Table of Contents

Please see http://z.umn.edu/ris-rnaseq/ for the latest version of this tutorial document!

RNASeq Analysis With the Command Line

Last Updated: 2023-04-10

Last Delivered: 2022-11-08

Previous recording on YouTube: https://www.youtube.com/watch?v=x3H8pn_wSjA

Expand all details (Useful for printing!)

Collapse all details

Part 0: Introduction

This tutorial will cover the basic use of the MSI workflow for bulk RNAseq data analysis. The tools are used via the Linux command line, and will cover important aspects of data QC and considerations to take when interpreting RNAseq results.

0.1 Formatting in This Document

Throughout this tutorial, there will be formatting cues to highlight various pieces of information.

This is just background information. There are no tutorial-related tasks in these boxes. Links to supporting material and further explanations of points we raise in the tutorial will appear like this.

This is a warning. Common pitfalls, cautionary information, and important points to consider will appear like this.

```
This is code, or a literal value that you must enter or select to run a part of the tutorial
```

▸ ***These boxes contain detailed information. Click on them to expand them***

Long command lines will be wrapped with a backslash (`\`); you do not need to enter these when typing in the command:

```
short command

this_is_a_really_long_command_line \
    with_many_options \
    and_with_many_long_arguments \
    such_that_wrapping_makes \
    it_easier_to_read
```

Return to top

0.2: Goals

By the end of this tutorial you should be able to:

- Use the UMII-MSI workflow to generate relative gene expression data from RNAseq data
- Assess the quality of your libraries with post-lab metrics
- Identify differentially-expressed genes among experimental groups using the automated DEG testing routines

Return to top

0.3: Scope of the Tutorial

This tutorial will only cover differential gene expression analysis of **bulk RNASeq** of mRNA and pathway enrichment analyses of differentially expressed genes (DEGs). We will not cover single cell RNASeq analysis or small RNA sequencing analysis. We will also not cover coexpression analysis or transcriptome assembly in detail. There are links to guides at the bottom of the tutorial document for coexpression analysis and transcriptome assembly.

While we will be teaching how to use analysis tools from the command-line, the names and options that we are supplying should be available in the Galaxy versions. This tutorial will not cover workflow development nor tool use in Galaxy.

The sections toward the end of the document has supplementary sections with brief discussions about RNAseq topics beyond “standard” differential expression analysis in bulk samples. If you are looking to perform more specialized analysis with RNAseq data, please see those sections; perhaps there are useful references or concepts for you there!

Return to top

0.4: Prerequisites

This tutorial requires that you be familiar with accessing MSI via command line tools. You can access MSI via terminal emulator or SSH client by following the instructions on this guide: https://www.msi.umn.edu/content/connecting-hpc-resources

You can also use MSI's Jupyter Notebook service for command line access to MSI resources. Instructions for using MSI's Jupyter Notebook service are located at this link: https://www.msi.umn.edu/support/faq/how-do-i-get-started-jupyter-notebooks

You must also have familiarity with using the command line to interact with Linux systems. The tutorial will use standard Linux utilities to navigate the file system, and our RNAseq tools are written to work from the command line. You can view MSI's tutorial for the Linux command line at this link (requires UMN Internet ID): https://pages.github.umn.edu/dunn0404/intro-to-linux/

If you do not have an active UMN Internet ID, you can also get a good introduction to the Linux command line from the "Linux Essentials" guide from UC Riverside: http://hpcc.ucr.edu/manuals_linux-basics_cmdline-basics.html

The final sections of this tutorial will require you to have a way to transfer files between your local workstation and MSI servers. I recommend tutorial attendees use Filezilla because it is cross-platform, and MSI maintains a guide for its setup: https://www.msi.umn.edu/support/faq/how-do-i-use-filezilla-transfer-data

Return to top

Part 1: Command Line Refresher

Accessing MSI systems requires that you either be connecting through the UMN network (`eduroam` or Ethernet in a UMN building) or connected to the UMN VPN. Please see https://it.umn.edu/services-technologies/virtual-private-network-vpn for information on connecting to the UMN VPN.

You will also be required to have DUO multi-factor authentication set up. Please see https://it.umn.edu/services-technologies/self-help-guides/duo-set-use-duo-security for information about enrolling your device in DUO.

This section will take approximately 50 minutes to complete. It serves as a refresher for the command line and will give you the essential skills needed to run the later parts of the tutorial. Please see MSI's "Introduction to Linux" tutorial for a more complete tutorial on using the command line on MSI systems. You can find the materials for the "Introduction to Linux" tutorial at this link: https://pages.github.umn.edu/dunn0404/intro-to-linux/

Return to top

1.1: What is a Shell?

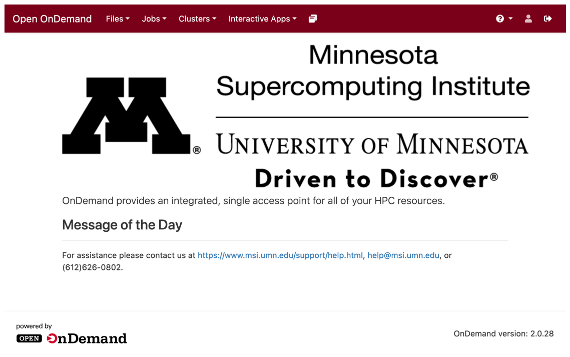
A general and functional definition of a "shell" is a user interface that allows someone to interact with an operating system. Shells can either be *graphical* or *textual* in nature. The Microsoft Windows interface or the GNOME desktops that are running on the workstations in the MSI computer lab are examples of different graphical shell interfaces. The terminal emulator program or the MS-DOS command prompt are examples of textual shell interfaces. The main way we will be interacting through MSI systems in this tutorial is through a textual shell interface (the "command line").

Learning how to use the command line interface for MSI takes some practice, but you will find that it can be much more powerful and fast to manipulate files than through the graphical shell.

Return to top

1.2: Accessing a Command Line on MSI

In this tutorial, we will use the MSI Open OnDemand (OOD) service to access a command line interface. Open a web browser and navigate to https://ood.msi.umn.edu/. Log in with your UMN user name and password. You will see a screen like the one shown below:

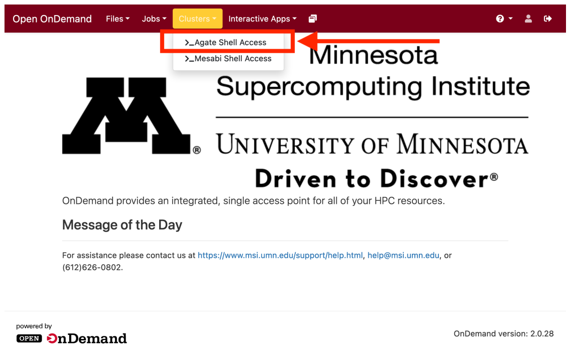


OnDemand provides an integrated, single access point for all of your HPC resources.

Message of the Day

For assistance please contact us at https://www.msi.umn.edu/support/help.html, help@msi.umn.edu, or (612)626-0802.

Click on the `Clusters` button. A drop-down menu will fold out. Click on `Agate Shell Access`:

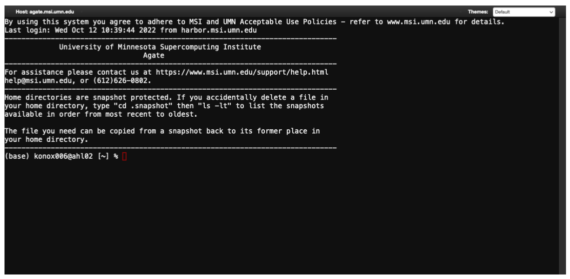


OnDemand provides an integrated, single access point for all of your HPC resources.

Message of the Day

For assistance please contact us at https://www.msi.umn.edu/support/help.html, help@msi.umn.edu, or (612)626-0802.

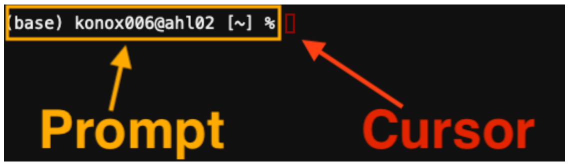
A new tab will open with a command line prompt. You can change the appearance with the `Theme:` drop-down menu.



```
By using this system you agree to adhere to MSI and UMN Acceptable Use Policies - refer to www.msi.umn.edu for details.
Last login: Wed Oct 12 10:39:44 2022 from harbor.msi.umn.edu
-------------------------------------------------------------------------------
             University of Minnesota Supercomputing Institute
                                 Agate
-------------------------------------------------------------------------------
For assistance please contact us at https://www.msi.umn.edu/support/help.html
help@msi.umn.edu, or (612)626-0802.
-------------------------------------------------------------------------------
Home directories are snapshot protected. If you accidentally delete a file in
your home directory, type "cd .snapshot" then "ls -lt" to list the snapshots
available in order from most recent to oldest.

The file you need can be copied from a snapshot back to its former place in
your home directory.
-------------------------------------------------------------------------------
(base) konox006@ahl02 [~] %
```

The portion of the prompt to the left of and including the percent symbol (`%`) is called the "prompt." The block is called the "cursor." When you type commands into the command line, the symbols will appear to the left of the cursor:



```
(base) konox006@ahl02 [~] %
```

Note that my prompt may look different from yours! The function is identical, regardless of appearance.

Note that you cannot use the mouse to move the cursor around in the terminal area! You must use the arrow keys to move the cursor to edit the command you are currently typing. We will cover this later when we start entering actual commands!

Before we start to write and enter commands at the command prompt, though, we will go over directory and file structure on MSI systems.

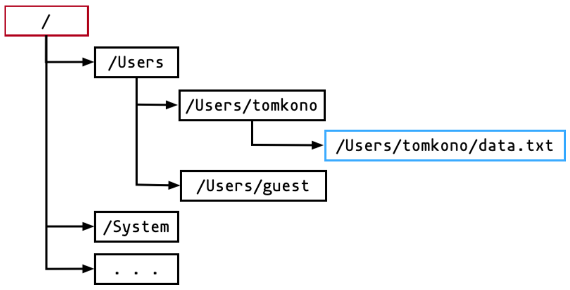
▸ ***Other ways to access a command line on MSI systems***

Return to top

1.3: Directories and Files

Understanding files and directories is one of the key pieces of using a UNIX-like system (which includes MSI systems). One of the defining characteristics of UNIX-like systems is that "everything is a file." This means that everything that the computer can interact with, including input and output devices, is represented as a file on the filesystem. Keyboards, mice, monitors, hard disks, executable programs, data - these are all represented as files. You generally will not have to interact with the files that represent the computer hardware, but you will have to navigate the filesystem to find the programs that you want to use to analyze your data and point these programs at your data set.

The image that is often used to illustrate the concept of the directory and file structure is a tree. In this metaphor, a directory is a branch and a file is a leaf. Branches can contain multiple other branches and leaves. On Windows and Mac OS (and other common desktop operating systems), directories are often called "folders," so you can think of these as interchangeable terms.



In the tree cartoon above, boxes represent directories or files, and indentation represents nesting. The text inside the boxes represents the names. The red box (around `/`) represents the *root* of the filesystem, and the blue box (around `/Users/tomkono/data.txt`) represents a *file*. The boxes in black are *directories*.

Just like in evolutionary biology, a tree structure does not really capture the relatedness of directories and files. Some special types of files are actually links to paths in the filesystem that are very divergent from where the locations of the link files are, like reticulations in a phylogenetic tree. There is optional material later in this tutorial that will show you how to make links. These are very useful for making a dataset easy to find without having to actually copy the raw data.

Notice the slashes (`/`) between parts of the names - these delimit the actual directories that make up the full path to the file. In the case of `/Users/tomkono/data.txt`, you can read the name and know that the file called `data.txt` is located under the root (`/`), then the `Users` directory, then the `tomkono` directory. In UNIX terminology, this is known as the *path* to the file. More specifically, it is known as the *absolute path* because it starts at the root of the filesystem. This is the way I prefer to specify paths to files because there is no ambiguity as to where the file is located.

A path that does not start at the root is known as a *relative path*. To really show the difference between absolute paths and relative paths, we will have to run a few commands, so we will start to do that now.

When typing file paths, be sure to include the root slash at the beginning. A path of `Users/tomkono/data.txt` means something very different from the path `/Users/tomkono/data.txt`.

Return to top

1.4: Basic Commands and Exercises

1.4.1: The Structure of a Command

A UNIX or Linux command consists of a *program*, *options*, and *arguments*. The *program* is what actually does something to a file. The *options* modify the behavior of the program. The *arguments* are the files on which the program operates. The parts of a command are separated by *spaces*, so be sure to include those as you follow along with the examples in the next section. A command is completed with the `Enter` key.

Sometimes, as you will see in the next section, the arguments are not required. In cases where the argument is not required, a default value will be substituted in to the command. You should double-check that the program will work as intended with the default values before you run them.

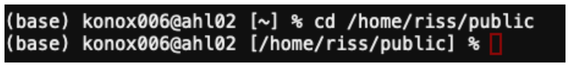
Note that mouse support is very limited in most terminal programs. You can highlight and copy text with the mouse, but you must use the arrow keys to move the cursor around. The `left` and `right` arrow keys will move you backward and forward within the current command, and the `up` and `down` arrow keys will move you earlier and later in the *command history*.

Also note that all parts of a command are *case sensitive*. That means that capitalization matters. `A` is different from `a`.

Return to top

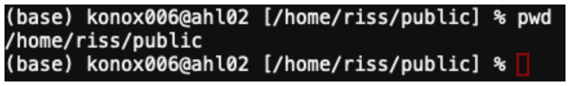
1.4.2: Running Commands

The first command we will run is one that will change directories. This is how we navigate the filesystem. The command is called `cd` for "change directory." Type `cd` then a space, then the path to where you would like to go. We will go to the `/home/riss/public` directory. Press `Enter` when you have finished typing the command:



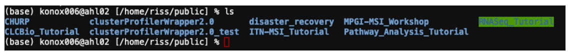
```
(base) konox006@ahl02 [~] % cd /home/riss/public
(base) konox006@ahl02 [/home/riss/public] %
```

Notice how your prompt updates to let you know that you are now in a different directory than when you started. This is a quick way to check where you are currently working. You can verify this with the `pwd` command. Type `pwd` then `Enter` to print the *working directory*:



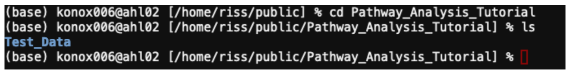
```
(base) konox006@ahl02 [/home/riss/public] % pwd
/home/riss/public
(base) konox006@ahl02 [/home/riss/public] %
```

Next, we will list the contents of the current working directory. We use the `ls` command to do this. Type `ls` then `Enter`:



```
(base) konox006@ahl02 [/home/riss/public] % ls
CHURP            clusterProfilerWrapper2.0       disaster_recovery  MPGI-MSI_Workshop          RNASeq_Tutorial
CLCBio_Tutorial  clusterProfilerWrapper2.0_test  ITN-MSI_Tutorial   Pathway_Analysis_Tutorial
(base) konox006@ahl02 [/home/riss/public] %
```

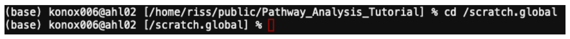
There are multiple items under `/home/riss/public`. The blue coloring indicates that they are *directories*. Let's `cd` into the `Pathway_Analysis_Tutorial` directory now and list contents again:



```
(base) konox006@ahl02 [/home/riss/public] % cd Pathway_Analysis_Tutorial
(base) konox006@ahl02 [/home/riss/public/Pathway_Analysis_Tutorial] % ls
Test_Data
(base) konox006@ahl02 [/home/riss/public/Pathway_Analysis_Tutorial] %
```

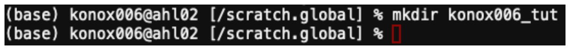
Note that we did not specify a leading slash (`/`) on this command. This is because we want to go into a directory that is *relative* to our current working directory. We will come back to this directory later. It has the test data set that you will use for the next part of the tutorial. For now, we will make a directory to hold the output files from the pathway enrichment analysis.

We will navigate to *global scratch*. This is a special part of the MSI filesystem where you can write temporary data. We will use it to hold the output files from the pathway analysis that will be run later in this tutorial. The global scratch directory is called `/scratch.global` (note the leading slash; this is an *absolute* path!):



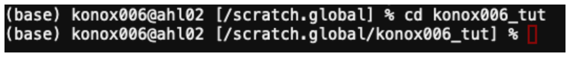
```
(base) konox006@ahl02 [/home/riss/public/Pathway_Analysis_Tutorial] % cd /scratch.global
(base) konox006@ahl02 [/scratch.global] %
```

Next, we will *make a new directory* with the `mkdir` command. Because global scratch is a public area, it can be difficult to keep track of which data are yours. To help organize it, we will make a directory that has your name on it so you can always find your files. Type `mkdir`, then a space, then your UMN internet ID, then `Enter`:



```
(base) konox006@ahl02 [/scratch.global] % mkdir konox006_tut
(base) konox006@ahl02 [/scratch.global] %
```

Again, we do not have to specify the full path to the new directory because we are making it *relative* to the working directory, which is `/scratch.global`. Let's `cd` into the directory that we just made:

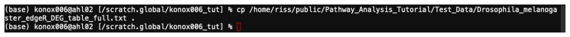


```
(base) konox006@ahl02 [/scratch.global] % cd konox006_tut
(base) konox006@ahl02 [/scratch.global/konox006_tut] %
```

Next, we will *copy* a file into our current directory. We use the `cp` command to do this. `cp` takes two *arguments*, a *source* and a *destination*. For this exercise, the *source* will be a long path name:

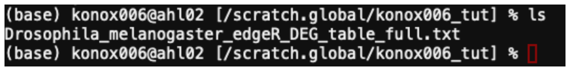
```
/home/riss/public/Pathway_Analysis_Tutorial/Test_Data/Drosophila_melanogaster_edgeR_DEG_table_full.txt
```

and the *destination* will be the current working directory. This is shorthanded by a dot (`.`) in the command:



```
(base) konox006@ahl02 [/scratch.global/konox006_tut] % cp /home/riss/public/Pathway_Analysis_Tutorial/Test_Data/Drosophila_melanoga
ster_edgeR_DEG_table_full.txt .
(base) konox006@ahl02 [/scratch.global/konox006_tut] %
```

Note that the line wrapping in the terminal has no function on the command. It just might make it a bit more difficult to read, unfortunately. Once you run this command, you can list the directory contents to verify that you have successfully copied the file:



```
(base) konox006@ahl02 [/scratch.global/konox006_tut] % ls
Drosophila_melanogaster_edgeR_DEG_table_full.txt
(base) konox006@ahl02 [/scratch.global/konox006_tut] %
```

These are mostly the commands you need to know to follow along with this tutorial. There are additional resources below, but they are not required for you to use our script for pathway analysis.

To recap:

- `cd` : change directory
- `pwd` : print working directory
- `ls` : list directory contents
- `mkdir` : make new directory
- `cp` : copy file

From this point on, commands will only be written in `monospace typeface` and the prompt will not be included. For the ease of copying and pasting commands, we will not include screenshots.

Return to top

1.5: Other Useful Commands

There are many more commands that exist on a UNIX or Linux system. There are several dozen that I know very well because I use them on a daily basis. One of the most useful commands is `man`, which brings up a manual page for a certain command. Manual pages list all of the available options for a given program. If you want to know all of the ways that the behavior of a program can be modified, check the manual!

▸ ***More about reading manual pages, with example***

A Note on `ln`

Linking a file is a very useful way to make it available in other directories without having to copy it. A link looks like a regular file, but is actually a reference or *alias* to another location on disk. We recommend using *symbolic links* rather than *hard links*. Without getting too technical, a symbolic link ("symlink" for short) behaves as a separate file from the data to which it points. A hard link references the exact same data, so if you delete a hard link, the source data is also deleted.

▸ ***Examples of making symbolic links***

There are many versions of UNIX out there, and some of these commands have different default behaviors and different options, depending on which version you are using. For example, Mac OS X is technically a UNIX, and has all the standard commands listed above, but they behave differently from the tools installed on MSI systems, which is GNU/Linux. If you are unsure of what a command will do, check the manual page!

Return to top

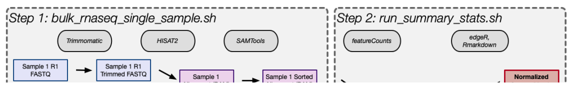
Part 2: MSI- and UMII-Developed Workflow

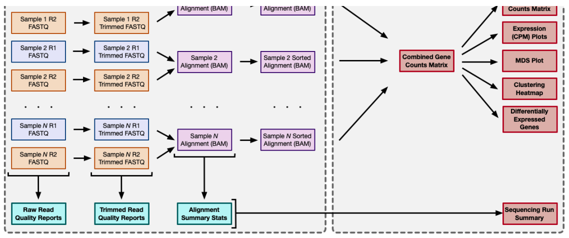
Goal: By the end of this section, you should be know where to find documentation for CHURP, the MSI-UMII workflows for data analysis.

MSI and the University of Minnesota Informatics Institute (UMII) maintain a workflow for bulk RNAseq analysis. The workflow is called "CHURP" (**C**ollection of **H**ierarchical **U**MII-**R**IS **P**ipelines), and it is developed as a software package that is run from the Linux command line. We will cover how to access and use the software in later sections.

Part 2.1: Workflow Steps

The diagram below shows the steps that our workflow handles.





Briefly, the steps of the workflow are as follows:

1. Summarize read quality for each sample.
2. Clean reads for low-quality bases and adapter contaminants for each sample.
3. Map reads to genome for each sample.
4. Count reads within genes.
5. Filter counts for genes with low expression.
6. Test for differential expression.

The results of these steps are written to both the MSI file system and also summarized in a HTML report. We will go over how to browse the output and read the report in later sections.

Part 2.2: Documentation

While we will go over a typical use case of CHURP in this tutorial, it is helpful to know where to find additional information for CHURP. This includes advanced or specialized use cases.

You can find the official documentation for CHURP at the UMN GitHub Wiki link:
https://github.umn.edu/MSI-RIS/CHURP/wiki

For example command lines, a description of the command line arguments, error codes, and overview of the required input data, please see the "Quickstart":
https://github.umn.edu/MSI-RIS/CHURP/wiki/PURR-Manual-Page

Part 2.3: How to Get Help

If you experience trouble with CHURP or have questions about how to apply it to your dataset, please contact the MSI Help Desk (help [at] msi.umn.edu).

Return to top

Part 3: CHURP Input Data

Goal: By the end of this section, you should be able to identify the required inputs for CHURP.

CHURP requires three pieces of data for a bulk RNAseq analysis. An optional fourth input file may be specified to enable differential gene expression testing. The input files are described below.

Part 3.1 FASTQ Files

The sequencing reads from the sequencing facility must be supplied in FASTQ format and placed into a single directory. The files may be gzip-compressed or uncompressed, though we recommend compressing the files to save disk space. In order for CHURP to identify the samples, the files must have standardized names in one of two formats:

- "Standard" Illumina file names without lane identifiers. This is the default delivery from the UMGC:

  ```
  Sample_01_R1_001.fastq.gz
  Sample_01_R2_001.fastq.gz
  ```

  If your samples are split across lanes, then you must combine them before running them through CHURP. Alternately, you can keep them separate but rename them to match the format above, and use them to look for "lane effects" in your expression analyses.
- SRA-generated filenames. This is the default from using the SRA toolkit to write reads to disk using `fastq-dump`:

  ```
  SRR1234567_1.fastq.gz
  SRR1234567_2.fastq.gz
  ```

Note that if you use data from SRA, it is important to make note of the protocol used to generate the data! Technical factors like the mean and variance of the insert size, the read length, and the strand specificity affect how you process the data and interpret the results.

Additionally, most tools require data from the SRA to be exported in a certain way. A common requirement is that the R1 read names end in `/1` and the R2 read names end in `/2`. To ensure that these are written into your SRA data, use the following command:

```
fastq-dump -Q 33 --defline-seq '@$sn[_$rn]/$ri' --defline-qual '+$sn[_$rn]/$ri' --split-files ACCESSION
```

where `ACCESSION` is the SRA accession number for the dataset.

▸ ***Click to view FASTQ format details***

Paired-End and Single-Read Data

CHURP supports both paired-end and single-read sequencing datasets. However, mixing technologies is not recommended because it complicates the expression analysis by requiring that expression quantification be performed differently for different samples. CHURP also supports interleaved paired-end sequencing data, but it requires special options be passed to the program, which is an advanced topic. Please see Supplement X or contact the MSI help desk for assistance with this use case!

Part 3.2 HISAT2 Reference Genome Index

The next piece of required input for CHURP is a reference genome that has been indexed for use with HISAT2. For this tutorial, we will be using a genome index that was prepared specially for the tutorial dataset. For an analysis of a real dataset, you will need to use an index prepared from a full genome assembly. A collection of pre-made indices for MSI users ("bioref," see [Part 2.4] (#2.4)) and instructions for making your own indices will not be covered directly as part of this tutorial, but will be described in [Supplement 1] (#S1).

Part 3.3 GTF Annotations

CHURP also requires a GTF-formatted file with the gene models annotated on the reference genome. This GTF must match the reference genome that is being used for the HISAT2 index; if the chromosome names or coordinates do not match, then expression levels cannot be accurately estimated. The file can be either gzip-compressed or uncompressed, but like with the FASTQ files, we recommend compression to save disk space.

For this tutorial, we will be using a GTF annotation file that was specially-prepared for use with the HISAT2 index. For an analysis of a real dataset, you will need to use a GTF that comes with a reference genome assembly. bioref contains GTF files that are compatible with the collection of HISAT2 genome indices. This is covered in Part 2.4.

Part 3.4: `bioref` Collection of Indices

For researchers working on MSI systems, a collection of reference genome indices and annotations for most model species is available in the following directory:

```
/common/bioref/ensembl
```

These are organized by Ensembl section, and we maintain the following sections:

- `main` : Model organisms and vertebrates
- `grch37` : GRCh37 build of the human genome
- `fungi` : Fungi
- `metazoa` : Invertebrates and other animals
- `plants` : Green plants and algae
- `protists` : Organisms colloquially known as “protists”

Under each of these, you will find sub-directories corresponding to the binomial species name of the organism and the Ensembl release. Under that, there will be sub-directories corresponding to the genome assembly/annotation version. For example, the files for the `GRCh38.p13` version of the human genome from Ensembl release 109 are located here:

```
/common/bioref/ensembl/main/Homo_sapiens-109/GRCh38.p13
```

Under the directory that corresponds to a specific genome version of a given organism, you can find index files for a variety of bioinformatics tools. Again, these are organized into sub-directories:

- `annotation` : Gene model annotations in GFF and GTF formats
- `blast` : NCBI BLAST+ index files for the genome (not peptides, not transcripts)
- `bowtie` : Index files for read mapping with `bowtie` version 1
- `bowtie2` : Index files for read mapping with `bowtie` version 2
- `bwa` : Index files for read mapping with BWA
- `hisat2` : Index files for splice-aware read mapping with HISAT2
- `seq` : FASTA sequence files for the genome

You would use the files in the `hisat2` and the `annotation` directory as inputs for CHURP. For example, to use the

`GRCh38.p13` version of the human genome for a bulk RNAseq analysis, the HISAT2 index would be:

```
/common/bioref/ensembl/main/Homo_sapiens-109/GRCh38.p13/hisat2/genome
```

and the GTF annotation would be

```
/common/bioref/ensembl/main/Homo_sapiens-109/GRCh38.p13/annotation/Homo_sapiens.GRCh38.109.gtf.gz
```

Part 3.4 Optional - Experimental Groups

An optional input for CHURP is a CSV file that describes the experimental metadata for the samples in the dataset. Providing this file to CHURP enables group-based coloring of plots in the final report and differential gene expression testing. CHURP has a routine for automatically generating a template file for the CSV, and we will cover how to use this routine in Part 4.

Return to top

Part 4: Accessing CHURP

Goal: By the end of this section, you should be able to view the CHURP versions that are available, load the correct version of CHURP, and verify that it works.

Note: this section requires that you be connected to MSI and logged in to a Mesabi or Mangi login node. Please review the instructions for connecting to MSI systems at the following link: https://www.msi.umn.edu/content/connecting-hpc-resources

The commands in the following sections can be found in a text file located at the following path on MSI systems:

```
/home/msistaff/public/RNAseq_Tutorial/Tutorial_Commands.txt
```

You can follow along with the tutorial by copying and pasting some of the commands as we encounter them in the tutorial. We will go over what each command does as we get to it, so please do not jump ahead in the file!

First, we will define a variable that holds your user account name. This is to make it more convenient to organize your output. This is command **1** in the text file:

```
ME=$(id -un)
```

Then, we will make an output folder in MSI's global scratch space that will hold the files of our analysis. This is command **2** in the text file:

```
mkdir -p /scratch.global/${ME}/RNAseq_Tutorial/Out
```

CHURP is available on MSI systems as part of the software module system. Use the `module` command to access these packages. We will first check for the version (s) of CHURP that are available on MSI systems:

```
module avail churp
-------------------------------- /panfs/roc/soft/modulefiles.common --------------------------------
churp/0.2.2-slurm  churp/0.2.3
```

There are two versions of CHURP available: `0.2.2-slurm` and `0.2.3`. We will load the latest version of the module. This is command **3** in the text file:

```
module load churp/0.2.3
Latest changes: 2022-05-11
 - agsmall (Agate) and amd512 (Mangi) partitions added
 - Option --command-log added to group_template
 - Option --tmp for requesting local scratch space during processing

!!! NOTE !!! If you use Agate, we recommend you request 4000MB of memory per CPU

!!! NOTE !!! To submit jobs to Agate, you must be logged in to an Agate login node (ssh agate.msi.umn.edu)
This is CHURP version 0.2.3.
CHURP can be called by running $CHURP.
```

Note that we include the version information in the `module load` command. You should always make a note of the versions of the software packages you are using! Default module versions change on MSI, so omitting this information puts you at risk for being unable to reproduce your results!

We will verify that the module works by running CHURP with no arguments. This is command **4** in the text file:

```
$CHURP
Usage: churp.py <subcommand> <options>

where <subcommand> is the name of the pipeline that is to be run. The specified
<options> will be applied to the operations in the pipeline. Each pipeline has
its own set of options that must be specified. To see a full listing of each
available option for a given pipeline, pass the '--help' option. Alternately,
online help is maintained at the GitHub repository.

Currently, the following subcommands are supported:
    - group_template
    - bulk_rnaseq

For issues, contact help@msi.umn.edu.
Version: 0.2.3
2022-05-06
```

The message above tells us that CHURP has loaded successfully and works - it is just waiting for us to tell it what to do!

To view more information about accessing MSI's software modules, please see this link:
https://www.msi.umn.edu/content/accessing-software-resources

To view the software catalogue maintained by MSI, see this link:
https://www.msi.umn.edu/software

Return to top

Part 5: Generate Experimental Data Sheet

Goal: By the end of this section, you should be able to generate an experimental metadata sheet and edit it to specify the experimental groups in your study. You will also make a directory in the global scratch space to hold your analysis output.

Part 5.1 CHURP Routine for Metadata Sheet Generation

We will then use the `group_template` routine in CHURP to generate a template for the experimental metadata CSV file. We will first check the usage message for the routine by running without any options:

```
$CHURP group_template
----------
Thank you for using CHURP. [ . . . ]
```

We have to specify which pipeline we want to use for generating an experimental metadata file. We will choose the `bulk_rnaseq` pipeline and ask for the help message with `--help`:

```
$CHURP group_template bulk_rnaseq --help
usage: churp.py group_template bulk_rnaseq --fq-folder <fastq folder> [--help]
                                           [--verbosity <loglevel>]
                                           [--output <output file>]
                                           [--extra-column <extra column>]

Required arguments:
  --fq-folder <fastq folder>, -f <fastq folder>
                        Directory that contains the FASTQ files.

Optional arguments:
[ . . . ]
```

We see from the help message that we need to specify the path to the FASTQ folder. For this tutorial, use the following path for the FASTQ folder:

```
/home/msistaff/public/RNAseq_Tutorial/Reads
```

We will also specify an output file located in the global scratch directory that we made at the start of this section. Supply these options to the `group_template` routine. This is command **5** in the text file:

```
$CHURP group_template bulk_rnaseq \
    -f /home/msistaff/public/RNAseq_Tutorial/Reads \
    -o /scratch.global/${ME}/RNAseq_Tutorial/Out/Groups.csv
----------
Thank you for using CHURP. [ . . . ]
```

CHURP allows you to specify additional metadata columns with the `-e` option. This option can be specified multiple times for multiple columns; for example, to add columns for `Age` and `Sex`, you can include `-e Age -e Sex` in the command. This makes it easy to build a metadata CSV that can accommodate a more complicated experimental design. Our automated testing, however, only considers the `Group` column. For analysis of more complicated experiments, please consult with a statistician or with us at RI Bioinformatics!

Part 5.2 Specifying Groups in the Metadata Sheet

The experimental metadata file is auto-populated with `NULL` values for the assignments. We do not make assumptions as to which group a given sample belongs to - the researcher who is analyzing the data is required to make that decision. Let us make assignments by editing the CSV template file. Open the file in the `nano` text editor. This is command **6** in the text file:

```
nano /scratch.global/${ME}/RNAseq_Tutorial/Out/Groups.csv
```

The CSV file will look like what is below:

```
SampleName,Group
BoneMarrow-1,NULL
BoneMarrow-2,NULL
BoneMarrow-3,NULL
BoneMarrow-4,NULL
Spleen-1,NULL
Spleen-2,NULL
Spleen-3,NULL
Spleen-4,NULL
```

Replace the `NULL` values with the correct group assignments. Be sure to not edit the first line of this file; the column names and column order are required to stay the same for our downstream analyses. The file should look like it does below:

```
SampleName,Group
BoneMarrow-1,BoneMarrow
BoneMarrow-2,BoneMarrow
BoneMarrow-3,BoneMarrow
BoneMarrow-4,BoneMarrow
Spleen-1,Spleen
Spleen-2,Spleen
Spleen-3,Spleen
Spleen-4,Spleen
```

Note that the group labels are **case-sensitive**. `BoneMarrow` is interpreted as a different group than `bonemarrow`.

Once you are done editing the file, press `Ctrl` + `o` (letter O, not number zero), then `Enter` to save the file. Then, press `Ctrl` + `x` to exit the editor.

Note that if you give a sample a `NULL` group assignment, then it will *not* be used for differential gene expression testing. You will still get expression counts and QC metrics, though.

CHURP will perform automated differential gene expression testing with up to **four** groups maximum. It will also only perform automated differential expression testing if and only if all groups have at least **three** replicates. If your experiment does not conform to these parameters, then you will have to perform differential expression testing yourself with the expression matrix produced by CHURP.

Return to top

Part 6: Submit Analysis Jobs

Goal: By the end of this section, you should be able to verify the information in the CHURP `samplesheet.txt` file, submit the CHURP jobs, and check their status in the job queue.

Part 6.1: Generating Pipeline Job Script

We will now use the `bulk_rnaseq` routine in CHURP to analyze the tutorial RNAseq dataset. This is command **7** in the text file. If you copy and paste any command in this tutorial, make it this one:

```
$CHURP bulk_rnaseq \
    -e /scratch.global/${ME}/RNAseq_Tutorial/Out/Groups.csv \
    -f /home/msistaff/public/RNAseq_Tutorial/Reads \
    -x /home/msistaff/public/RNAseq_Tutorial/Reference/GRCm38_19 \
    -g /home/msistaff/public/RNAseq_Tutorial/Reference/Annotations.gtf.gz \
    -o /scratch.global/${ME}/RNAseq_Tutorial/Out \
    -d /scratch.global/${ME}/RNAseq_Tutorial/Work \
    --strand U -q agsmall --ppn 4 --mem 16000 -w 2 --no-submit
----------
Thank you for using CHURP. [ . . . ]
```

The parameters specified here are specific for the tutorial dataset! Do not re-use these for an analysis of a real dataset! In particular, the `--ppn 4` and `-w 2` parameters are for extremely small datasets, and the `--strand U` parameter is for a specific type of library preparation.

The default RNAseq library preparation kit used by the UMGC is the Illumina TruSeq Stranded mRNA kit, which uses `RF` strand specificity. This means R1 maps to the "reverse" strand and R2 maps to the "forward" strand. This is the **default** strand specificity for CHURP.

For low input or marginal quality RNA, the UMGC may use the SMARTer Stranded RNAseq v2 kit from Takara Bio. This kit also has `RF` strand specificity, but **requires a "headcrop" of 3bp** (`--headcrop 3` for CHURP).

If you have older data that was from the SMARTer Stranded RNAseq v1 kit, it has `FR` strand specificity, and also requires `--headcrop 3` for CHURP.

Your data release email from UMGC will specify which kit was used to prepare your libraries. If you are analyzing data from other sources, you will have to consult the manual of the kit that was used to prepare your data to know the strand specificity. The libraries used in this tutorial come from the "NEBNext Ultra RNA," which is **not** strand-specific, which is why we use `--strand U` in the tutorial.

Telling CHURP the strand specificity does **not** affect read mapping; it only affects the way fragments are assigned to genes during the quantification step.

We supply a lot of special options for the tutorial run. Descriptions of the options are given below:

- `-o /scratch.global/${ME}/RNAseq_Tutorial/Out`
  - Specify path to output directory
  - Default behavior is to put it in global scratch with a name based on the system clock time
- `-d /scratch.global/${ME}/RNAseq_Tutorial/Work`
  - Specify path to working directory
  - Default behavior is to put it in global scratch with a name based on the system clock time
- `--strand U`
  - Count reads in a **strand non-specific** way.e
  - Default is reverse-stranded-specific
- `--ppn 4`
  - Use **4** CPUs for multithreaded processes
  - Ddefault is 6
- `--mem 16000`
  - Use 16000MB (16GB) of RAM
  - Default is 12000MB (12GB)
- `-w 2`
  - Use **2** hours of time
  - Default is 12
- `--no-submit`
  - Do not automatically submit pipeline jobs
  - Default is to automatically submit jobs

There are three lines of text output that you should note. The text that is written below has truncated paths, but you will see lines that look like this:

```
Pipeline script: /scratch.global/. . . bulk_rnaseq.pipeline.sh
Samplesheet: /scratch.global/. . . samplesheet.txt
Qsub array key: /scratch.global/. . . qsub_array.txt
```

CHURP submits a *job array* for the processing of the dataset, because the processing of one sample is independent of the processing of another. This allows the jobs to start sooner; each job is smaller so it can be split across partially-allocated nodes. To read more about job arrays, see this link: https://www.msi.umn.edu/support/faq/how-do-i-use-job-array

The `pipeline.sh` file is a shell script that submits the pipeline jobs to the scheduler. The `samplesheet.txt` file is a specially-formatted text file that describes the parameters of the analyses. The `qsub_array.txt` file is a text file that contains a key that associates each sample with an array index.

Part 6.2 Verifying Metadata

Let us take a look at the `samplesheet.txt` file with `less -S` to make sure that the groups were assigned properly (path shortened for readability):

```
less -S /scratch.global/. . . samplesheet.txt
```

The file is a bit complicated, and is not really intended for easy reading by humans! But we will look at the first two columns separated by the pipe character (`|`) to verify that the samples are assigned to the correct groups. The first column is the sample name and the second column is the group:

```
BoneMarrow-1|BoneMarrow|...
BoneMarrow-2|BoneMarrow|...
BoneMarrow-3|BoneMarrow|...
BoneMarrow-4|BoneMarrow|...
Spleen-1|Spleen|...
Spleen-2|Spleen|...
Spleen-3|Spleen|...
Spleen-4|Spleen|...

# Generated by CHURP version 0.2.2-slurm
# Generated at 2021-02-03 15:48:07
#0
```

These look good! Now, execute the `pipeline.sh` script with `bash` to submit the pipeline jobs (path shortened for readability):

```
bash /scratch.global/. . . pipeline.sh
sbatch: Setting account: msistaff
sbatch: Setting account: msistaff
Output and logs will be written to /scratch.global/YOUR_INTERNET_ID/RNAseq_Tutorial/Out
Emails will be sent to YOUR_INTERNET_ID@umn.edu
```

```
Qsub array to samplename key: /scratch.global/YOUR_INTERNET_ID/RNAseq_Tutorial/Out/. . .qsub_array.txt
Single samples job array ID: 802738
Summary job ID: 802739
```

The output message shows us the IDs of the jobs that were submitted and also where important files will be saved. We will revisit these directories in a later section!

Part 6.3 Viewing Queued Job Status

Let us verify that the jobs are submitted by running the `squeue` command, which prints the job queue status. We will also include the `-r` option (expand the array to show all elements), and the `--me` option, which shows only jobs that you own:

```
squeue -r --me
             JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
            802739     small run_summ <blanked>  PD       0:00      1 (Dependency)
          802738_1     small bulk_rna <blanked>  R        7:46      1 cn0511
          802738_2     small bulk_rna <blanked>  R        7:46      1 cn0517
          802738_3     small bulk_rna <blanked>  R        7:46      1 cn0061
          802738_4     small bulk_rna <blanked>  R        7:46      1 cn0061
          802738_5     small bulk_rna <blanked>  R        7:46      1 cn0061
          802738_7     small bulk_rna <blanked>  R        7:46      1 cn0062
          802738_8     small bulk_rna <blanked>  R        7:46      1 cn0062
```

You see nine jobs total: there are eight for the individual samples and one for the summary. The eight single-sample jobs are in the “Running” state, indicated by the `R` in the `ST` column. The summary job is in the “Pending” state, indicated by the `PD` in the `ST` column. The reason that the summary job is still pending is `Dependency`: it depends on all of the single-sample jobs to finish before it can start. You will be notified by email of job state changes, like “Pending” to “Running” and then from “Running” to “Complete”.

It is a good idea to set up an email filter for the messages from the job scheduler! They are sent from `msi_slurm@msi.umn.edu`, so you can use this address to send the messages to a special email folder. You can view more about the email settings for a job on MSI’s job sub submission page: https://www.msi.umn.edu/content/job-submission-and-scheduling-slurm

Each of the single-sample jobs is running through the left panel of the workflow depicted in the figure of Part 2.

Return to top

Part 7: Overview of CHURP Results

Goal: By the end of this section, you should be able to verify that your CHURP jobs have finished running and identify the major components of the CHURP output.

You should have gotten several emails from the scheduling system pertaining to your CHURP run. If the title of the email has the word `FAILED` in it, then the job did not complete successfully, and you will need to dig into the log files to see what went wrong. We cover the log files and how to read them in [Part 8] (#8).

Recall from Part 6 that the output directory we are using is

```
/scratch.global/${ME}/RNAseq_Tutorial/Out
```

Navigate to this directory and use `ls` to view the contents:

```
cd /scratch.global/${ME}/RNAseq_Tutorial/Out
ls
2021-02-03.YOUR_INTERNET_ID.bulk_rnaseq.pipeline.sh
2021-02-03.YOUR_INTERNET_ID.bulk_rnaseq.qsub_array.txt
2021-02-03.YOUR_INTERNET_ID.bulk_rnaseq.samplesheet.txt
allsamples_work_directory
Bulk_RNAseq_Report.html
bulk_rnaseq_single_sample-802738.1.err
bulk_rnaseq_single_sample-802738.1.out
bulk_rnaseq_single_sample-802738.2.err
bulk_rnaseq_single_sample-802738.2.out
bulk_rnaseq_single_sample-802738.3.err
bulk_rnaseq_single_sample-802738.3.out
bulk_rnaseq_single_sample-802738.4.err
bulk_rnaseq_single_sample-802738.4.out
bulk_rnaseq_single_sample-802738.5.err
bulk_rnaseq_single_sample-802738.5.out
bulk_rnaseq_single_sample-802738.6.err
bulk_rnaseq_single_sample-802738.6.out
bulk_rnaseq_single_sample-802738.7.err
bulk_rnaseq_single_sample-802738.7.out
bulk_rnaseq_single_sample-802738.8.err
bulk_rnaseq_single_sample-802738.8.out
Coordinate_Sorted_BAMs
Counts
DEGs
gene_id_gene_name_map.txt
Groups.csv
InsertSizeMetrics
Logs
Mus_musculus.GRCm38.100.gtf.gz
Plots
run_summary_stats-802739.err
run_summary_stats-802739.out
singlesamples_work_directory
```

where `YOUR_INTERNET_ID` is your UMN Internet ID. There is a lot here, so we will go through it in pieces, and each piece will be discussed in detail in its own section. If your output folder has fewer files than the listing above, then either your run is not yet complete or encountered an error.

The major components of the CHURP output are as follows:

- Log files
- Insert size metrics (only for paired-end data)
- Gene expression matrix
- Differentially-expressed genes (only if metadata sheet with group assignments was supplied)

- HTML summary report
- Coordinate-sorted BAM files

These files are written to global scratch, and will be removed if they are not copied to your group's MSI directory! When you run the workflows on your actual dataset, be sure to copy the relevant parts of the output into your group's directory to save them.

The log files are useful for identifying problems with a CHURP run. The insert size metrics are important for submitting data to public databases, like the NCBI Gene Expression Omnibus resource. The gene expression matrix is used for expression analyses, including differential gene expression testing. THe HTML summary report includes quality control information and an overview of the workflow applied to the dataset. The coordinate-sorted BAM files are useful for visualization or other genetic analyses that you may want to perform.

Return to top

Part 8: CHURP Log Files

Goal: By the end of this section, you should be able to view logs of the CHURP run and know where to look if there was an error during one of the jobs.

CHURP produces three types of log files during its run. Two of these types of logs are written in the `Logs/` sub-directory of the output folder that you specified. These logs are produced by the CHURP script and the tools that the script calls. The third type of log file is written to the root of the output folder, and contains scheduler and job progress information. We will start by looking at the log files in the root of the output folder.

Part 8.1 Scheduler Log Files

The log files from the scheduler have names that end with `.out` and `.err`. We have written CHURP to write the job progress information to the `.err` files; you will see one for each sample and one for the summary job. We will look at the first one with `less` (your filename will be different from the example below):

```
less bulk_rnaseq_single_sample-802738.1.err
```

You can use the arrow keys to scroll around in the file to read the contents. The information written here is mostly interesting from a bookkeeping perspective; you can see how long each step took from the time stamps written into the file. For example, in my file:

```
# 2021-02-04 11:49:53: Entering section HISAT2
# 2021-02-04 11:50:42: Finished section HISAT2
```

we can see that HISAT2 mapping started at 11:49:53 and ended at 11:50:42 for this sample, meaning that HISAT2 took a little less than one minute to map the reads. These time stamps can give you clues as to where a sample processing bottleneck may exist. Press `q` to quit out of the viewer.

Remember that the tutorial dataset is very small! The time required to process your full dataset will likely take at least several hours.

Part 8.2 Analysis Log Files

The next log file we will examine is located in the `Logs/` sub-directory of the output folder. Navigate to this directory and list the contents:

```
cd /scratch.global/${ME}/RNAseq_Tutorial/Out
ls
BoneMarrow-1_Analysis.log
BoneMarrow-1_Trace.log
BoneMarrow-2_Analysis.log
BoneMarrow-2_Trace.log
BoneMarrow-3_Analysis.log
BoneMarrow-3_Trace.log
BoneMarrow-4_Analysis.log
BoneMarrow-4_Trace.log
BulkRNASeq_Analysis.log
BulkRNASeq_Trace.log
Spleen-1_Analysis.log
Spleen-1_Trace.log
Spleen-2_Analysis.log
Spleen-2_Trace.log
Spleen-3_Analysis.log
Spleen-3_Trace.log
Spleen-4_Analysis.log
Spleen-4_Trace.log
```

You will see two log files per sample, one ending with `Analysis.log` and one ending with `Trace.log`. Use `less` to view the `BoneMarrow-1_Analysis.log` file:

```
less BoneMarrow-1_Analysis.log
```

You can use the arrow keys to scroll around in the log file. Every line starting with the number sign (`#`) is intended to be a human-readable message regarding the behavior of CHURP. Lines that do not start with number signs are messages from the tools that CHURP uses to process a dataset.

The first lines show you the sample name and the job ID for the analysis. We also list all of the software modules and versions that were loaded to perform the analysis. You will not need to memorize nor copy this list, but do know that it is available for you to include in manuscript methods.

If your job encountered an error with one of the samples, then this log file will contain messages at the very end that are relevant to the error. These can give you clues as to problems with your dataset; if you would like our help with addressing the issues, please contact the MSI Help Desk and attach the log file!

Part 8.3 Trace Log Files

The third type of log file that CHURP produces is a trace log file. View the trace log file for the `BoneMarrow-1` sample:

```
less BoneMarrow-1_Trace.log
```

This log file is *not* intended to be as human-friendly as the analysis log file. The trace log file contains an account of every command that is executed in the course of the workflow. If you are interested in seeing the exact commands that were run, then this file will show them to you. Most researchers would not be interested in such detail, but we do provide them for interested parties.

Return to top

Part 9: Insert Size Metrics

Goal: By the end of this section, you should be able to extract summary statistics about the insert size distributions of your libraries.

If you ran CHURP with paired-end data, then you will have a sub-directory in your output folder for per-sample summaries of the insert sizes. This information is important for checking that your reads map in the expected configuration, given your library preparation and sequencing. It is also important for submission of data to public databases: many repositories of datasets require summary statistics of insert size distributions for paired-end data.

An overview and graphical summary of these data are presented in the HTML report (see Part 11), but we will look at more detailed information in this section.

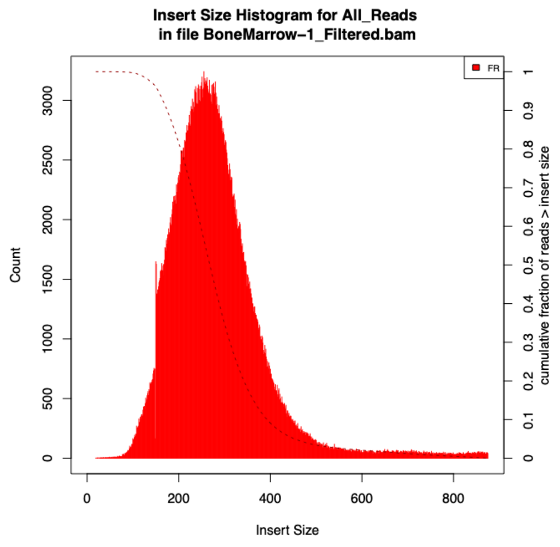
Navigate to the sub-directory in your CHURP output folder:

```
cd /scratch.global/${ME}/RNAseq_Tutorial/Out/InsertSizeMetrics
```

List the contents of this directory to see what files are written for each sample:

```
ls
BoneMarrow-1_hist.pdf
BoneMarrow-1_metrics.txt
BoneMarrow-2_hist.pdf
BoneMarrow-2_metrics.txt
BoneMarrow-3_hist.pdf
BoneMarrow-3_metrics.txt
BoneMarrow-4_hist.pdf
BoneMarrow-4_metrics.txt
Spleen-1_hist.pdf
Spleen-1_metrics.txt
Spleen-2_hist.pdf
Spleen-2_metrics.txt
Spleen-3_hist.pdf
Spleen-3_metrics.txt
Spleen-4_hist.pdf
Spleen-4_metrics.txt
```

We see that there are two files for each sample, a `hist.pdf` file and a `metrics.txt` file. The `hist.pdf` file contains a histogram of insert sizes for the sample. The histogram for sample `BoneMarrow-1` is presented below:



You should check that this distribution matches what was expected based on your molecular protocol QC. Unfortunately, for the tutorial dataset, we do not have access to such information.

The histogram is nice, but to describe the distributions in writing, we will need textual and numerical summaries of the insert sizes. We can find these in the `metrics.txt` files. Use `less` to examine the metrics of sample BoneMarrow-1:

```
less BoneMarrow-1_metrics.txt
```

This file has very long lines, so you will likely have to use the arrow keys to scroll side-to-side to read the information. We will not cover the meaning of every field in this file, but you can get instructions for how to read this file from the Picard Tools documentation:
https://broadinstitute.github.io/picard/picard-metric-definitions.html#InsertSizeMetrics

Press `q` to quit out of the viewer when you are done.

Return to top

Part 10: Expression Counts Matrix

Goal: By the end of this section, you should be able to identify the raw read counts matrix output and understand what information it contains. You should also be able to identify which analyses are possible and which tools to use for those.

The expression counts matrix is one of the most important output files from CHURP. This file contains the *read counts* that are used for differential expression testing, coexpression network analysis, or other clustering analyses based on gene expression. We will focus on the *raw* read counts matrix, because this is file that you should use to begin any expression analyses that you will perform.

The counts matrix that CHURP produces is generated with the `featureCounts` program from the `subread` package. It produces counts in a strand-specific manner, which is why it is important to know the specificity of your library protocol when you analyze the data. To read more about `featureCounts` and `subread`, see their user guide, linked on their homepage:
http://subread.sourceforge.net/

Navigate to the sub-directory of the output folder for the expression counts matrix:

```
cd /scratch.global/${ME}/RNAseq_Tutorial/Out/Counts
```

List the contents of this directory to see the files:

```
ls
cpm_list.txt
subread_counts_gene_symbol.txt
subread_counts.txt
subread_counts.txt.summary
```

There are four files that describe the results of the read counting. We will focus on just the *raw read counts* matrix, which is of primary interest. These are stored in the `subread_counts.txt` file.

The other files may be of interest, too, but they require careful understanding of their contents. The `cpm_list.txt` file contains filtered and normalized counts in "*log base 2 counts per-million*" values. This file is not of primary interest for data analysts because the data have been processed slightly, and results generated from this file may not be reproducible.

The `subread_counts_gene_symbol.txt` file contains *raw read counts* as well, but the stable gene identifiers have been replaced with gene symbols. These may be more familiar to researchers, but gene nomenclature can be inconsistent, and even vitiates analyses done with Excel:
https://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-1044-7

The `subread_counts.txt.summary` file contains high-level summaries of the read assignments to gene features. This file is summarized graphically in the HTML report, so we do not need to examine it here.

If you want to analyze the "log(CPM)" values from edgeR, for example, to visualize the per-sample expression levels of DEGs, do keep the following in mind. In some cases, the "log(CPM)" value is written as "log(1+CPM)" to reflect the fact that a pseudocount of 1 has been added to the raw fragment counts, but this is not an accurate name. The numerical values are actually defined by the following expression:

```
log2([raw counts + 1] / [library size in millions of fragments])
```

This means it is possible that there are *negative values* for specific genes, which would be impossible if the value were literally "log(1+CPM)." For example, in a library with 20 million fragments, a gene may have six fragments mapped to it. This gene's "log(1+CPM)" value is then:

```
  log2([6 + 1] / 20)
= log2(7 / 20)
= -1.514573
```

The reasoning from the edgeR developers is that adding a pseudocount of `1` to the fragment counts before normalization has a very minor effect on the inferred expression responses, but adding `1` to the CPM value before taking the log has a large effect, because it effectively changes the order of magnitude of the expression value.

Look at the `subread_counts.txt` file with `less` to see what format the raw counts are in. We will additionally use the `-S` option to disable line-wrapping:

```
less -S subread_counts.txt
```

The first line shows the version of `featureCounts` that was used to produce the file and the command that was run. You do not need to do anything with this line, unless you are interested in running a modified command line.

The second line shows the names of the columns. The first six columns contain information related to the gene models, such as gene identifier, position, and strand. The seventh column until the final column contain the counts for each sample.

These counts are generated at the *gene* level, meaning the counts are aggregated and assigned on a gene-by-gene basis. We do not support counting at the *transcript-level*, because transcript-level expression requires very specific data and analytical routines. If you are interested in transcript-level analyses (isoform expression), please contact RI Bioinformatics and/or the UMGC for consultation or make sure you already know how to collect transcript-level expression data!

We colloquially call them *raw read counts*, but they are really *raw fragment counts*. This is an important distinction - we require *both reads* of a pair to map to a gene model in order for it to be counted.

We will not perform any analysis with this file directly, least of all with a text file viewer. This is outside of the scope of this tutorial, but if you would like to read these values into R, you can use the following snippet:

```
subread_cts <- read.table("subread_counts.txt", header=TRUE, sep="\t")
raw_read_cts <- as.matrix(subread_cts[,-c(1:6)])
rownames(raw_read_cts) <- subread_cts[, 1]
```

These can then be used as the expression data for analysis with EdgeR or DESeq2. If you are interested in performing your own differential expression analysis, then you can refer to the guides for EdgeR and DESeq2:

- EdgeR: http://bioconductor.org/packages/release/bioc/html/edgeR.html
- DESeq2: https://bioconductor.org/packages/release/bioc/html/DESeq2.html

The experimental metadata CSV that you generated in Part 4 will also be helpful for running your own differential expression analyses.

Return to top

Part 11: Differentially Expressed Genes

Goal: By the end of this section, you should be able to identify the differentially expressed genes (DEGs) identified by CHURP and know which tests are being performed.

Note that the differential gene expression tests are only performed if you supplied non- `NULL` experimental group assignments to the samples. There must be at least two groups and at most four groups, with at least three replicates per group.

Navigate to the sub-directory in the output folder that contains the lists of differentially-expressed genes and list the contents:

```
cd /scratch.global/${ME}/RNAseq_Tutorial/Out/DEGs
ls
DE_Spleen-BoneMarrow_list.txt
```

There is one file here. It is not the best practice, but the filename has some interpretative meaning here. You will notice that the filename has a pair of the group names that we assigned in Part 4.2, in this case `Spleen` and `BoneMarrow`. These denote the groups that were compared for differential gene expression.

The general form the filename takes is `DE_GROUP2-GROUP1_list.txt`, and `GROUP1` is treated as the reference group and `GROUP2` is treated as the test group. Therefore, genes that have *positive* log(fold change) values are *up-regulated* in `GROUP2` with respect to `GROUP1`, and genes with *negative* log(fold change) values are *down-regulated* in `GROUP2` with respect to `GROUP1`. In this case, `BoneMarrow` is the reference group, and `Spleen` is the test group.

Use `less` to look at the file:

```
less DE_Spleen-BoneMarrow_list.txt
```

The first row has the names of the columns. There are six pieces of information for each gene:

1. Gene identifier
2. log2(fold change)

3. log2(CPM)
4. F test statistic
5. Raw P-value
6. FDR value

Only the genes that are significant at a 5% false discovery rate (FDR < 0.05) are contained in these files, so there will be substantially fewer genes in these files than are annotated on the genome assembly. Press `q` to quit out of the text viewer when you are done looking at the file.

The differential expression testing is carried out between all pairwise combinations of groups that were specified. For each pair of groups being compared, the reference group is the one that is first when *sorted lexicographically*, using the English alphabet.

The test that is done is the *quasi-likelihood F test* in EdgeR. You can read more about the quasi-likelihood F test in the EdgeR user manual:
https://bioconductor.org/packages/release/bioc/html/edgeR.html

The expression values are filtered before being used for differential gene expression testing with CHURP. The following filters are applied:

1. Genes with variance in raw counts that is less than 1 are removed.
2. Genes that are shorter than 200bp are removed (this can be adjusted with a command line argument).
3. Samples with `NULL` group assignments are removed.
4. Genes with low expression* are removed.

*: We define "low expression" as follows. Calculate *C*, the log(CPM) value that would be obtained from a raw count of 1 fragment in the smallest library. We then remove genes that have log(CPM) expression of less than *C* in *G* or more samples, where *G* is the size of the smallest group. The rationale with these filtering criteria is to restrict differential gene expression testing to only genes that are expressed in a majority of the samples.

These filtering criteria are **pretty conservative**, which is why I tend to advocate for researchers performing differential expression testing with human input, rather than relying on an automated workflow. For instance, a gene that is *completely unexpressed* in one experimental condition would be filtered prior to testing. Additionally, a series of pairwise comparisons is not always the most appropriate way to analyze data; for example, data from a dosage series or a time series would not be appropriately handled as a series of pairwise comparisons.

Return to top

Part 12: HTML Report

Goal: By the end of this section, you should be able to identify the key sections of the HTML report and identify the sections that contain information on rerunning the analysis.

Note that for this section, you will be required to have a web browser and a way to transfer files between your workstation and MSI servers. One easy way to transfer files is with Filezilla, for which MSI maintains a guide:

https://www.msi.umn.edu/support/faq/how-do-i-use-filezilla-transfer-data

The HTML report is another one of the pieces of CHURP output that is of primary interest. The report contains a description of the input data for the workflow, including the reference genome that was used for analysis, the path to the sequence reads, and the names of the samples that were included in the dataset. There are additionally several important summaries of data quality that you can view in the report. These will help you to interpret the results of your experiment or guide you on making adjustments to the analysis.

Navigate to the output folder and list its contents:

```
cd /scratch.global/${ME}/RNAseq_Tutorial/Out
ls
2021-02-03.YOUR_INTERNET_ID.bulk_rnaseq.pipeline.sh
2021-02-03.YOUR_INTERNET_ID.bulk_rnaseq.qsub_array.txt
2021-02-03.YOUR_INTERNET_ID.bulk_rnaseq.samplesheet.txt
allsamples_work_directory
Bulk_RNAseq_Report.html
bulk_rnaseq_single_sample-802738.1.err
bulk_rnaseq_single_sample-802738.1.out
bulk_rnaseq_single_sample-802738.2.err
bulk_rnaseq_single_sample-802738.2.out
bulk_rnaseq_single_sample-802738.3.err
bulk_rnaseq_single_sample-802738.3.out
bulk_rnaseq_single_sample-802738.4.err
bulk_rnaseq_single_sample-802738.4.out
bulk_rnaseq_single_sample-802738.5.err
bulk_rnaseq_single_sample-802738.5.out
bulk_rnaseq_single_sample-802738.6.err
bulk_rnaseq_single_sample-802738.6.out
bulk_rnaseq_single_sample-802738.7.err
bulk_rnaseq_single_sample-802738.7.out
bulk_rnaseq_single_sample-802738.8.err
bulk_rnaseq_single_sample-802738.8.out
Coordinate_Sorted_BAMs
Counts
DEGs
gene_id_gene_name_map.txt
Groups.csv
InsertSizeMetrics
Logs
Mus_musculus.GRCm38.100.gtf.gz
Plots
run_summary_stats-802739.err
run_summary_stats-802739.out
singlesamples_work_directory
```

where `YOUR_INTERNET_ID` is your UMN Internet ID. The file of interest here is the `Bulk_RNAseq_Summary.html` file. If you have a remote desktop connection to MSI, you can open this file with the `firefox` web browser to view it. Otherwise, please copy it to your local workstation then open it in your web browser.

The HTML report is large! If you have many samples, it will be even larger. While it is a useful file for keeping your data summaries in one place, do be aware of the logistical challenges its size may pose for sharing with collaborators.

We have put a sample of the report from the tutorial dataset analysis at this link:
https://pages.github.umn.edu/MSI-RIS/Tutorials_Beta/materials/rnaseq_cmd/Bulk_RNAseq_Report.html

This is a long file! The organization and format of the contents are still works in progress.

An example look through the QC report might be as follows. The values shown and interpretations provided are merely for illustration. As with any other data summary, the meanings of these values will change depending on your expectations and experimental design.

1. Scroll to section 2.1 in the report, "Experiment Summary." Check the following:
   - Does the experiment contain the correct number of samples?
   - Are the sample names correct?
   - Is the reference genome the correct assembly and version?
2. Scroll to section 2.2, "Fragment Counts Plot." This plot displays the read counts before trimming and after trimming. Check the following:
   - Do any of the samples have unusually high or low fragment counts?
   - Do any of the samples lose a lot of reads after trimming?
3. Scroll to section 2.3, "Read Quality Plots." The plots in this section display an overview of the mean base quality scores across the read. Check the following:
   - Are there reads with large stretches of low quality bases (yellow or red)?
   - Are R1 and R2 properly detected for each sample?
4. Scroll to section 2.4, "HISAT2 Plots." The plots in this section display summaries of the HISAT2 mapping of the trimmed reads against the reference genome. Check the following:
   - Are there any samples with many more unmapped reads than the others? Or discordantly mapped reads?
   - Of the mapped reads in each sample, do most of them map uniquely, or multiply?
5. Scroll to section 2.5, "Expression Plots." The plots in this section display summaries of the estimated gene expression in the experiment. Check the following:
   - Are the number of expressed features approximately the same in each sample?
   - Do the "CPM" distributions have roughly the same shape for each sample?
   - Do the samples cluster as you would expect in the clustering heatmap?
   - Can you identify the major axes of differentiation among the samples in the MDS plot? Is it what you expect it to be? For example, to the samples cluster by a biological character that "makes sense?"
6. Navigate to section 4.2, "rRNA Contamination (Based on Kmer Matching)." This section shows an estimate of the proportion of reads in each sample that come from ribosomal sequences. Check the following:
   - Are there samples that have an unusually high or low proportion of rRNA reads?

   - Are all of the samples below about 0.4*?
7. Scroll to section 4.3, "Mapping-based QC Metrics." This section contains plots that summarize how well the RNAseq data covers the protein coding portion of the genome. Check the following:
   - Are there any samples that have especially low exon profiling efficiency? Are all of the samples above 0.4*?
   - Do any of the samples have especially high sequence duplication levels?
   - Do the "strand-aware" and "unstranded metrics" barplots look very different from each other? Are there a lot of "intergenic" reads?

8. Navigate to section 5.3, “Insert Size Metrics.” This section contains a high-level summary of the insert size metrics that we examined earlier in this tutorial. Check the following:
    - Do all of the samples have similar insert size summary statistics?
9. Navigate to section 3.3, “Pipeline Script.” This section contains a verbatim reproduction of the `pipeline.sh` file that was produced by CHURP. Click the arrow to show the pipeline script. You can copy and paste this into a **plain text editor** (Notepad, TextWrangler, Notepad++, etc) and save it for rerunning an analysis.
10. Scroll to section 3.4, “Samplesheet.” This section contains a verbatim reproduction of the `samplesheet.txt` file that was produced by CHURP. Click the arrow to show the samplesheet. You can copy and paste this into a **plain text editor** and save it for rerunning an analysis.

*: These are not magic numbers nor general recommendations! They are simply alues that we have found to be worrisome from a data quality standpoint. Depending on your library type, source material quality, and sequencing target, these values may be inappropriate. Always think about your experiment when assessing data quality!

To rerun the workflow to regenerate the results, you must do the following:

1. Make sure the **output** and **working** directories defined in the `pipeline.sh` script exist.
2. Make sure that FASTQ input folder is the same as when you first ran the workflow. If you are using data from the UMGC, then this path should be stable. If you are using data from the SRA, you will have to be careful with it.
3. Make sure the reference genome index and GTF annotation are in the same location and have the same name as when you first ran the workflow.
4. Make sure you have saved the `samplesheet.txt` file as *plain text*, and that it has the same name and path as those defined by `pipeline.sh`.
5. Make sure you have saved the `pipeline.sh` file as *plain text*.

Once you have verified that the above conditions, run the `pipeline.sh` script with `bash`:

```
bash /path/to/pipeline.sh
```

This will resubmit the workflow jobs and recreate the output files.

</div>

Return to top

Part 13: Feedback

This tutorial document was prepared by Thomas Kono, in the RI Bioinformatics group at MSI.Please send feedback and comments to konox006 [at] umn.edu. You may also send tutorial delivery feedback to that address.

Return to top

Supplementary Information

The information in the following sections is not part of the primary tutorial. We provide it only to discuss and illustrate issues and operations that are relevant to RNAseq analysis, but only come up in special circumstances.

Supplement 1: Custom HISAT2 Indices

In the cases where your study organism does not have a reference genome in the sections of Ensembl that we include in bioref, you will need to prepare your own index files for read mapping. You may also want to prepare a custom index if you want to restrict the genome to a specific set of regions (like we did for the tutorial), or if you have an experiment that involves a transgenic organism with a known construct transformed into it, and you want to map reads to the genome as well as the construct.

To do this, you will need at minimum the FASTA file that contains the genome sequence for your study organism. If you have the GTF annotation file, this can aid in preparing the HISAT2 index, as well. Known single nucleotide polymorphism sites (from dbSNP or VCF) can also be used.

You can find a full list of options for HISAT2 index building in their manual:
http://daehwankimlab.github.io/hisat2/manual/#the-hisat2-build-indexer

To build the index, you must be logged in to an interactive compute session:

```
srun -N 1 -n 1 -c 1 -t 60 --mem=4gb --tmp=2gb -p interactive --pty bash
```

Make sure the genome file is unzipped; the HISAT2 index builder does not take compressed FASTA input. Place this file into a directory where you would like to store the HISAT2 index files. Do the same with the GTF annotations, if you have them:

```
mkdir -p /home/GROUP/shared/HISAT2_Genomes/Genus_species
cp /path/to/genome.fasta.gz /home/GROUP/shared/HISAT2_Genomes/Genus_species
cp /path/to/annotations.gtf.gz /home/GROUP/shared/HISAT2_Genomes/Genus_species
gzip -d /home/GROUP/shared/HISAT2_Genomes/Genus_species/genome.fasta.gz
gzip -d /home/GROUP/shared/HISAT2_Genomes/Genus_species/annotations.gtf.gz
```

where `GROUP` is your MSI group name.

If you do not have a *GTF* annotation file, and instead only have a *GFF* annotation file, you can convert the GFF to a GTF using the `gffread` utility from the `cufflinks` package. This is a common scenario with reference genome assemblies from NCBI:

```
module load cufflinks/2.2.1
gffread annotations.gff3 -T -o annotations.gtf
```

Load the HISAT2 module and navigate to the genome directory:

```
module load hisat2/2.1.0
cd /home/GROUP/shared/HISAT2_Genomes/Genus_species
```

If you want to use known splice sites to help with spliced alignment, use the `extract_splice_sites.py` script that is bundled with HISAT2 to extract the splice sites from the GTF:

```
extract_splice_sites.py annotations.gtf > splice_sites.txt
```

If you would like to use known polymorphic sites to help map reads across known polymorphisms in your study organism, you must have either a dbSNP database export (for humans), or a VCF with known polymorphisms. If you have a dbSNP export, make sure the chromosome names match the genome you are using (UCSC, Ensembl, and NCBI all have different naming conventions), then use the `hisat2_extract_snps_haplotypes_UCSC.py` script to extract them:

```
hisat2_extract_snps_haplotypes_UCSC.py genome.fasta dbSNP.txt hisat2_snps
```

If you have a VCF, you can use the `hisat2_extract_snps_haplotypes_VCF.py` script to do the same thing:

```
hisat2_extract_snps_haplotypes_VCF.py genome.fasta SNPs.vcf hisat2_snps
```

Then, you are ready to build the genome index. **This step should be done as a non-interactive batch job.** It requires a lot of time and memory. For example, to build the human genome index with known splice sites and known polymorphisms, it takes at least several hours and about 200GB of RAM. The script would look something like the following. Remove the option for `--ss` if you do not have known splice sites, and remove options for `--snp` and `--haplotype` if you do not have known polymorphic sites. Note that you may have to adjust the resource requests if you experience walltime errors or out-of-memory errors, and use your email address instead of `YOUR_INTERNET_ID`:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 16
#SBATCH --mem=500gb
#SBATCH -t 36:00:00
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --mail-user=YOUR_INTERNET_ID@umn.edu
#SBATCH -p ram1t
#SBATCH -e HISAT2_Build_%j.err
#SBATCH -o HISAT2_Build_%j.out

module load hisat2/2.1.0

cd /home/GROUP/shared/HISAT2_Genomes/Genus_species
hisat2-build -f \
    -p 16 \
    --ss splice_sites.txt \
    --snp hisat2_snps.snp \
    --haplotype hisat2_snps.haplotype \
    genome.fasta \
    hisat2_genome_idx
```

Submit the job with `sbatch` and wait for it to finish. There will eventually be eight files that make up the genome index, and they must all be present in the same directory and have the same prefix. The name of your genome index will then be `hisat2_genome_idx`, and you can use this with the `-x` option to CHURP's bulk RNAseq workflow:

```
...
-x /home/GROUP/shared/HISAT2_Genomes/Genus_species/hisat2_genome_idx
...
```

You can find a full list of options for HISAT2 index building in their manual:
http://daehwankimlab.github.io/hisat2/manual/#the-hisat2-build-indexer

Return to top

Supplement 2: Customizing CHURP Behavior

We try to provide “sensible defaults” for CHURP’s behavior, but as with any analytical workflow, there exist datasets that require adjustment to the parameters. Most of the time, these cases can be addressed by adjusting the options to the read mapping software or the trimming software. In Supplement 3, we provide instructions for adjusting the expression quantification if you need a different type of expression analysis than the default gene-level expression.

Supplement 2.1: Custom HISAT2 Options

CHURP provides the `--hisat2-opts` option for adjusting the HISAT2 options. This is required when handling datasets that do not match the “default” dataset that CHURP expects. For example, if your paired-end reads are “interleaved” - R1 and R2 are contained in the same file rather than in separate files. Another case might be if you are analyzing a dataset with different quality score encodings: at the time of this writing, Illumina quality scores are “PHRED+33” but older datasets are “PHRED+64.”

Wikipedia has a good description of the various types of quality score encodings that you may encounter in FASTQ format:
https://en.wikipedia.org/wiki/FASTQ_format#Quality

Data that is generated contemporary to this writing will be in “Sanger” scale, with “PHRED+33” encoding.

You may additionally want to change how HISAT2 handles discordant read pairs or read pairs that fail to map to the genome. All of these options are available to adjustment! Please see the HISAT2 manual for the full range of options that you can provide:
http://daehwankimlab.github.io/hisat2/manual/

The *default* options that we supply for HISAT2 are the following:

- `-p 6` (use 6 cores, adjustable with the `--ppn` option to CHURP, too)
- `--no-mixed` (do not map discordant pairs as separate single-reads)
- `--new-summary` (print machine-friendly summary)

To specify custom HISAT2 options, you **must** pass it as a quoted string. This is because CHURP unfortunately tries to treat the options meant for HISAT2 as its own, and throws errors. For example, the default options defined above would be passed as follows:

```
...
--hisat2-opts="-p 6 --no-mixed --new-summary"
...
```

Any adjustment to the HISAT2 options with `--hisat2-opts` **requires** that you set **all** of the HISAT2 options. For example, if you are handling data that has “PHRED+64” quality scores, you would specify `--phred64`, but you will then also have to specify the multithreading options and pair-handling options:

```
...
```

```
---hisat2-opts="-p 6 --phred64 --no-mixed"
...
```

We require this because it is impossible for us to verify every possible combination of HISAT2 options. It is also possible that the custom HISAT2 options are incompatible with downstream expression quantification routines. If you encounter errors while providing custom HISAT2 options, you must troubleshoot the errors independently.

Supplement 2.2: Custom Trimmomatic Options

CHURP also provides the `--trimmomatic-opts` option for adjusting the behavior of Trimmomatic. This would be required when handling data that have non-standard adapter sequences or if your data have a quality score encoding that is not "PHRED+33."

The default options string that we pass to Trimmomatic is below:

```
ILLUMINACLIP:4:15:7:2:true LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:18
```

Which performs the following steps:

1. `ILLUMINACLIP:4:15:7:2:true` : trim the standard Illumina adapters, allowing up to 4 mismatches, a palindrome clip threshold of 15 (when "read-through" into the adapter is detected in short fragments), minimum palindrome adapter length of 7, and keep both reads in the pair if a palindrome is detected.
2. `LEADING:3` : trim bases from the start of the read if they have quality scores lower than 3.
3. `TRAILING:3` : trim bases from the end of the read if they have quality scores lower than 3.
4. `SLIDINGWINDOW:4:15` : divide the read into 4-base wide sliding windows, removing windows where the mean quality score is below 15.
5. `MINLEN:18` : remove read if it is shorter than 18bp after the previous trimming steps have been applied.

Like with the custom HISAT2 options, these must be passed as a quoted string. The default options would look like the following:

```
...
--trimmomatic-opts="ILLUMINACLIP:/path/to/adapters.fa:4:15:7:2:true LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:18"
...
```

Additionally, any adjustment to the Trimmomatic options requires that you specify the **entire** options string. The only exception is adding a `HEADCROP` operation, which can be done through CHURP by adding the `--headcrop` option (we provide this for handling the Takara Bio library preparation protocols).

Please see the Trimmomatic homepage to view the manual that gives the full range of options that are supported: http://www.usadellab.org/cms/?page=trimmomatic

Supplement 3: Transcript-level Analyses

By default, CHURP performs expression quantification at the *gene* feature level. This is the feature level that is targeted by most bulk RNAseq experiments and the feature level at which expression is most confidently measured. If you would like to instead measure *transcript* (isoform) feature expression, you will have to take special care with the molecular protocol and sequencing (data collection) as well as the downstream analytical workflow.

We cannot provide nearly complete scripts like we can with Supplement 1 because transcript-level analyses are highly specific to each study. Both the experimental design and the idiosyncrasies of the study system affect how the analysis is carried out. We instead provide points to consider when performing a transcript-level analysis. Of course, if you work in a research system that does not have alternative splicing, then this does not apply!

Supplement 3.1: Data Collection Considerations

Measuring isoform expression requires that you be able to distinguish the transcripts based on their splicing junctions or untranslated sequences. Additionally, detecting isoforms that have low expression requires deeper sequencing than detecting gene-level expression. You may also be interested in conditionally-expressed isoforms (tissue-specific, disease-specific, etc), which requires special experimental designs.

Unfortunately, we cannot provide general recommendations for sequencing depth nor sampling strategies to capture these cases. You should have a good sense of the “dynamics” of the genome of your study species - the distribution of the number of annotated isoforms for each gene model and the circumstances under which they are expressed will help to guide your experiments. The moments of that distribution will also guide you in sequencing depth; for example, if there are 4 isoforms per gene on average, you should expect to collect **at least** 4 times as much sequence data as for a gene-level experiment. The real number will be higher because not every isoform is expressed at the same level.

For human or mouse RNAseq experiments, this means you will need to collect at least 100 million read pairs (fragments) **per sample** for isoform level expression studies. If you want to assay the rare isoforms, then you likely will need to collect at least 200 million read pairs **per sample**. You may be able to use shorter read length to offset the cost of collecting so many reads, but that depends on the nature of the genes in your study system.

Supplement 3.2: Analytical Workflow Considerations

The read QC and trimming portions of the workflow can proceed as they would with CHURP. However, obtaining accurate transcript-level counts requires deviating from the CHURP workflow at the read mapping step. Instead of mapping read pairs to the reference genome with HISAT2, you should use a tool like RSEM (**R**NA**S**eq by **E**xpectation **M**aximization; Li and Dewey 2011; https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-12-323) to perform transcript quantification. The reason you need to change the read mapping workflow is that RSEM can estimate the transcript of origin for each fragment, while HISAT2 cannot. Because the multiple transcripts from a gene model are highly overlapping, the mapping-then-counting approach with HISAT2 and featureCounts cannot accurately estimate expression of the isoforms.

Please see the RSEM GitHub page (https://github.com/deweylab/RSEM#readme-for-rsem) for usage information and examples that show how to use it.

If you are confident that the reads from your experimental samples are **very similar** to the reference genome (less than 0.1% different at the nucleotide level), then you can use tools like Salmon (Patro et al. 2017; https://www.nature.com/articles/nmeth.4197) or Kallisto (Bray et al. 2016; https://www.nature.com/articles/nbt.3519) to estimate transcript abundances from reads. These two methods rely on very high similarity between the reads the reference genome, however, so we do not recommend them for general-purpose RNAseq, where you expect sequence differences between the experimental samples and the reference genome.

See the Salmon documentation (https://salmon.readthedocs.io/en/latest/) and the Kallisto documentation (https://pachterlab.github.io/kallisto/manual) for instructions on how to use their software to estimate transcript abundance.

All of RSEM, Salmon, and Kallisto are available on MSI systems as software modules:

```
module avail rsem
-------------------------------- /panfs/roc/soft/modulefiles.common --------------------------------
rsem/1.3.0
module avail salmon
--------------------------------- /panfs/roc/soft/modulefiles.hpc ----------------------------------
salmon/0.14.1  salmon/1.2.1
module avail kallisto
-------------------------------- /panfs/roc/soft/modulefiles.common --------------------------------
kallisto/0.43.1  kallisto/0.46.2
```

Transcript abundance estimates from the above-mentioned tools will **not** work with the same routines for differential expression testing as gene-level counts data. Gene-level expression is **count**-based (i.e., the data are integers), while transcript abundances are often reported in "TPM" (transcripts per million) values and "estimated counts" values. You can use the "estimated counts" values in a differential expression workflow built for analyzing counts data, like those in DESeq2 or EdgeR. If you use the TPM values, you can use a program like Sleuth (Pimentel et al. 2017; https://www.nature.com/articles/nmeth.4324; https://github.com/pachterlab/sleuth) to perform differential expression analysis.

Supplement 4: CPM, TPM, FPKM, RPKM, etc

RNAseq expression analyses are sometimes performed with *normalized relative expression* values rather than with *raw fragment counts* like we have done in this tutorial. Some analyses like coexpression network analyses, require normalized relative expression values rather than fragment counts.

Differential gene expression analyses should always be done with the raw fragment counts because the software packages that perform these analyses have statistical routines to properly model the distribution of expression values from fragment count data. The R packages discussed in this tutorial(`edgeR` and `DESeq2`) additionally perform robust cross-sample normalization, which none of the units described here will incorporate. These expression values should only be used as auxiliary to the results from the actual differential gene expression tests.

These units are also specific to the experiment or sequencing run. They are NOT comparable across experiments because they are very sensitive to the composition of the libraries. To properly analyze data from multiple experiments, you must take a *meta-analysis* approach; consult with a statistician!

Normalized relative expression values are useful for comparing genes or samples when there are differences in *sequencing depth* and *gene length*. For example, if you were to compare the fragment counts of a single gene across multiple samples, then both gene expression variation and sequencing depth variation will contribute to the fragment counts differences. On a related note, if you were to compare the fragment counts of different genes in the same sample (e.g., to check if knock-down of one gene leads to other gene expression responses), then both gene expression response and gene length differences will contribute to counts differences. This is because longer genes and transcripts have a larger "sequencing target" than shorter genes and transcripts. If two genes have the same transcript abundance, then the longer gene will have higher fragment counts simply because there is a higher representation of its transcript in the sequencing library.

One common relative expression value is normalized *counts per million*(CPM). To calculate CPM, first sum the fragment counts from all genes, and divide the sum by 1,000,000: this is the "per million" scaling factor. Then, for each gene, divide its fragment count by the scaling factor. This value is useful for comparing the *same gene across samples* because it accounts for sequencing depth variation. It is not useful for comparing *different genes within the same sample* because it does not account for gene length differences. Specifically with EdgeR, the values are output on the log2 scale, and a "pseudocount" is added to the raw counts value before normalizing by library size to avoid calculating `log2(0)`.

Another common relative expression value is *transcripts per kilobase-million* (TPM). This value is proportional to transcript relative abundance within a single sequencing library. For each gene or transcript, calculate a "counts per kilobase" value, which is the fragment counts divided by the length of the gene/transcript in kilobases (`length / 1000`). Sum the "counts per kilobase" values across all genes or transcripts and divide by 1,000,000: this is the "per million" scaling factor. Divide each gene or transcript's "counts per kilobase" value by the scaling factor. One feature of TPM values is that the sum of all per-feature TPM values will be equal across samples. This makes it useful for comparing relative expression values both between samples and within a sample.

Really, RPKM and FPKM should just be abandoned. We include them here just because some researchers still use them (and older publications report them).

Older units like *reads per kilobase per million* (RPKM; single-read data) and *fragments per kilobase per million* (FPKM; paired-end and single-read data) are very similar in spirit to TPM. The main difference is that TPM normalizes first by the gene or transcript length, then by sequencing depth, while RPKM and FPKM normalize first by sequencing depth then by gene or transcript length. While this difference seems minor, RPKM and FPKM do not have the property that the sum of expression values is the same for each sample. Thus, it is not possible to interpret FPKM or RPKM differences as proportional differences in transcript abundance. It is also worth noting that `Cufflinks` includes some probabilistic read mapping parameters in its FPKM calculations, so the FPKM values obtained from `Cufflinks` will be different than if you calculate them via the formula as written.

This was once a topic of much discussion, and you can read a nice summary of various expression values and their merits and weaknesses here:
https://haroldpimentel.wordpress.com/2014/05/08/what-the-fpkm-a-review-rna-seq-expression-units/

Supplement 5: Alignment-free Expression Analyses

- Kallisto
- Salmon
- RSEM

Supplement 6: Transcriptome Assemblies

- Considerations for data collection methods
- Trinity

Supplement 6.1: Functional Annotation

- Trinotate
- Dammit
- BLAST2GO

Supplement 7: Coexpression Analyses

- Experimental design considerations
- WGCNA

Supplement 8: Depositing Data into GEO

After you have finished analyzing your experiment and interpreting the results, you will likely want to publish your findings. Many journals require that your raw reads and processed expression data be made available in public repositories, like NCBI's Gene Expression Omnibus (GEO). GEO accession numbers may also be required **during peer review** so it is important to note this section as you are preparing your manuscript. GEO can also place a hold or embargo on your data until publication, so your data can remain private until you decide to release it.

Submitting your raw and processed data to GEO will make it available in **both** GEO and NCBI's Sequence Read Archive (SRA).

GEO says it can take up to five (5) business days to issue accession numbers for your dataset. Please plan accordingly!

The lab who is publishing the study should be the ones to manage the GEO submission. The submission process requires detailed knowledge of the experimental design, so the people who performed the experiment should be the ones to submit the data.

Required Data/Information

GEO requires both data and *metadata* for your submission. It is important that your metadata be complete and informative; these are what make datasets in public repositories useful.

You will also need to log in to GEO (requires an NCBI account) to initiate the submission process. You can initiate the submission process here: https://www.ncbi.nlm.nih.gov/geo/submitter/

Do make a note of the FTP URL and credentials (FTP username and FTP password) for later. You will need these for data transfer.

Required Data

The data required are the raw demultiplexed FASTQ files and the processed relative expression data. If you collected your data through the UMGC, then the raw data are the FASTQ files that are included in the data release from UMGC. If you processed your data through CHURP, then the `cpm_list.txt` normalized relative expression matrix and `subread_counts.txt` raw read counts matrix have the expression data that are required. Additionally, the MD5 sums for verifying file integrity are required; we will go over this in a later section.

The metadata required are given in the GEO submission template linked below (Example 1 is for ChIP-seq, and example 2 is for RNAseq):
https://www.ncbi.nlm.nih.gov/geo/info/seq.html#metadata

The required information includes:

- Project title
- Project summary
- Short description of the experimental design
- Contributors' names
- Study organism, including genotype/strain if applicable
- Sample characteristics:
  - Human-readable sample name
  - Experimental treatment or condition
  - Age (if applicable)
  - Sex (if applicable)
  - Tissue/organ (if applicable)
  - Other characteristics that are important for the experimental design
- Molecule type (e.g., messenger RNA)
- Sample rearing/handling protocol:
  - Growth conditions (if applicable)
  - Experimental treatment protocol (if applicable)
- Molecular protocol information:
  - Nucleic acid extraction protocol
  - Library preparation protocol
- Data processing protocol*

*: This tutorial covers the data processing protocol! The read trimming, read mapping, alignment filtering, expression quantification, and expression normalization steps described in this tutorial describe the steps taken to handle the data.

Calculating MD5 Sums

The MD5 sums are required for all raw and processed data that will be deposited into GEO. These are strings that can be used to verify file integrity in non-cryptographic situations, e.g., to make sure the file you are using is not unintentionally corrupted.

If you are working on a **funded collaboration** with RI Bioinformatics, we can perform the MD5 sum calculation step for you!

You can perform this calculation on MSI's compute servers. First, request an interactive compute session:

```
srun -N 1 -n 1 -c 1 --mem=2gb -t 4:00:00 -p interactive --pty bash -l
```

Once you have an interactive session, navigate to where your raw reads are stored. For this example, we will use the tutorial reads:

```
cd /home/msistaff/public/RNAseq_Tutorial/Reads
```

Then use the `md5sum` command to generate the checksums and write them to a file:

```
md5sum *.fastq.gz > ~/geo_submission_md5sums.txt
```

You can verify they were properly generated by passing the `-c` option to the `md5sum` command:

```
md5sum -c ~/geo_submission_md5sums.txt
BoneMarrow-1_S23_R1_001.fastq.gz: OK
BoneMarrow-1_S23_R2_001.fastq.gz: OK
BoneMarrow-2_S21_R1_001.fastq.gz: OK
BoneMarrow-2_S21_R2_001.fastq.gz: OK
BoneMarrow-3_S19_R1_001.fastq.gz: OK
BoneMarrow-3_S19_R2_001.fastq.gz: OK
BoneMarrow-4_S17_R1_001.fastq.gz: OK
BoneMarrow-4_S17_R2_001.fastq.gz: OK
Spleen-1_S31_R1_001.fastq.gz: OK
Spleen-1_S31_R2_001.fastq.gz: OK
Spleen-2_S29_R1_001.fastq.gz: OK
Spleen-2_S29_R2_001.fastq.gz: OK
Spleen-3_S27_R1_001.fastq.gz: OK
Spleen-3_S27_R2_001.fastq.gz: OK
Spleen-4_S25_R1_001.fastq.gz: OK
Spleen-4_S25_R2_001.fastq.gz: OK
```

Every file passes! The contents of the file are pasted below. The first column is the MD5 sum and the second column is the filename. The strings listed in the first column are what you would enter into your GEO submission sheet:

```
ee99e5e5262a0131451460845197378d  BoneMarrow-1_S23_R1_001.fastq.gz
b7ad1164d691010b397942cca8f6fc39  BoneMarrow-1_S23_R2_001.fastq.gz
614195cfc82221d93861ec265c627d29  BoneMarrow-2_S21_R1_001.fastq.gz
b49dc13ae696334525daa0b91fb1bda6  BoneMarrow-2_S21_R2_001.fastq.gz
ae2071b5c9f7e966a01e8c73779de204  BoneMarrow-3_S19_R1_001.fastq.gz
2208e126306480767354b135e9d3475b  BoneMarrow-3_S19_R2_001.fastq.gz
7c5168e84f55ba2bf6072ff2bbdb855b  BoneMarrow-4_S17_R1_001.fastq.gz
de2a0f9b67a61444cbd6d2bdd9840263  BoneMarrow-4_S17_R2_001.fastq.gz
e7900214eef5872f7797656933b32182  Spleen-1_S31_R1_001.fastq.gz
2d9fde82c0dc57fff855897eac1bc6b2  Spleen-1_S31_R2_001.fastq.gz
5d17363dc573b37c2c23c9acc592a200  Spleen-2_S29_R1_001.fastq.gz
9112d0acbb1f7eace510f00e72ddd828  Spleen-2_S29_R2_001.fastq.gz
```

```
821f8d7ed25d6e37a8a5bd0123193833  Spleen-3_S27_R1_001.fastq.gz
eebe5f6e30ab1fafd38426e20887a09e  Spleen-3_S27_R2_001.fastq.gz
b20468603ce262c38f5378e7feb580e8  Spleen-4_S25_R1_001.fastq.gz
5ba0d064a1bc09e7f38138f4fe55dec3  Spleen-4_S25_R2_001.fastq.gz
```

The process is the same for the raw read counts matrix and the normalized expression matrix.

Uploading Data to GEO

Now that you have the MD5 sums calculated and entered into your submission spreadsheet, you can upload the data files to GEO.

If you are working on a **funded collaboration** with RI Bioinformatics, we can upload the data for you, too!

If your data are larger than 1 terabyte (TB, 1000 GB) in volume, you must contact GEO and wait for a response from them before uploading the data.

Again, we can do this from the MSI compute node. First request an interactive compute session:

```
srun -N 1 -n 1 -c 1 --mem=2gb -t 4:00:00 -p interactive --pty bash -l
```

Then, navigate to the directory where your raw data are stored. This example has the tutorial dataset, but you should use your raw FASTQ folder:

```
cd /home/msistaff/public/RNAseq_Tutorial/Reads
```

We will then use an FTP client (`lftp`) to connect to the GEO upload location and put the files there. Fetch the FTP address and username and password that were provided to you by GEO. Use these to connect to the server, make a directory for your data uploads, and then upload the files. Note that there are no spaces between the `USERNAME` and `PASSWORD` in the following command, just a comma:

```
lftp -u USERNAME,PASSWORD ftp://ftp.ncbi.../uploads/
lftp ftp.ncbi.nlm.nih.gov:/uploads> mkdir my_submission
lftp ftp.ncbi.nlm.nih.gov:/uploads> cd my_submission
lftp ftp.ncbi.nlm.nih.gov:/uploads/my_submission> mput *.fastq.gz
```

Note that the `lftp` prompt shows you which server you are connected to and which directory you are in. Once these files are uploaded, you can use the `lcd` command to navigate to where your raw read counts and normalized expression matrices are stored. Upload these, too. Again, we will use the tutorial directory, but you should use your real data:

```
lftp ftp.ncbi.nlm.nih.gov:/uploads/my_submission> lcd /scratch.global/konox006/RNAseq_Tutorial/Out/Counts
lftp ftp.ncbi.nlm.nih.gov:/uploads/my_submission> put subread_counts.txt
lftp ftp.ncbi.nlm.nih.gov:/uploads/my_submission> put cpm_list.txt
lftp ftp.ncbi.nlm.nih.gov:/uploads/my_submission> bye
```

The `bye` command closes the FTP connection. Note that your terminal may print more output messages than what I have listed here. So long as you do not see errors, your command worked.

Using your favorite desktop FTP client (OSX Finder or Windows Explorer both work for this; if you are on Linux, then you can find an FTP client on your own), transfer your GEO metadata spreadsheet into the submission directory as well. You can transfer it through MSI servers via the same process above if you are more comfortable with it, but I find it more convenient to transfer from my workstation.

Notifying GEO of File Upload

Once your raw data, processed data, and metadata spreadsheet are uploaded, you can go to this link to notify GEO that your files are ready for review:
https://submit.ncbi.nlm.nih.gov/geo/submission/

If you have a **funded collaboration** with RI Bioinformatics, then we can do this step for you!

This form will require the following information:

- FTP directory where the data and metadata are deposited (in this example `/uploads/my_submission`)
- List of filenames that were uploaded
- Release date of the data (can be in the future for data to be held privately during review)
- Optional information:
  - NCBI account name for someone who should be associated with the study, e.g., the PI of the study if you are uploading on behalf of them
  - Explanations of incomplete data

Once you submit the form, wait for a response from GEO. They will contact you if any of the metadata are incomplete or if any files were corrupted during upload. Once everything checks out, they will send you a link to the study that you can include for peer review.

Glossary of Terms

This section describes the specialized terms that we use in this tutorial. I have tried to point out terms that may be used differently from how I use them in this tutorial.

Molecular Biology (Wet Bench) Terms

- **(Sequencing) Adapter**: Short nucleotide oligomers that are ligated to the 5′ and 3′ ends of a fragment in a *sequencing library*. These are non-biological sequences and are complementary to the oligomers that are present in the flow-cell for sequencing.
- **Insert**: A specific stretch of nucleic acid that retained for sequencing during library preparation. It is called an “insert” because it is “inserted” between two sequencing adapters. One characteristic of a sequencing library is its *insert size distribution* - the distribution of lengths of the insert sequences.
- **Fragment**: A molecule in the sequencing library that consists of an insert between two adapters:

`Adapter-Insert-Adapter`.

- **Library**: A collection of nucleic acid molecules that represents the sample organism that you wish to study. For RNAseq, it is usually a collection of cDNA (complementary DNA) sequences synthesized from RNA molecules.
- **Library Preparation**: The protocol that converts an extracted sample of nucleic acid into a sequencing library, read to run on the instrument. Various library preparation protocols exist, so it is important to know the details of the protocols your experiments are using. The details of the protocol do affect how your data should be handled, in particular, the sequencing adapters, strand-specificity, and ribosomal RNA depletion method must be known.
- **Sequencing Read**: A string of nucleotide bases identified by the sequencing instrument. The nucleotide bases are derived from the sequence of the inserts of the sequencing library.

Bioinformatic Terms

- **Base Quality**: A numerical value associated with each nucleotide base call generated by a sequencing instrument. The number is proportional to the probability that the base call is correct.
- **Coverage**: The degree to which a particular sequence (gene, e.g.) is represented by reads. Higher coverage means more of the sequence is represented in the data.
- **Depth**: The number of times a particular nucleotide position has been seen in sequencing reads. Higher depth means the nucleotide has been read more times and thus typically has less uncertainty. Note that some people use "coverage" or "fold coverage" to refer to this definition of "depth."
- **Gene Annotation**: (Noun) The assignment of regions of a reference genome to gene models. The assignments are stored in "annotation files" with formats such as GTF or GFF3. A gene annotation file *must only* be used with its accompanying *reference genome* file.
- **Mapping**: A process by which short sequencing reads are compared to a reference genome to identify where they originated. This is the process that is used in inferring the relative abundance of transcripts in a RNA extraction. I distinguish *mapping* from *alignment* in that alignment is used to identify homologous sequences (rather than simply identifying a genomic position) and uses different algorithms from mapping. Some people use "mapping" and "alignment" interchangeably.
- **Mapping Quality**: A numerical value that describes the confidence with which the mapping position is known. Higher values indicate higher confidence. Each mapped read (or read pair) will have a mapping quality value associated with it.
- **Normalization**: A technique for adjusting the relative gene expression values across samples to account for variation in library size.
- **(Raw) Counts Matrix**: A matrix of integers describing the assignment of mapped reads to annotated gene models across multiple samples. The raw counts matrix is the input for a differential gene expression analysis.
- **Reference Genome**: A string of nucleotides that describes the genome of a given species. Sequencing reads are *mapped* against a reference genome, and gene expression is quantified using the accompanying *gene annotation* file.