# Module 03 Part 1 – Transcriptome Assembly and Differential Expression Analysis

The genome of an organism can tell you what proteins that organism can potentially produce, but to see what's actually happening as far as gene expression at a specific life stage or under specific conditions, you need to know the RNA transcripts being produced. RNA-Seq allows you to do that. The basic steps in RNA-Seq are:

* RNA extraction
* mRNA isolation or rRNA depletion
* Reverse transcription to complementary DNA (cDNA)
* Fragmentation
* Library preparation
* Sequencing
* Quality trimming
* Assembly
* Annotation
* Abundance Estimation
* Differential expression analysis

## Required Reading

The exercises within this module are based on the Trinity tutorial, with modifications to allow you to run the pipeline on the course Linux server. To provide background information on RNASeq and Trinity, read the papers listed below. You’ll find them as downloadable files in the Blackboard module.

* *De novo* transcript sequence reconstruction from RNA-Seq: Reference generation and analysis with Trinity
* RNA-Seq: a revolutionary tool for transcriptomics

You will also need to use Trinity, blastn, and blastx command-line help for the following programs:

* Trinity --show\_full\_usage\_info
* blastn -help
* blastx -help
* java –jar /usr/local/programs/Trimmomatic-0.36/trimmomatic-0.36.jar -help

Run the help commands and review the command line options. Throughout this module remember to use tab completion rather than typing the whole command. Follow the commands outlined, and use the Trinity paper as a guide to interpret what's being accomplished at each step of the pipeline.

## Learning Objectives

* Assemble a transcriptome from RNA-Seq Illumina reads
* Identify differentially expressed isoforms and genes using the Trinity pipeline
* Characterize the predicted proteins by BLASTing against the SwissProt database
* Match differentially expressed genes to the SwissProt BLAST hits
* Query the Swiss-Prot BLAST database using blastdbcmd to obtain the SwissProt protein descriptions

## Assignment

Note: Unless you are creating or editing a file (quality trimming reads, for example), you must read the files directly from the shared locations on the server:

* /scratch/TrinityNatureProtocolTutorial
* /blastDB/S\_pombe\_refTrans

Copies of shared data in your home directory waste server resources and will negatively affect your grade on this assignment.

Part 1 consists primarily of trimming and assembling the reads, performing differential expression analysis with edgeR, and BLASTing against Swissprot. In Part 2 you'll need to draw on what you've learned in BIOL6308 to write a Perl annotation script. The script will query the Swissprot database and use hashes to merge Trinity gene IDs, SwissProt BLAST hits, and expression data produced by edgeR. All the input files are tab-separated files that you can parse using split().

### Data

You will use strand-specific RNA-Seq data from *Schizosaccharomyces pombe* in three growth stages and one stress condition:

* log growth (log)
* plateau phase (plat)
* diauxic shift (ds)
* heat shock (hs)

For each there are 1M Illumina paired-end strand-specific RNA-Seq reads, for a total of 4M paired-end reads. The reads are already available and extracted on the server in:

/scratch/TrinityNatureProtocolTutorial/1M\_READS\_sample

so you don't need to download them. They were retrieved from:

<http://sourceforge.net/projects/trinityrnaseq/files/misc/TrinityNatureProtocolTutorial.tgz/download>

and extracted with the command:

tar –xvf TrinityNatureProtocolTutorial.tgz

The raw reads are in the directory:

/scratch/TrinityNatureProtocolTutorial/1M\_READS\_sample

* Sp.ds.1M.left.fq
* Sp.ds.1M.right.fq
* Sp.hs.1M.left.fq
* Sp.hs.1M.right.fq
* Sp.log.1M.left.fq
* Sp.log.1M.right.fq
* Sp.plat.1M.left.fq
* Sp.plat.1M.right.fq

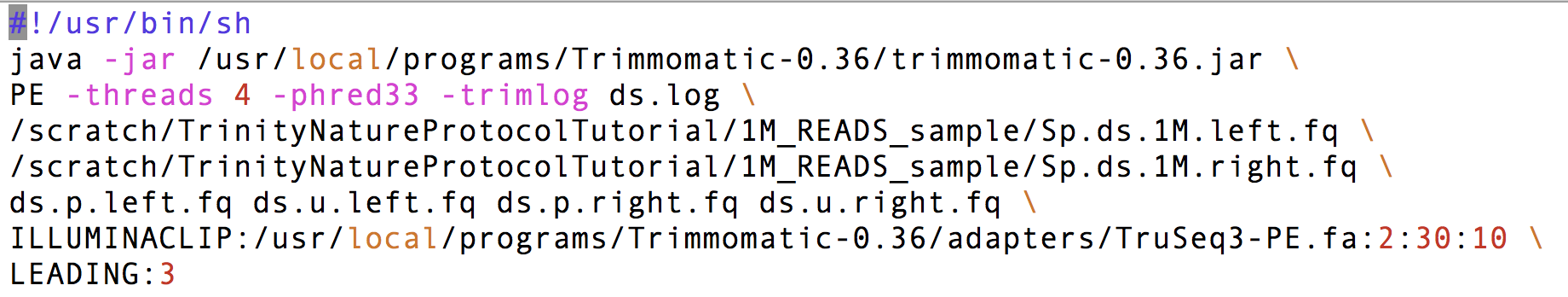
Create a ~/BINF6309/Module03 directory and use it as your working directory for all the commands that follow.

### Quality Trimming

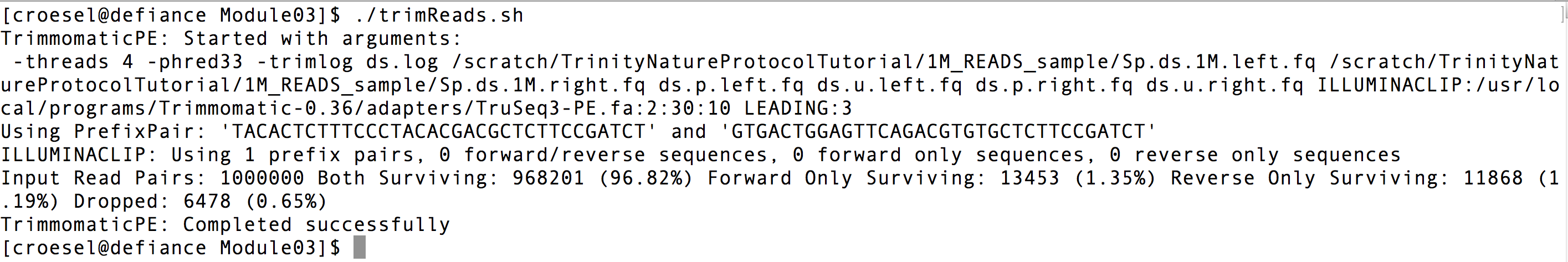
First you need to quality trim all these reads. From reading the Trimmomatic manual

<http://www.usadellab.org/cms/?page=trimmomatic>

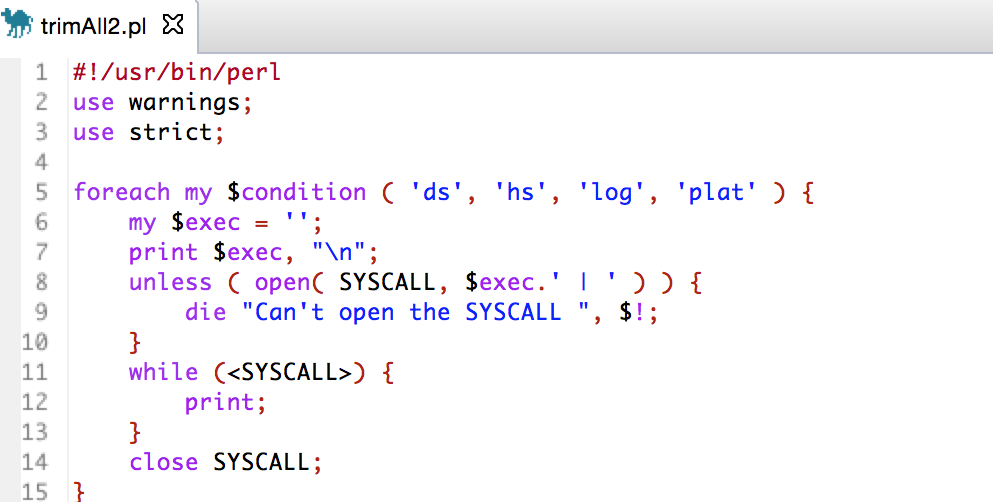
you'll find that the command for the ds reads would be as shown in the shell script below. Create this shell script and run it. Pay close attention to the backslashes (\) in the script, which mean the command continues on the next line. Every space is important.



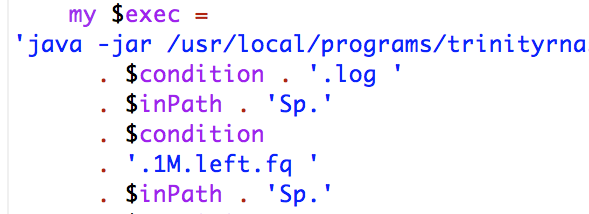
When it runs successfully you should see something like this:



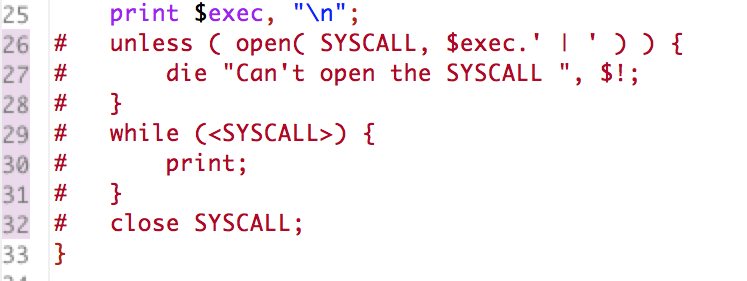
That's a long command, and you have four data sets to trim. For a big RNA-Seq project you may have a dozen or more. For each of these only a small part of the command needs to change. You can automate this with a Perl script. Later you may also want to trim with different parameters, and once you have the commands built into a Perl script, you can change parameters and easily re-trim dozens of fastq files. Using the starter script below, write the Perl script named trimAll2.pl by typing the command for trimming the ds files between the single quotes on line 6, then replacing ds with $condition.



Enclose the portions that don’t vary in single quotes, then use the concatenate (.) operator to concatenate the $condition variable wherever necessary. The example below shows concatenation, but you need to figure out the exact text for your trim command from the ds example in the shell script.



Make sure you test the formatting of your commands by commenting out the SYSCALL code and just printing the formatted commands first.



Make sure the printed commands are correct before un-commenting and actually running with SYSCALL.

Since Eclipse highlights errors in code as you type, I recommend doing this in Eclipse so you can see when you’ve successfully replaced all occurrences of ds with the variable $condition.

Write a shell script named concatReads.sh that uses the cat command to concatenate the RNA-Seq data across all samples into a single set of inputs for a single reference Trinity assembly. Combine all left reads into a single file, and combine all right reads into a single file. If you need to refresh your memory on cat, run the command:

man cat command

You can use:

\*.paired.left.fq

for cat to read all the left files, and

\*.paired.right.fq

for cat to read all the right files. Use

>

to redirect the cat output to trimmed.left.fq and trimmed.right.fq.

### Check Trimming Results

It's important to have exactly the same number of reads in the left and right files to preserve the pairings. Check this with:

wc -l trimmed.left.fq

wc -l trimmed.right.fq

The commands return the line counts for each file, and you should get exactly the same line count for each file.

### Assembly

Now, assemble the reads into transcripts using Trinity. To find the command syntax, run Trinity without any parameters:

Trinity

A shortcut to go from program help to a shell script with the parameters you need is to redirect help to a shell script. It helps to have everything in one place while editing the shell script.

Trinity &> runTrinity.sh

You should set the parameters to:

* 20G max\_memory
* Library type strand-specific reverse-forward (RF)
* 4 CPUs
* Type of reads, fq

You should also precede the command with:

nice -n 19

This ensures that a CPU-intensive process doesn't starve the server of the resources it needs to handle essential system functions like ssh.

Redirect standard output to trinity.log, redirect standard error to trinity.err, and run the assembly in the background.

Run the command:

top -u yourUserName

to monitor the progress of the assembly. You'll see the pipeline run jellyfish, bowtie, and samtools. You can logout while the assembly is running since you started it in the background. Login after an hour or two to see if the assembly has finished.

### Checking the Assembly

The Trinity output will be FASTA-formatted sequences in the file

trinity\_out\_dir/Trinity.fasta

TrinityStats.pl will report the number of transcripts, components, and the transcript contig N50 value based on the Trinity.fasta file. The N50 value is useful for confirming that the assembly succeeded. It should be near the average transcript length of S. pombe, which is 1397 bases. TrinityStats.pl is not in your path, so you'll need to give the absolute path. It's located in

/usr/local/programs/trinityrnaseq-2.2.0/util

Based on knowing the names and location of the files you should be able to determine how to run TrinityStats.pl against your Trinity.fasta file from your Module03 directory. Run TrinityStats.pl without parameters to see the usage, and find the N50. Write a shell script named checkStats.sh to run this command. The shell script should be created in your Module03 directory.

### Compare the transcriptome to a reference data set using blastn

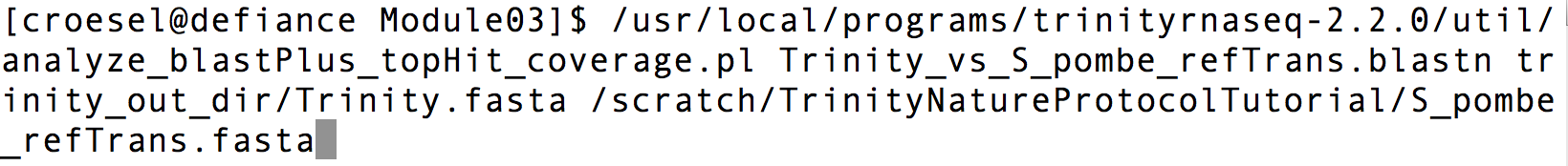
One way to evaluate the quality of a transcriptome assembly is to find the number of assembled transcripts that are full-length or nearly full-length. You can do this by aligning the transcriptome to a known reference. The reference transcriptome of Schizosaccharomyces pombe is available on the server as BLAST database  /blastDB/S\_pombe\_refTrans. Write a shell script named runBlast.sh that will use megablast to align the known transcripts to the Trinity assembly. Use the following parameters for blastn:

* -query trinity\_out\_dir/Trinity.fasta
* -evalue 1e-20
* -db S\_pombe\_refTrans
* -dust no
* -task megablast
* -num\_threads 2
* -out Trinity\_vs\_S\_pombe\_refTrans.blastn
* -max\_target\_seqs 1
* -outfmt 6

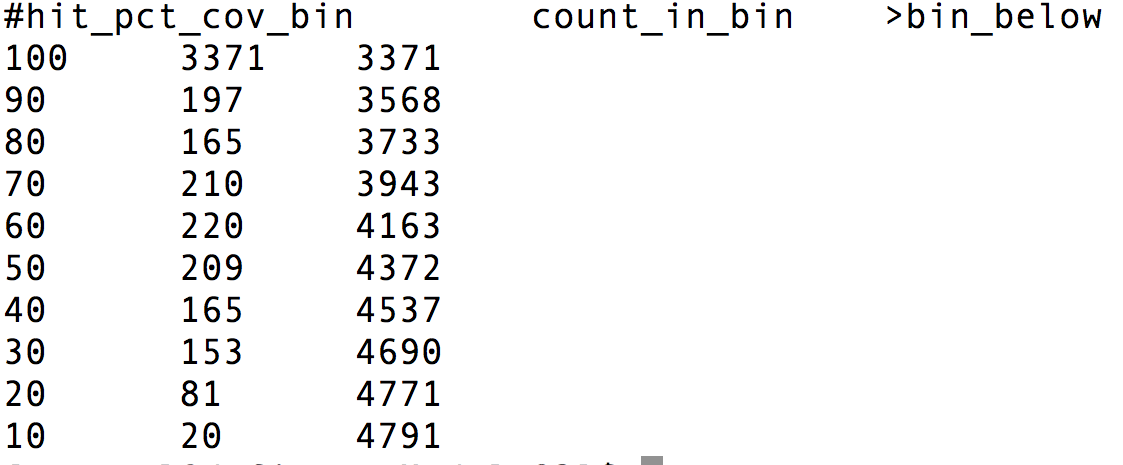
You can reduce typing by redirecting blastn help to create the shell script. Include this at the end of your script to run in the background and redirect output:

1>blastOut.txt 2>blastErr.txt &

Once BLAST is finished, write a shell script named checkCov.sh to run the command below and examine the length coverage of top database hits:



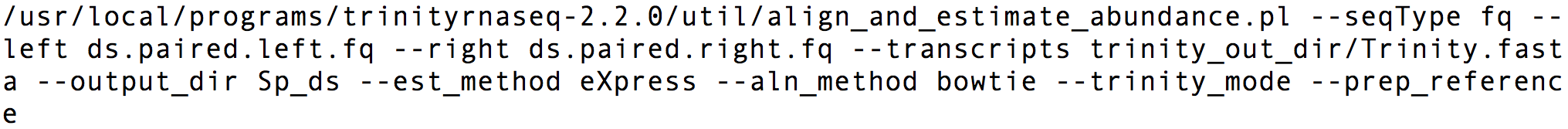
You should see output like this:



This table shows that there are 197 proteins that each match a Trinity transcript between 80% and 90% of their protein lengths. There are 3568 proteins that are represented by nearly full-length transcripts, having >80% alignment coverage. There are 3371 proteins that are covered by more than 90% of their protein lengths.

### Abundance Estimation Using eXpress

Transcript abundance estimates are obtained by running eXpress separately for each sample. The PERL script align\_and\_estimate\_abundance.pl provides an interface to the eXpress software, translating the Trinity command-line parameters to their eXpress equivalents and then executing the eXpress software. This script still has to been run for each sample though, so using the same approach as you did with Trimmomatic, automate this to run eXpress for all samples in a single script using a Perl script named eXpressAll.pl. The command for the ds sample is shown below. Note that the command below is all one line. It wraps in this document to look like two.



Each step generates the files

/${output\_dir}/results.xprs

/${output\_dir}/results.xprs.genes

containing the abundance estimations for Trinity transcripts and components. The ${output\_dir} in the filename is set based on the --output\_dir setting in the above commands, which is unique to each sample. Note, the genes and transcripts can be examined separately using their corresponding abundance estimates in the differential expression analysis guide below.

### Differential Expression Analysis Using edgeR

To combine the results from the four directories into two matrix files (one for isoforms, one for genes) run the commands:

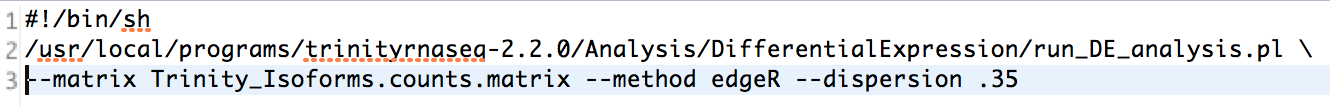
/usr/local/programs/trinityrnaseq-2.2.0/util/abundance\_estimates\_to\_matrix.pl --est\_method eXpress --name\_sample\_by\_basedir --out\_prefix Trinity\_Isoforms Sp\_ds/results.xprs Sp\_hs/results.xprs Sp\_log/results.xprs Sp\_plat/results.xprs

/usr/local/programs/trinityrnaseq-2.2.0/util/abundance\_estimates\_to\_matrix.pl --est\_method eXpress --name\_sample\_by\_basedir Sp\_ds/results.xprs Sp\_hs/results.xprs Sp\_log/results.xprs Sp\_plat/results.xprs

Later, you’ll need the transcript length information, which you can extract from one of the eXpress files with the command:

cat Sp\_ds/results.xprs | cut –f2,3,4 > trans\_lengths.txt

Now, write a shell script named runDE.sh to run edgeR via the helper script provided in the Trinity distribution:



Run the command:

ls -ltrh

edgeR creates a new directory for each run, so find your edgeR output directory using ls.

ls edgeR\*

The files \*.DE\_results contain the output from running EdgeR to identify differentially expressed transcripts in each of the pairwise sample comparisons. Examine the format of one of the files, such as the results from comparing Sp\_log to Sp\_plat:

head edgeR\*/Trinity\_Isoforms.counts.matrix.Sp\_log\_vs\_Sp\_plat.edgeR.DE\_results

rsync the edgeR directory to your computer to view the PDF files. Assuming your local directory structure mirrors your remote directory structure and your user ID is white.w, the command would be:

rsync -rutv white.w@defiance.neu.edu:~/BINF6309/Module03/edgeR\* ~/BINF6309/Module03

### Annotation

Now you have tables of differentially expressed transcripts between each of the conditions in your edgeR directory. To make sense of these you need to figure out what these proteins are. You can do that by using blastx to align the transcripts to known proteins in the Swissprot BLAST database.

Write a shell script named blastTranscripts.sh to BLAST the transcripts against SwissProt:

blastx -query Trinity.fasta.transdecoder.pep -db swissprot -num\_threads 8 -max\_target\_seqs 1 -outfmt 6 1> blastp.outfmt6 2>blast.err &

Familiarize yourself with the blastp output file blastp.outfmt6 and the results files in your edgeR directory. Next week in part 2 you’ll write a script to merge these files into a single annotated differential expression report.