflink*/cuffcompare Sample2_ChrKnown_ref2.gtf -s canFam6.fa
   ↪  -r ref2.gtf -o Sample2_ChrKnown_ref2_Cuffcompare
software/cufflink*/cuffcompare Sample3_ChrUn_ref1.gtf -s chrUn.fa -r
   ↪ ref1.gtf -o Sample3_ChrUn_ref1_Cuffcompare
software/cufflink*/cuffcompare Sample3_ChrKnown_ref1.gtf -s canFam6.fa
   ↪  -r ref1.gtf -o Sample3_ChrKnown_ref1_Cuffcompare
```

```
software/cufflink*/cuffcompare Sample3_ChrUn_ref2.gtf -s chrUn.fa -r
    ↪ ref2.gtf -o Sample3_ChrUn_ref2_Cuffcompare
software/cufflink*/cuffcompare Sample3_ChrKnown_ref2.gtf -s canFam6.fa
    ↪  -r ref2.gtf -o Sample3_ChrKnown_ref2_Cuffcompare
```

Output:

- A ".combined.gtf" file for each reference, a gtf file containing all the transfrags
- A ".loci" file for each reference, a file containing points of interest
- A ".refmap" file for each reference, a file assigning Cufflink matches to each transcript
- A "tmap" file for each reference, a file listing closely matched Cufflink transcripts
- A ".stats" file for each reference, a file detailing the queries and matches within each comparison
- A ".tracking" file for each comparison between transcripts

Notes:

- Note that there is not a file extension on the output flag (-o), this is intentional. This command produces several output files, they will all have the same name with varying file extensions.
- By this point, the file names are getting long. This will be rectified in the next step
- All "_CuffCompare" files except the ".combined.gtf" file can be removed
- If any ".fa.fai" files were generated by this step, they can also be removed

Reconstitute files

Purpose: This step is dual purposed, par down the commands needed for analysis and create an opportunity to shorten file names back to a reasonable length.

Command:

```
cat [data name]_ChrUn_ref1_Cuffcompare.combined.gtf [data name]
    ↪ _ChrKnown_ref1_Cuffcompare.combined.gtf > [data name]
    ↪ _ref1_Cuffcompare_Out.gtf
cat [data name]_ChrUn_ref2_Cuffcompare.combined.gtf [data name]
    ↪ _ChrKnown_ref2_Cuffcompare.combined.gtf > [data name]
    ↪ _ref2_Cuffcompare_Out.gtf
```

Example:

```
cat Sample1_ChrUn_ref1_Cuffcompare.combined.gtf
    ↪ Sample1_ChrKnown_ref1_Cuffcompare.combined.gtf >
    ↪ Sample1_ref1_Cuffcompare_Out.gtf
cat Sample1_ChrUn_ref2_Cuffcompare.combined.gtf
    ↪ Sample1_ChrKnown_ref2_Cuffcompare.combined.gtf >
    ↪ Sample1_ref2_Cuffcompare_Out.gtf
cat Sample2_ChrUn_ref1_Cuffcompare.combined.gtf
    ↪ Sample2_ChrKnown_ref1_Cuffcompare.combined.gtf >
    ↪ Sample2_ref1_Cuffcompare_Out.gtf
cat Sample2_ChrUn_ref2_Cuffcompare.combined.gtf
    ↪ Sample2_ChrKnown_ref2_Cuffcompare.combined.gtf >
    ↪ Sample2_ref2_Cuffcompare_Out.gtf
```

```
cat Sample3_ChrUn_ref1_Cuffcompare.combined.gtf
    ↪ Sample3_ChrKnown_ref1_Cuffcompare.combined.gtf >
    ↪ Sample3_ref1_Cuffcompare_Out.gtf
cat Sample3_ChrUn_ref2_Cuffcompare.combined.gtf
    ↪ Sample3_ChrKnown_ref2_Cuffcompare.combined.gtf >
    ↪ Sample3_ref2_Cuffcompare_Out.gtf
```

Output:

- A ".gtf" file containing the cuffcompare annotated transcripts for both known and unknown chromosomes for each reference genome

Notes:

- The ".combined.gtf" files can be removed at this point, this will eliminate confusion
- You should also remove the "ChrUn" and "ChrKnown" files generated from step 9

Intergenic list

Purpose: Using the labels assigned to transcripts with Cuffcompare, we will now generate a list of only those transcripts marked as intergenic. If a transcript is annotated as being anything else, it can be assumed to not be a lncRNA as it associates with a known gene in some way.

Command:

```
perl software/lncRNA-Identification-Pipeline-main/
    ↪ get_intergenicLoci_list.pl -g1 [data name]_ref1_Cuffcompare_Out
    ↪ .gtf -g2 [data name]_ref2_Cuffcompare_Out.gtf -o [data name]
    ↪ _intergenic.txt
```

Example:

```
perl software/lncRNA-Identification-Pipeline-main/
   ↪ get_intergenicLoci_list.pl -g1 Sample1_ref1_Cuffcompare_Out.gtf
   ↪  -g2 Sample1_ref2_Cuffcompare_Out.gtf -o Sample1_intergenic.txt
perl software/lncRNA-Identification-Pipeline-main/
   ↪ get_intergenicLoci_list.pl -g1 Sample2_ref1_Cuffcompare_Out.gtf
   ↪  -g2 Sample2_ref2_Cuffcompare_Out.gtf -o Sample2_intergenic.txt
perl software/lncRNA-Identification-Pipeline-main/
   ↪ get_intergenicLoci_list.pl -g1 Sample3_ref1_Cuffcompare_Out.gtf
   ↪  -g2 Sample3_ref2_Cuffcompare_Out.gtf -o Sample3_intergenic.txt
```

Output:

- A ".txt" file listing only the transcripts that are annotated as intergenic

Notes:

- Both reference genomes are used in this step to increase accuracy
- If you only have 1 reference genome, supply the cuffcompare output for both -g1 and -g2

Generate intergenic GTF

Purpose: Using the list of intergenic transcripts, this step will filter the previous GTF files to reflect only intergenic transcripts for further filtering.

Command:

```
perl software/lncRNA-Identification-Pipeline-main/get_intergenicGTF.pl
    ↪  -g [data name]_ref1_Cuffcompare_Out.gtf -l [data name]
    ↪ _intergenic.txt -o [data name]_ref1_intergenic.gtf
perl software/lncRNA-Identification-Pipeline-main/get_intergenicGTF.pl
    ↪  -g [data name]_ref2_Cuffcompare_Out.gtf -l [data name]
    ↪ _intergenic.txt -o [data name]_ref2_intergenic.gtf
```

Example:

```
perl software/lncRNA-Identification-Pipeline-main/get_intergenicGTF.pl
    ↪  -g Sample1_ref1_CuffCompare_Out.gtf -l Sample1_intergenic.txt
    ↪ -o Sample1_ref1_intergenic.gtf
perl software/lncRNA-Identification-Pipeline-main/get_intergenicGTF.pl
    ↪  -g Sample1_ref2_CuffCompare_Out.gtf -l Sample1_intergenic.txt
    ↪ -o Sample1_ref2_intergenic.gtf
perl software/lncRNA-Identification-Pipeline-main/get_intergenicGTF.pl
    ↪  -g Sample2_ref1_CuffCompare_Out.gtf -l Sample2_intergenic.txt
    ↪ -o Sample2_ref1_intergenic.gtf
perl software/lncRNA-Identification-Pipeline-main/get_intergenicGTF.pl
    ↪  -g Sample2_ref2_CuffCompare_Out.gtf -l Sample2_intergenic.txt
    ↪ -o Sample2_ref2_intergenic.gtf
perl software/lncRNA-Identification-Pipeline-main/get_intergenicGTF.pl
    ↪  -g Sample3_ref1_CuffCompare_Out.gtf -l Sample3_intergenic.txt
    ↪ -o Sample3_ref1_intergenic.gtf
perl software/lncRNA-Identification-Pipeline-main/get_intergenicGTF.pl
    ↪  -g Sample3_ref2_CuffCompare_Out.gtf -l Sample3_intergenic.txt
```

```
    ↪ -o Sample3_ref2_intergenic.gtf
```

Output:

- 2“.gtf” files, one for each reference genome, containing only transcripts deemed intergenic

Notes:

- Feel free to remove the ”CuffCompare_Out” files at this point

Generate a summary file

Purpose: A GTF file contains a lot of information and can be very overwhelming. This step will generate a more streamlined file summarizing the intergenic transcripts in the sample.

Command:

```
perl software/lncRNA-Identification-Pipeline-main/summary_gtf.pl -i [
    ↪ data name]_ref1_intergenic.gtf -o [data name]
    ↪ _ref1_intergenic_summary.txt
perl software/lncRNA-Identification-Pipeline-main/summary_gtf.pl -i [
    ↪ data name]_ref2_intergenic.gtf -o [data name]
    ↪ _ref2_intergenic_summary.txt
```

Example:

```
perl software/lncRNA-Identification-Pipeline-main/summary_gtf.pl -i
    ↪ Sample1_ref1_intergenic.gtf -o Sample1_ref1_intergenic_summary.
    ↪ txt
```

```
perl software/lncRNA-Identification-Pipeline-main/summary_gtf.pl -i
    ↪ Sample1_ref2_intergenic.gtf -o Sample1_ref2_intergenic_summary.
    ↪ txt
perl software/lncRNA-Identification-Pipeline-main/summary_gtf.pl -i
    ↪ Sample2_ref1_intergenic.gtf -o Sample2_ref1_intergenic_summary.
    ↪ txt
perl software/lncRNA-Identification-Pipeline-main/summary_gtf.pl -i
    ↪ Sample2_ref2_intergenic.gtf -o Sample2_ref2_intergenic_summary.
    ↪ txt
perl software/lncRNA-Identification-Pipeline-main/summary_gtf.pl -i
    ↪ Sample3_ref1_intergenic.gtf -o Sample3_ref1_intergenic_summary.
    ↪ txt
perl software/lncRNA-Identification-Pipeline-main/summary_gtf.pl -i
    ↪ Sample3_ref2_intergenic.gtf -o Sample3_ref2_intergenic_summary.
    ↪ txt
```

Output:

- 2 ".txt" files summarizing the intergenic transcripts found in the sample, one for each reference genome

Notes:

- It is recommended to retain the intergenic GTF files for your sample for future reporting purposes. The note section of this tutorial will provide examples of potential analyses to report findings.

Filter by size

Purpose: Since lncRNA by definitoin are longer than 200 basepairs in length, one filtering step needed is to remove those transcripts shorter than 200 bases. This step will perform that.

Command:

```
awk '(NR==1) || ($8 > 199) ' [data name]_ref1_intergenic_summary.txt >
   ↪  [data name]_ref1_size.txt
awk '(NR==1) || ($8 > 199) ' [data name]_ref2_intergenic_summary.txt >
   ↪  [data name]_ref2_size.txt
```

Example:

```
awk '(NR==1) || ($8 > 199) ' Sample1_ref1_intergenic_summary.txt >
   ↪ Sample1_ref1_size.txt
awk '(NR==1) || ($8 > 199) ' Sample1_ref2_intergenic_summary.txt >
   ↪ Sample1_ref2_size.txt
awk '(NR==1) || ($8 > 199) ' Sample2_ref1_intergenic_summary.txt >
   ↪ Sample2_ref1_size.txt
awk '(NR==1) || ($8 > 199) ' Sample2_ref2_intergenic_summary.txt >
   ↪ Sample2_ref2_size.txt
awk '(NR==1) || ($8 > 199) ' Sample3_ref1_intergenic_summary.txt >
   ↪ Sample3_ref1_size.txt
awk '(NR==1) || ($8 > 199) ' Sample3_ref2_intergenic_summary.txt >
   ↪ Sample3_ref2_size.txt
```

Output:

- 2 ".txt" files, one for each reference, containing intergenic transcripts that are longer than 200 basepairs in length.

Notes:

- N/A

Convert file format

Purpose: The summary text file contains lots of useful information for summarizing findings and characteristics, but there is a lot of extraneous information that will not be needed for continued analysis. This step will reduce the file to just the genomic coordinates in a ".bed" file.

Command:

```
awk '{print $3"\t"$5"\t"$6}' [data name]_ref1_size.txt > [data name]
   ↪ _ref1_size.bed
awk '{print $3"\t"$5"\t"$6}' [data name]_ref2_size.txt > [data name]
   ↪ _ref2_size.bed
```

Example:

```
awk '{print $3"\t"$5"\t"$6}' Sample1_ref1_size.txt > Sample1_ref1_size
   ↪ .bed
awk '{print $3"\t"$5"\t"$6}' Sample1_ref2_size.txt > Sample1_ref2_size
   ↪ .bed
awk '{print $3"\t"$5"\t"$6}' Sample2_ref1_size.txt > Sample2_ref1_size
   ↪ .bed
awk '{print $3"\t"$5"\t"$6}' Sample2_ref2_size.txt > Sample2_ref2_size
   ↪ .bed
```

```
awk '{print $3"\t"$5"\t"$6}' Sample3_ref1_size.txt > Sample3_ref1_size
   ↪ .bed
awk '{print $3"\t"$5"\t"$6}' Sample3_ref2_size.txt > Sample3_ref2_size
   ↪ .bed
```

Output:

- 2 ".bed" files containing the genetic coordinates of intergenic transcripts over 200 base pairs in length.

Notes

- N/A

Combine samples

Purpose: You are most likely working with several samples, maybe even many samples for different condition types! This step serves to generate a list of intergenic transcripts for each condition. The following command will need to be run for each collection of samples.

Command:

```
sort -u [data name]_ref1_size.bed ... [data name]_ref1_size.bed [data
   ↪ name]_ref2_size.bed ... [data name]_ref2_size.bed ... > [
   ↪ condition name].bed
```

Example:

```
sort -u Sample1_ref1_size.bed Sample1_ref2_size.bed Sample2_ref1_size.
   ↪ bed Sample2_ref2_size.bed Sample3_ref1_size.bed
   ↪ Sample3_ref2_size.bed > All.bed
```

Output:

- A ".bed" file for each condition, containing unique intergenic transcripts over 200 bases in length

Notes:

- Be sure you include both reference genomes in this command, this is the point where they combine
- From this point forward, all commands will need to be ran on each condition

Split bed file

Purpose: In order to avoid errors and loss of information, files should be split to represent known chromosomes and unknown chromosomes.

Command:

```
grep 'chrUn*' [condition name].bed > [condition name]_ChrUn.bed
grep -v 'chrUn*' [condition name].bed > tmp1
sort -rk2 tmp1 > tmp2
tail -n+2 tmp > [condition name]_ChrKnown.bed
rm tmp
```

Example:

```
grep 'chrUn*' All.bed > All_ChrUn.bed
grep -v 'chrUn*' All.bed > tmp1
sort -rk2 tmp1 > tmp2
tail -n+2 tmp2 > All_ChrKnown.bed
rm tmp*
```

Output:

- A ".bed" file containing coordinates for intergenic transcripts over 200 bases in length located on known chromosomes
- A ".bed" files containing coordinates for intergenic transcripts over 200 bases in length located on unknown chromosomes

Notes:

- This command is very similar to the command in step 9, the addition of "tail -n +2" is necessary to ensure the file format matches what further tools expect

Obtain fasta file

Purpose: A further filtering step requires the nucleotide sequences of interest for analysis. This step functions to use the BED files previously generated to isolate nucleotide sequences of interest.

Command:

```
bedtools getfasta -fi [unknown chromosome].fa -bed [condition name]
   ↪ _ChrUn.bed -fo [condition name]_ChrUn.fasta
bedtools getfasta -fi [reference fasta].fa -bed [condition name]
   ↪ _ChrKnown.bed -fo [condition name]_ChrKnown.fasta
```

Example:

```
bedtools getfasta -fi chrUn.fa -bed All_ChrUn.bed -fo All_ChrUn.fasta
bedtools getfasta -fi canFam6.fa -bed All_ChrKnown.bed -fo
   ↪ All_ChrKnown.fasta
```

Output:

- A ".fasta" file containing nucleotide sequences for intergenic transcripts over 200 bases in length located on known chromosomes
- A ".fasta" files containing nucleotide sequences for intergenic transcripts over 200 bases in length located on unknown chromosomes

Notes:

- There is no need to include "software/" in this command
- If any ".fai" files were generated, they can be removed

Reconstitute fasta file

Purpose: This step functions to reconstitute fasta files generated in step 19 to represent all query sequences in a single file. This avoids confusion.

Command:

```
cat [condition name]_ChrUn.fasta [condition name]_ChrKnown.fasta > [
    ↪ condition name].fasta
```

Example:

```
cat All_ChrUn.fasta All_ChrKnown.fasta > All.fasta
```

Output:

- A fasta file with nucleotide sequences for all intergenic transcripts over 200 base pairs in length on both known and unknown chromosomes

Notes:

- You can now remove the known and unknown fasta files generated in step 18 ([condition name]_ChrUn.fasta and [condition name]_ChrKnown.fasta)

Find reverse transcript

Purpose: To be as thorough as possible, the reverse compliment of each sequence of interest also needs to be analyzed for coding potential in a later step. This step serves to generate the reverse compliment sequence for all sequences of interest.

Command:

```
software/seqtk*/seqtk seq -r [condition name].fasta > [condition name]
   ↪ _RC.fasta
```

Example:

```
software/seqtk*/seqtk seq -r All.fasta > All_RC.fasta
```

Output:

- A fasta file with the reverse compliment to nucleotide sequences for all intergenic transcripts over 200 base pairs in length on both known and unknown chromosomes

Notes:

- N/A

Coding Potential Calculator

Purpose: Created to gauge the coding potential of transcripts based on sequence features like open reading frames, the coding potential calculator classifies provided transcripts as either coding or noncoding. In this step, sequences of interest and their reverse complement will be marked as either coding or noncoding. Given the nature of lncRNA, this will allow for removal of coding transcripts as they cannot be lncRNA.

Command:

```
software/CPC*/bin/CPC2.py -i [condition name].fasta -o [condition name
    ↪ ]_CPC
software/CPC*/bin/CPC2.py -i [condition name]_RC.fasta -o [condition
    ↪ name]_RC_CPC
```

Example:

```
software/CPC*/bin/CPC2.py -i All.fasta -o All_CPC
software/CPC*/bin/CPC2.py -i All_RC.fasta -o All_RC_CPC
```

Output:

- A ".txt" file containing the identifier, the coding potential score, and the classification (coding or noncoding) for each transcript of interest
- A ".txt" file containing the identifier, the coding potential score, and the classification (coding or noncoding) for the reverse complement of each transcript of interest

Notes:

- When calling the CPC2.py command, be sure to remember the /bin/ section of the path name
- You can remove the fasta files generated by step 19 and step 20
- This is a python command. If you need to troubleshoot an error, this information is critical

Coding Potential Filtration

Purpose: Now that transcripts have been labeled as coding or noncoding, you can eliminate those deemed to be coding. That is the function of this step.

Command:

```
awk '$8 == "noncoding"' [condition name]_CPC.txt > [condition name]_NC
   ↪ .txt
awk '$8 =="noncoding"' [condition name]_RC_CPC.txt > [condition name]
   ↪ _RC_NC.txt
```

Example:

```
awk '$8 == "noncoding"' All_CPC.txt > All_NC.txt
awk '$8 =="noncoding"' All_RC_CPC.txt > All_RC_NC.txt
```

Output:

- 2 ".txt" files containing entries deemed "noncoding" from the forward transcripts of interest and the reverse compliment to transcripts of interest

Notes:

- In this command, there is both a single and a double quotation mark, be careful not to miss either

Generate noncoding lists

Purpose: The Coding Potential Calculator provides lots of information, however not all of it is necessary for continuing your analysis. This step pars down the noncoding entries to only contain the information required.

Command:

```
awk '{print $1}' [condition name]_NC.txt > [condition name]_NC.list
awk '{print $1}' [condition name]_RC_NC.txt > [condition name]_RC_NC.
   ↪ list
```

Example:

```
awk '{print $1}' All_NC.txt > All_NC.list
awk '{print $1}' All_RC_NC.txt > All_RC_NC.list
```

Output:

- 2 text based files containing only the identifiers for noncoding transcripts

Notes:

- The file extension of these files is non-traditional, these are an intermediate step that will be combined into a more succinct list in the next step

Finalize noncoding list

Purpose: This step will combine the results from the forward and reverse compliment analysis into a consolidated list of intergenic, noncoding transcripts that are over 200 base pairs in length.

Command:

```
sort -u [condition name]*.list >> [condtion name]_NC.txt
```

Example:

```
sort -u All*.list >> All_NC.txt
```

Output:

- A ".txt" file containing the unique identifies of noncoding, intergenic transcripts over 200 base pairs in length

Notes:

- You can remove both intermediate files with the file extension ".list"

Convert file

Purpose: Currently the file containing transcripts being investigated holds chromosome identifiers in a "Chr:start-end" format. This information needs to be converted to a BED file format for further use. This step acts to change the format of the file to match what's needed.

Command:

```
sed "s,:,\t,g" [condition name]_NC.txt >> [condition name]_NC.bed
sed -i "s,-,\t,g" [condition name]_NC.bed
```

Example:

```
sed "s,:,\t,g" All_NC.txt >> All_NC.bed
sed -i "s,-,\t,g" All_NC.bed
```

Output:

- A ".bed" file containing the genetic coordinates of noncoding, intergenic transcripts over 200 base pairs in length

Notes:

- The output file generated by this step has the same name as the input file, note the change in file extension
- There is no output file indicated in the second step, this is not a mistake. The "-i" flag leads to changes to the file being made in place.

Split bed file

Purpose: In order to avoid errors and loss of information, files should be split to represent known chromosomes and unknown chromosomes.

Command:

```
grep 'chrUn*' [condition name]_NC.bed > [condition name]_NC_ChrUn.tmp
grep -v 'chrUn*' [condition name]_NC.bed | tail -n+2 > [condition name
    ↪ ]_NC_ChrKnown.tmp
awk {'print $1"\t"$2"\t"$3'} [condition name]_NC_ChrUn.tmp > [
    ↪ condition name]_NC_ChrUn.bed
awk {'print $1"\t"$2"\t"$3'} [condition name]_NC_ChrKnown.tmp > [
    ↪ condition name]_NC_ChrKnown.bed
```

Example:

```
grep 'chrUn*' All_NC.bed > All_NC_ChrUn.tmp
grep -v 'chrUn*' All_NC.bed | tail -n+2 > All_NC_ChrKnown.tmp
awk {'print $1"\t"$2"\t"$3'} All_NC_ChrUn.tmp > All_NC_ChrUn.bed
awk {'print $1"\t"$2"\t"$3'} All_NC_ChrKnown.tmp > All_NC_ChrKnown.bed
```

Output:

- A ".bed" file containing coordinates for intergenic transcripts over 200 bases in length located on known chromosomes
- A ".bed" files containing coordinates for intergenic transcripts over 200 bases in length located on unknown chromosomes

Notes:

- This is similar to step 19 with an additional print command to clean up the bed files.
- The ".tmp" files can be removed

Obtain fasta file

Purpose: This step functions to use the BED files previously generated to isolate nucleotide sequences of interest. These sequences will be converted to the protein sequence for a future filtering step.

Command:

```
bedtools getfasta -fi [unknown chromosome].fa -bed [condition name]
   ↪ _NC_ChrUn.bed
-fo [condition name]_NC_ChrUn.fasta
```

```
bedtools getfasta -fi [reference fasta].fa -bed [condition name]
    ↪ _NC_ChrKnown.bed -fo [condition name]_NC_ChrKnown.fasta
```

Example:

```
bedtools getfasta -fi chrUn.fa -bed All_NC_ChrUn.bed -fo All_NC_ChrUn.
    ↪ fasta
bedtools getfasta -fi canFam6.fa -bed All_NC_ChrKnown.bed -fo
    ↪ All_NC_ChrKnown.fasta
```

Output:

- A ".fasta" file containing nucleotide sequences for noncoding, intergenic transcripts over 200 bases in length located on known chromosomes
- A ".fasta" files containing nucleotide sequences for noncoding, intergenic transcripts over 200 bases in length located on unknown chromosomes

Notes:

- There is no need to include "software/" in this command
- Aside from the input and output file, this command is identical to step 19
- If any ".fai" files were generated during this step, they can be removed

Reconstitute fasta file

Purpose: This step functions to reconstitute fasta files generated in step 28 to represent all query sequences in a single file. This avoids confusion.

Command:

```
cat [condition name]_NC_ChrUn.fasta [condition name]_NC_ChrKnown.fasta
   ↪  > [condition name]_NC.fasta
```

Example:

```
cat All_NC_ChrUn.fasta All_NC_ChrKnown.fasta > All_NC.fasta
```

Output:

- A fasta file with nucleotide sequences for all noncoding, intergenic transcripts over 200 base pairs in length on both known and unknown chromosomes

Notes:

- You can now remove the known and unknown fasta files generated in step 26 ([condition name]_NC_ChrUn.fasta and [condition name]_ChrKnown.fasta)
- Aside from input and output file, this command is identical to step 20

Move codon.txt

Purpose: A future step requires a text file holding codons to convert nucleotide sequences to protein sequences. This step acts to move the necessary text file to the current working area to avoid future errors.

Command:

```
cp software/lncRNA-Identification-Pipeline-main/codon.txt .
```

Example:

```
cp software/lncRNA-Identification-Pipeline-main/codon.txt .
```

Output:

- A ".txt" file containing genetic codons

Notes:

- It may be easy to miss so note the period at the end of the command

Convert to protein sequence

Purpose: The next filtering step requires protein sequences, as opposed to nucleotide sequences, this step functions to convert the nucleotide sequences generated in step 27 to protein sequences.

Command:

```
software/lncRNA-Identification-Pipeline-main/sixFrameTranslate.pl -i [
   ↪ condition name]_NC.fasta -o [condition name].faa -l 6
```

Example:

```
software/lncRNA-Identification-Pipeline-main/sixFrameTranslate.pl -i
   ↪ All_NC.fasta -o All.faa -l 6
```

Output:

- A ".faa" file containing protein sequences for each nucleotide sequences provided in the fasta file generated in step 27

Notes:

- You can remove the fasta file generated in step 27 at this point if space is a concern

Remove excessively long transcripts

Purpose: The protein sequence parsing database function is unable to handle protein sequences that exceed 100,000 amino acids in length. A transcript of this size is very likely to contain a protein domain of interest so it would be filtered out in subsequent steps regardless, this step retains only protein sequences shorter than 100,000 amino acids.

Command:

```
awk 'BEGIN {RS = ">" ; ORS = ""} length($2) <= 100000 {print ">"$0}' [
   ↪ condition name].faa > [condition name]_Pfam_Ready.faa
```

Example:

```
awk 'BEGIN {RS = ">" ; ORS = ""} length($2) <= 100000 {print ">"$0}'
   ↪ All.faa > All_Pfam_Ready.faa
```

Output:

- A ".faa" file containing protein sequences for each nucleotide sequences provided in the fasta file generated in step 27, excluding those over 100,000 amino acids in length

Notes:

- N/A

Split protein files

Purpose: The protein sequence file can be quite large and time consuming to analyze as a single file, to speed up the process, as well as avoid having to restart the analysis if the

analysis is interrupted for whatever reason, this step will split the protein sequence file into many smaller pieces.

Command:

```
split --additional-suffix=.faa [condition name]_Pfam_Ready.faa [
   ↪ condition name]_part
```

Example:

```
split --additional-suffix=.faa All_Pfam_Ready.faa All_part
```

Output:

- Many ".faa" files holding a portion of the sequences from the ".faa" file generated in step 30

Notes:

- Don't be intimidated by the number of files! They will be analyzed and cleaned up in the next 3 steps.

Parse Pfam database

Purpose: This step will compare the protein sequences generated in step 29 to the PFam database to identify those with protein family domains within them. If a transcript has protein domain(s) within it, it can be removed in a subsequent step as the transcript is likely a functional protein.

Command:

```
for files in [condition name]_part*; do software/hmmer*/src/hmmsearch
   ↪ --tblout $files.txt [Pfam database] $files; rm $files; done
```

Example:

```
for files in All_part*; do software/hmmer*/src/hmmsearch --tblout
   ↪ $files.txt [Pfam database] $files; rm $files; done
```

Output:

- Many ".txt" files containing information about matches, or lack thereof, to known protein domains

Notes:

- This command looks different than other commands as it is a loop running over all the partial files generated in step 31
- This command removes the partial ".faa" files generated in step 31 automatically, there will still be many files present as there is one output for every partial file that went in to the command
- Do not panic over the large number of ".txt" files, they will be combined and tidied within the next steps
- This step will take a while to run, that is normal

Merge all outputs

Purpose: This step will combine the many outputs from the Pfam search performed in step 32 into a single table for further filtering and analysis.

Command:

```
cat [condition name]_part*.txt > [condition name]_Pfam.txt
```

Example:

```
cat All_part*.txt > All_Pfam.txt
```

Output:

- A ".txt" file containing all results from the Pfam database search

Notes:

- Don't worry about removing the partial results files, this will be done in a single command in the next step

Clean up

Purpose: This step will remove the partial results files from step 32.

Command:

```
rm [condition name]_part*
```

Example:

```
rm All_part*
```

Output:

- There is no output from this step

Notes:

- This should make the working space look much less overcrowded

Filter Pfam list

Purpose: This step will filter the results from the Pfam search to limit results to those that do not contain a protein family. The results at this point will only include transcripts that are intergenic, noncoding, lacking a protein family, and over 200 bases in length.

Command:

```
awk '(NR==1) || ($5 > .1)' [condition name]_Pfam.txt > [condition name
   ↪ ]_Pfam_filtered.txt
```

Example:

```
awk '(NR==1) || ($5 > .1)' All_Pfam.txt > All_Pfam_filtered.txt
```

Output:

- A ".txt" file containing only transcripts without a protein family. As mentioned in the purpose portion of this step, these transcripts are intergenic, noncoding, lacking a protein family, and over 200 bases in length.

Notes:

- If you open this text file, it's not the prettiest. It will be tidied in the next step

List tidying

Purpose: The output of the Pfam database search is not the prettiest and is not ready for the final filtering step, this step cleans the text file up and prepares it for the final filtering step.

Command:

```
awk '{print $1 }' [condition name]_Pfam_filtered.txt > tmp1
grep -v "# *" tmp1 > tmp2
sed -i 's/|/\t/g' tmp2
awk '{print $1 }' tmp2 > tmp3
sort -u tmp3 > [condition name]_Pfam.txt
rm tmp*
```

Example:

```
awk '{print $1 }' All_Pfam_filtered.txt > tmp1
grep -v "# *" tmp1 > tmp2
sed -i 's/|/\t/g' tmp2
awk '{print $1 }' tmp2 > tmp3
sort -u tmp3 > All_Pfam.txt
rm tmp*
```

Output:

- A ".txt" file containing transcripts that are intergenic, noncoding, lacking a protein family, and over 200 bases in length in an easy to read format

Notes:

- This step does a lot of small tasks in one go, be sure to not miss small details like quotation marks and open and closing brackets
- These commands are best typed to avoid formatting issues

Convert to bed file

Purpose: Similar to previous steps, the text file generated in step 36 is not in the appropriate format for further steps so this step will convert the file to be in the correct format.

Command:

```
sed -i 's/:/\t/g' [condition name]_Pfam.txt
sed 's/-/\t/g' [condition name]_Pfam.txt > [condition name]_Pfam.bed
```

Example:

```
sed -i 's/:/\t/g' All_Pfam.txt
sed 's/-/\t/g' All_Pfam.txt > All_Pfam.bed
```

Output:

- A ".bed" file containing genomic coordinates for transcripts that are intergenic, noncoding, lacking a protein family, and over 200 bases in length

Notes:

- This command is similar to other steps where our files have been converted to bed files

Split bed file

Purpose: In order to avoid errors and loss of information, files should be split to represent known chromosomes and unknown chromosomes.

Command:

```
grep 'chrUn*' [condition name]_Pfam.bed > [condition name]_Pfam_ChrUn.
   ↪ bed
grep -v 'chrUn*' [condition name]_Pfam.bed | tail -n+2 > [condition
   ↪ name]_Pfam_ChrKnown.bed
```

Example:

```
grep 'chrUn*' All_Pfam.bed > All_Pfam_ChrUn.bed
grep -v 'chrUn*' All_Pfam.bed | tail -n+2 > All_Pfam_ChrKnown.bed
```

Output:

- A ".bed" file containing coordinates for intergenic, noncoding, protein family lacking, transcripts over 200 bases in length located on known chromosomes
- A ".bed" files containing coordinates for intergenic, noncoding, protein family lacking, transcripts over 200 bases in length located on unknown chromosomes

Notes:

- Aside from input and output files, this step is identical to step 17 and 25

Obtain fasta file

Purpose: This step functions to use the BED files previously generated to isolate nucleotide sequences of interest for the final filtering step.

Command:

```
bedtools getfasta -fi [unknown chromosome].fa -bed [condition name]
   ↪ _Pfam_ChrUn.bed -fo [condition name]_Pfam_ChrUn.fasta
bedtools getfasta -fi [reference fasta].fa -bed [condition name]
   ↪ _Pfam_ChrKnown.bed -fo [condition name]_Pfam_ChrKnown.fasta
```

Example:

```
bedtools getfasta -fi chrUn.fa -bed All_Pfam_ChrUn.bed -fo
   ↪ All_Pfam_ChrUn.fasta
bedtools getfasta -fi canFam6.fa -bed All_Pfam_ChrKnown.bed -fo
   ↪ All_Pfam_ChrKnown.fasta
```

Output:

- A ".fasta" file containing nucleotide sequences for protein family lacking, noncoding, intergenic transcripts over 200 bases in length located on known chromosomes
- A ".fasta" files containing nucleotide sequences for protein family lacking, noncoding, intergenic transcripts over 200 bases in length located on unknown chromosomes

Notes:

- Aside from the input and output file, this command is identical to step 18 and 26
- If any .fai files were generated, they can be removed at this point
- You can remove the two partial bed files from step ([condition name]_Pfam_ChrUn.bed and [condition name]_Pfam_ChrKnown.bed)

Reconstitute fasta file

Purpose: This step functions to reconstitute fasta files generated in step 39 to represent all query sequences in a single file. This avoids confusion.

Command:

```
cat [condition name]_Pfam_ChrUn.fasta [condition name]_Pfam_ChrKnown.
   ↪ fasta > [condition name]_Pfam.fasta
```

Example:

```
cat All_Pfam_ChrUn.fasta All_Pfam_ChrKnown.fasta > All_Pfam.fasta
```

Output:

- A fasta file with nucleotide sequences for all protein family lacking, noncoding, intergenic transcripts over 200 base pairs in length on both known and unknown chromosomes

Notes:

- You can now remove the known and unknown fasta files generated in step 41 ([condition name]_Pfam_ChrUn.fasta and [condition name]_Pfam_ChrKnown.fasta)
- Aside from input and output file, this command is identical to step 19 and 27

Prepare BLAST database

Purpose: In order to perform the final filtering step, the known gene database needs to be prepared for parsing. This step prepares that database.

Command:

```
software/ncbi*/bin/makeblastdb -in [cds genome].fa -out BLAST_DB -
   ↪ parse_seqids -dbtype nucl
```

Example:

```
software/ncbi-blast-2.14.0+/bin/makeblastdb -in Canis_lupus_familiaris
   ↪ .ROS_Cfam_1.0.cds.all.fa -out BLAST_DB -parse_seqids -dbtype
   ↪ nucl
```

Output:

- 10 BLAST_DB files that are ready for parsing in the next step

Notes:

- Be sure you are using the cds file, if you use another file, your analysis will run but will be incorrect.
- Be sure your cds file is unzipped, so there is no file type other than ".fa" or ".fasta"

Run BLAST

Purpose: The final filtering step is to compare the remaining transcripts to known genes to remove those statistically significantly similar to said known genes. Most known genes have already been removed by prior filtering steps, this step will remove anything that previously slipped through the cracks.

Command:

```
software/ncbi*/bin/blastn -db BLAST_DB -query [condition name]_Pfam.
   ↪ fasta -outfmt 6 -max_target_seqs 1 -max_hsps 1 -out [condition
   ↪ name]_BLAST.txt
```

Example:

```
software/ncbi-blast-2.14.0+/bin/blastn -db BLAST_DB -query All_Pfam.
   ↪ fasta -outfmt 6 -max_target_seqs 1 -max_hsps 1 -out All_BLAST.
   ↪ txt
```

Output:

- A ".txt" file containing BLAST results for all query sequences generated from previous filtering steps

Notes:

- This command will likely generate a warning recommending more matches be analyzed, since the function of this step is only to remove those with a significant match, this warning can be disregarded
- You can remove the BLAST_DB files made in step 43 once this step has been completed

Filter BLAST results

Purpose: This step will limit the results from the BLAST search to only those deemed statistically significant.

Command:

```
awk '(NR==1) || ($11 < .1)' [condition name]_BLAST.txt > [condition
   ↪ name]_BLAST_Matches.txt
```

Example:

```
awk '(NR==1) || ($11 < .1)' All_BLAST.txt > All_BLAST_Matches.txt
```

Output:

- A ".txt" file containing transcripts that match to known genes at a statistically significant level

Notes:

- Unlike previous steps, this list is made up of transcripts that are known genes. This will be used in a future step to remove transcripts
- This command should look similar, it is reminiscent of step 35

Generate match list

Purpose: The BLAST output contains lots of useful information, however, it is not all necessary for the next step. This step removes extraneous information.

Command:

```
awk '{print $1}' [condition name]_BLAST_Matches.txt > [condition name]
   ↪ _BLAST_Matches.list
```

Example:

```
awk '{print $1}' All_BLAST_Matches.txt > All_BLAST_Matches.list
```

Output:

- A ".list" file containing the location of transcripts matching known genes in the "chr:start-end" format

Notes:

- This command and step is similar to step 22

Convert list file format

Purpose: Ahead of the final step, the BLAST hits file needs to be converted into a BED file format. This step does that.

Command:

```
sed -i 's/:/\t/g' [condition name]_BLAST_Matches.list
sed 's/-/\t/g' [condition name]_BLAST_Matches.list > [condition name]
    ↪ _BLAST_Matches.bed
```

Example:

```
sed -i 's/:/\t/g' All_BLAST_Matches.list
sed 's/-/\t/g' All_BLAST_Matches.list > All_BLAST_Matches.bed
```

Output:

- A ".bed" file containing genomic coordinates for transcripts that returned a statistically significant BLAST hit

Notes:

- This is the same command as step 37 with different input/output files

Generate final lncRNA list

Purpose: You've reached the end! This final step will compare the list of transcripts that went into the BLAST to the list of those deemed to be statistically significant matches and will report back only those that did not generate a BLAST hit. The transcripts returned by this command will be intergenic, noncoding, lacking a protein domain, over 200 base pairs in length, and do not match a known gene within the organism. This is good evidence that these transcripts, although transcribed, are not translated into proteins, and therefore are good candidates for lncRNA.

Command:

```
diff [condition name]_BLAST_Matches.bed [condition name]_Pfam.bed |
   ↪ sed -n -e 's/^> //p' > [condition name]_lncRNA.bed
```

Example:

```
diff All_BLAST_Matches.bed All_Pfam.bed | sed -n -e 's/^> //p' >
   ↪ All_lncRNA.bed
```

Output:

- A final ".bed" file containing coordinates of filtered transcripts that are likely to be lncRNA!!

Notes:

- Be sure to note the inputs of this command are the BED file from step 39 and the BED file from step 47

5.4 Further Analysis and Notes

5.4.1 Further Analysis

Expression

One avenue of investigation that is often pursued with lncRNA is expression analysis.
A tool that can be used for this analysis is Salmon. Salmon is ”a lightweight method for quantifying transcript abundance from RNA–seq reads” that can easily be used to assess lncRNA expression levels. This can be done by creating a set of index files based on a provided reference genome. These index files can then be used to quantify expression of transcripts within the index. Salmon provides 2 excellent tutorials for installation and usage, they can be located at the following locations: link and link.
For lncRNA quantification, the lncRNA list generated by this tutorial can be used with the bedtools function to generate a lncRNA fasta file. This is similar to steps 19, 28, and 41 in this tutorial. This fasta file can be provided to Salmon to generate the index files. The original sample files can then be assessed with this tool and index files to quantify lncRNA expression in each sample.
Salmon also possess a tool called ”quantmerge” that can combine the quantification files into one file. There is limited online documentation for this tool so using the command: ”salmon quantmerge –help” will provide the documentation for this tool’s usage.
Here is an example using the data from this tutorial, beginning in the directory where the analysis was done:

```
bedtools getfasta -fi canFam6.fa -fo lncRNA.fasta -bed All_lncRNA.bed
mkdir Salmon
cd Salmon
```

```
wget https://github.com/COMBINE-lab/salmon/releases/download/v1.5.1/
   ↪ salmon-1.5.1_linux_x86_64.tar.gz
tar -xvf salmon*
salmon*/bin/salmon index -t ../lncRNA.fasta -i Index
salmon*/bin/salmon quant -i Index -l A -1 ../SRR18899314_1.fastq.gz -2
   ↪  ../SRR18899314_2.fastq.gz -p 8 --validateMappings -o quants/
   ↪ Sample1_quant
salmon*/bin/salmon quant -i Index -l A -r ../SRR24066292.fastq.gz -p 8
   ↪  --validateMappings -o quants/Sample2_quant
salmon*/bin/salmon quant -i Index -l A -r ../SRR24066293.fastq.gz -p 8
   ↪  --validateMappings -o quants/Sample3_quant
salmon*/bin/salmon quantmerge --quants quants/Sample1_quant/ quants/
   ↪ Sample2_quant/ quants/Sample3_quant/ -o lncRNA_Expression.txt
```

The resulting files can be used as is or opened in programs such as Excel for reporting. The ”quant.sf” files for each sample contain the lncRNA name, length, effective length, expression as measured in transcripts per million (TPM), and number of reads. The output file from the quantmerge command contains the lncRNA name and expression in each merged sample. This tool can also be used with a standard reference genome and used to obtain expression data for all transcripts as well [17].

Conservation

Another investigation technique many researchers take on with lncRNA is their conservation score. This can be done with the UCSC tool bigWigAverageOverBed and a bigwig file containing conservation scores for multiple species. The caveat with the tool is

the alignment files tend to be based on human genome locations and therefore identified lncRNA must be converted to human coordinates first.

Luckily, there is an online tool that can easily do this conversion for you. Located here: LiftOver, you can set your original species, upload the lncRNA bed files generated by this tutorial, and press submit. The tool will convert your genomic locations to human equivalents, and you can click ”View Conversions” to download a new bed file with human genomic coordinates. This file can then be renamed and moved to your analysis location for the next steps [15].

After converting your lncRNA to human coordinates, your bed file will have 5 columns: human chromosome, human start position, human end position, the original transcript ID, and a number indicating if the transcript is repeated and if so, what occurrence of the transcript each new coordinate represents. There is one more modification that needs to be done to prepare your bed file, the fourth and fifth column need to be combined to act as a label for each lncRNA investigated. This can easily be done with a single command:

```
awk '{print $1"\t"$2"\t"$3"\t"$4"."$5}' [Converted lncRNA] > [Modified
    ↪ File]
```

Once your coordinates have been converted and a label has been added, you can use your file and a conservation file in the bigwig (.bw) format to assign a conservation score for each transcript. These scores will be between 0 and 1, with a higher score indicating the transcript is more conserved. Before this analysis can be performed, you’ll need said conservation file. These files can be found here: PhastCon Files. There are two folders that contain the needed files, the phastCons46way folder and the phastCons100way folder. The phastCons46way folder holds data comparing human transcripts to 45 vertebrates; there are 3 options in this folder, all 45 vertebrate comparisons, placental mammal comparisons, and primate comparisons. The phastCons100way folder holds data comparing human

transcripts to 99 vertebrate genomes; there is only one bigwig file within this folder. The file you choose to use is up to you, there are details about the species used in the comparisons within the preamble text in each folder [12]. You can click on the file name you choose and it will download, then it can be moved to the analysis directory or you can use the following command to download the file directly to your analysis folder:

```
wget [link to phastCon file]
```

Now that the lncRNA file has been converted and the conservation file obtained, the Phastcon analysis can be performed. The following documentation explains the options available with the UCSC bigWigAverageOverBed function: link.
Here is an example of obtaining and using the necessary files and tools, having already downloaded the converted lncRNA file, moved it to the analysis directory, and working in the analysis directory:

```
mkdir Phastcon
mv lncRNA_To_Human.bed Phastcon/
cd Phastcon/
awk '{print $1"\t"$2"\t"$3"\t"$4"."$5}' lncRNA_To_Human.bed >
    ↪ lncRNA_To_Human_Labeled.bed
wget https://hgdownload.cse.ucsc.edu/goldenpath/hg19/phastCons46way/
    ↪ placentalMammals.phastCons46way.bw
rsync -aP \rsync://hgdownload.soe.ucsc.edu/genome/admin/exe/linux.
    ↪ x86_64/bigWigAverageOverBed ../software
../software/bigWigAverageOverBed placentalMammals.phastCons46way.bw
    ↪ lncRNA_To_Human_Labeled.bed Phastcon.tab
```

The .tab file has 2 columns you should pay attention to: the first column that is the identifier (the previous steps in this tutorial means these identifiers are the lncRNA identified originally) and the last column is the average score over the covered bases. The last column is the phastCon score for the regions investigated. You now have these scores for reporting and further analysis. This analysis can be performed again on any group of data, you may want to perform it on all transcripts and intergenic transcripts for comparison purposes [11, 2].

Annotation

Many research endeavors intend to assign a role to identified lncRNA. Although definitively assigning lncRNA a role is very difficult without choosing a single lncRNA to research thoroughly through modeling and manipulation, it is possible to associate lncRNA with potential roles and open doors for future research endeavors. This can be achieved in many ways, below is a summary of some of these ways. It may be beneficial to your research to find a contemporaneous study and examine how they annotated lncRNA. A useful tool for annotation comes from the UCSC Genome browser. You may choose to use the genome browser and examine the locations identified as lncRNA. There may be genes nearby that the lncRNA could interact with. The genome browser also allows built in comparative genomics, you can look at a region of interest in your organism and see if there is overlap in other species (predominantly human and mice genomes). A more specific way to annotate lncRNA is based on sequence homology. If a transcript has a similar sequence to other organisms, perhaps they act in similar ways. Phastcon scores can easily allow you to assess if a sequence is highly conserved. These more highly conserved regions can be retained for analysis like any other transcript, just with the knowledge that it is a more conserved region. The UCSC genome browser is full of information and tools,

it is worth playing around with and reading up on [24].

Another way lncRNA can be associated with lncRNA is by correlation. If the expression of lncRNA is correlated with a gene, perhaps they're working towards the same goal. In order to generate a correlation matrix, you would need a single file containing expression levels for all lncRNA and all transcripts in samples. This file could then use a tool such as Excel or the R function "corr" to generate a correlation matrix. Depending on how you go about generating this matrix, different data preparation may be required. A few online searches would likely help you prepare your data for this analysis.

There are also several online tools that can be used with lists of genes to associate them with functions. If you are able to identify a list of associated genes (perhaps by proximity or correlation with a lncRNA), tools like the gene ontology resource, DAVID bioinformatics resources, and the PANTHER classification system will allow you to enter said list and uncover associations. The gene ontology resource will report what biological processes are associated with the specific gene collection [7]. DAVID is able to report associated diseases, functional annotations, gene ontology, interactions, pathways, protein domains, tissue specific expression, and associated literature–this is assuming though, that these associations exist [5]. PANTHER is able to report functional classification and statistical enrichment for gene lists [22]. These tools all have their uses and will be variably useful to you depending on your data set and identified genes. Further research and investigation into each tool will allow you to make an educated decisions about what tools to use.

Other

There are many more tools that can be used to analyze lncRNA in effective manners, such as enrichment analysis and interaction network predictions. As previously recommended,

reading of literature similar to your project can be a great tool for deciding how you want to analyze your data.

5.4.2 Notes

The final section of this tutorial will provide a few notes of common problems you may encounter and how to navigate them.

Data Management

As mentioned very early in this tutorial, there are many files downloaded and generated that are quite large and can pose a challenge if you have a limited amount of space. For this reason, there are notes indicating what files can be removed when. The removal of these files is strategic, they do not intend to delete files that may be used in reporting or analysis of findings. For instance, the original files are not recommended to be removed as they can be used with tools like Salmon later in analysis; files such as the first Hisat alignment files are recommended to be removed as more accurate alignment files will be generated and the information in these files is very unlikely to be used later in the analysis. If you are truly lacking space, removing of fasta files can help conserve space without lose of information (as the fasta file can be regenerated from a file with genomic coordinates). Another data management pitfall you may find comes to the naming convention used. The commands provided by this code are intended to be descriptive and clear but you are free to name files as you see fit. Although in the moment, you may feel as if you'll remember what each file is, that may not be the case when you come back to your data after taking a break. For this reason, it is recommended to keep the naming conventions as descriptive as possible without being excessive in length.

Error Troubleshooting

As you progress through the tutorial, you may encounter an error or two. It is important to know how to navigate errors as they occur.

The first step in troubleshooting an error is to read the error itself. It may seem silly but errors tend to tell you exactly what needs to be modified. It is most likely you would receive an error that states ”No such file or directory,” this error means one (or more) arguments cannot be found. The simplest solution is to ensure you haven’t made a typo in your code, a missing space or character can cause big problems. Other errors may tell you that a file is not in the right format or data needs to be sorted in a specific manner. As stated, many errors will tell you exactly what you need to fix.

If you are unable to get a command to work after checking the syntax and reading/trying to address the error, online is full of resources and kind people. Sources like BioStars, GitHub, Galaxy, and Stack Overflow are all full of people troubleshooting errors and helping each other. A few online searches should allow you to fix many errors. Finally, if you seem to have tried every solution to getting a software to run and it just will not cooperate, there is always the option to delete and re-download a tool. Sometimes things don’t go according to plan and the best course of action is to start over.

Words of Encouragement

This tutorial is very long and involved but you can do it! It is written to in small, simple steps to make it as beginner friendly as possible so don’t let the length scare you. It can be disheartening when a command doesn’t run and it seems like a solution is unattainable; take a break, get a drink and a snack and look at it with fresh eyes. You are unlikely to be the first to have this problem so someone, somewhere has solved it before and you can too!

5.5 References

[1] *amarceau-998/lncRNA-Identification-Pipeline: Collection of scripts used to identify lncRNA*. URL: https://github.com/amarceau-998/lncRNA-Identification-Pipeline.

[2] *bigWigAverageOverBed manual with usage examples — BioQueue Encyclopedia*. URL: https://open.bioqueue.org/home/knowledge/showKnowledge/sig/ucsc-bigwigaverageoverbed.

[3] *CPC2 @ CBI, PKU*. URL: http://cpc2.gao-lab.org/download.php.

[4] *Cufflinks*. URL: http://cole-trapnell-lab.github.io/cufflinks/cuffcompare/.

[5] *DAVID Functional Annotation Bioinformatics Microarray Analysis*. URL: https://david.ncifcrf.gov/home.jsp.

[6] *Ensembl genome browser 110*. URL: https://useast.ensembl.org/index.html.

[7] *Gene Ontology Resource*. URL: http://geneontology.org/.

[8] *GitHub - lh3/seqtk: Toolkit for processing sequences in FASTA/Q formats*. URL: https://github.com/lh3/seqtk.

[9] *GitHub - ncbi/sra-tools: SRA Tools*. URL: https://github.com/ncbi/sra-tools.

[10] *HMMER*. URL: http://hmmer.org/.

[11] *Index of /admin/exe/linux.x86_64*. URL: https://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64/.

[12] *Index of /goldenpath/hg19*. URL: http://hgdownload.cse.ucsc.edu/goldenpath/hg19/.

[13] Daehwan Kim, Ben Langmead, and Steven L. Salzberg. "HISAT: a fast spliced aligner with low memory requirements". In: *Nature Methods 2015 12:4* 12.4 (Mar. 2015), pp. 357–360. ISSN: 1548-7105. DOI: 10.1038/nmeth.3317. URL: https://www.nature.com/articles/nmeth.3317.

[14] Johnny T.Y. Kung, David Colognori, and Jeannie T. Lee. *Long noncoding RNAs: Past, present, and future*. 2013. DOI: 10.1534/genetics.112.146704. URL: /pmc/articles/PMC3583990/?report=abstract%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3583990/.

[15] *Lift Genome Annotations*. URL: https://genome.ucsc.edu/cgi-bin/hgLiftOver.

[16] *Nucleotide BLAST: Search nucleotide databases using a nucleotide query*. URL: https://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastn&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome.

[17] Rob Patro et al. "Salmon provides fast and bias-aware quantification of transcript expression". In: *Nature Methods 2017 14:4* 14.4 (Mar. 2017), pp. 417–419. ISSN: 1548-7105. DOI: 10.1038/nmeth.4197. URL: https://www.nature.com/articles/nmeth.4197.

[18] *Pfam: Search Pfam*. URL: http://pfam.xfam.org/search#tabview=tab1.

[19] Aaron R Quinlan and Ira M Hall. "BEDTools: a flexible suite of utilities for comparing genomic features". In: *Bioinformatics* 26.6 (Mar. 2010), pp. 841–842. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btq033. URL: https://doi.org/10.1093/bioinformatics/btq033.

[20] *samtools Tutorial*. URL:
http://quinlanlab.org/tutorials/samtools/samtools.html.

[21] *StringTie*. URL:
http://ccb.jhu.edu/software/stringtie/index.shtml?t=manual.

[22] Paul D. Thomas et al. "PANTHER: Making genome-scale phylogenetics accessible to all". In: *Protein Science* 31.1 (Jan. 2022), pp. 8–22. ISSN: 1469896X. DOI: 10.1002/PRO.4218.

[23] *UCSC Genome Browser Downloads*. URL:
https://hgdownload.soe.ucsc.edu/downloads.html.

[24] *UCSC Genome Browser Gateway*. URL:
https://genome.ucsc.edu/cgi-bin/hgGateway.