Guided Study Final Report
Michael Dhar
Guided Study: Spring-Summer 2015
NYU-Poly Bioinformatics MS Program

SUMMARY:
This report covers the Guided Study project completed by Michael Dhar in the Spring/Summer semesters of 2015 at NYU Poly. I completed the guided/independent study, working with Prof. Jonathan Flowers (NYU) as Mentor and Prof. Mgavi Brathwaite (NYU Poly) as supervisor. The project was to create a pipeline for differential expression analysis of RNAseq data, for use in Prof. Flowers' lab at NYU, where the genome of the datepalm (Phoenix dactylifera) is being studied. The code for this project was written and tested on the NYU HPC Mercer cluster, via SSH from my home computer. The final, working scripts are attached below in appendices. Together, they comprise a one-button, modular pipeline for the differential expression analysis of paired-end short-read RNAseq data stored in FastQ files. The pipeline conducts validation of files, alignment of reads to a reference genome, counts of read expression, and differential expression analysis. There is also a Preprocessing pipeline that validates and quality checks the FastQ files.

INTRODUCTION:

The Main pipeline resulting from this guided study project is a single-button analysis pipeline that takes in a list of paired-end RNAseq-read samples (FastQ files) and applies a series of analyses and validations resulting in differential expression analysis. The pipeline is set up to handle a basic experimental design: treatment vs. control. There is an additional pre-processing pipeline that validates the (FastQ) files in the sample list. Both pipelines are written to run on the Mercer cluster of the NYU high performance computing center. The pipeline is designed to be one-button tool that can be modular and applied to other genomic data. Thus, PBS scheduling scripts for the HPC cluster, where possible, are written to be independent of the specific tools called for the analysis steps. Thus, for example, an alignment tool other than the default Tophat tool used here could be slotted in. The pipeline could also be used for RNA-seq data other than that produced by the datepalm study.

Testing for both pipelines was done using a set of small FASTQ files from an experiment on japonica rice (see Materials and Methods).

MAIN PIPELINE: In more detail, the main pipeline operates as follows:

INPUT FILES: The pipeline requires two input files in order to run: a Control file and a Configuration file. The Control file (see example "cntrlfile.txt" below) is the list of samples (paired-end FastQ files) to be analyzed. It is supplied by the user of the pipeline. It must be tab-delimited, containing one line for each sample. Each line contains the sample ID, the path to the forward FastQ file, the path to the reverse FastQ file, and the experimental condition (e.g., treatment vs. control). The Configuration file contains links to genomic files needed to run the pipeline (see "Config.txt" below, with data from the rice samples filled in). There are a number of slots of information that the user of the pipeline will fill in, namely: REF (the path to the reference genome), FASTA_IND (the path to the fai index file), GFF (the path to the GFF3 genomic feature file), GENOME_BASE (the base name for the genome file) and SAMPLE_TABLE (the control file mentioned previously). The user must also ensure that the proper R libraries are downloaded to the computational environment; these consist of Bioconductor, easyRNASeq and DESEq (see Materials and Methods for details).

PREP SCRIPTS: To then run the pipeline, the user must simply execute the BASH master script ("RNAseq_master.sh") by typing into the terminal: "bash RNAseq_master.sh". This script then executes, using the Control and Configuration files, the entirety of the main analysis pipeline, as follows: The master script first runs a Perl validator script ("Validator.pl") on the control file to check that it is formatted correctly (four columns, tab delimited, no empty lines, proper end-of-line characters, sample files exist, and sample and id names contain the proper characters). An error found by this Perl script is reported to the user, and the Perl script dies, which kills the overall pipeline. If the validation tests are passed, the master script moves on to the next step: calling "DirStruct.sh", a BASH script that creates the directory structure for the pipeline. Within the overall "project" folder, this script creates an "alignments.1" directory, with sub-directories for each sample and a "bam_links" directory to store soft-links to the BAM files coming from the alignment step (see below).

ALIGNMENT: The master script then runs the Bowtie program to create Bowtie Indices for the sample files (this is necessary for the Tophat alignment step to run). Bowtie indices are created by calling a PBS script ("BowtieIndex.pbs"), which schedules the Bash script ("BowtieIndex.sh") that actually executes the Bowtie program, using the reference genome and genome base name from the Configuration file. (This is done, as explained above, to separate the scheduling from the tool execution, in order to make the overall script modular; i.e., an alternative to Bowtie could be slotted in.) The output Bowtie indices are stored within the working "project" directory.

The status of the Bowtie PBS job is stored by the master script in a variable ("INDEX"), which is used to create a dependency. That is, the next step ("ALIGNMENT") does not run until INDEX has completed (with the system returning an "ok" status to the INDEX variable). The alignment step is then run, using the Tophat program. This, again, is done by launching a PBS scheduler ("Tophat.pbs") that in turn calls the Tophat Bash script ("Tophat.sh"). Here, however, the PBS is called as an array job, with the number of the array set as the number of samples in the control file (this number is fed into the PBS script by the master script). The PBS script reads through the control file, and assigns each sample to a separate array ID, feeding this to an iteration of the Tophat.sh script. In this way, the PBS and BASH scripts together loop through the control file and run Tophat on each sample. The resulting alignments (BAM files) are stored in the appropriate "Sample" directories within "alignments.1". The Tophat.sh script also uses Samtools to create .bai indices (needed for downstream analysis) for each BAM file; the indices are stored in the same directories.

The status of the PBS Tophat job is stored in an ALIGNMENT variable, and a dependency is again set up for the following step: SOFTBAM. This step creates softlinks for every BAM file contained within the directory "bam_links". (This is done so that the subsequent step, the count step, can produce a count table with unique names for each BAM file, as Tophat saves all BAMS as "accepted_hits.bam").

COUNT: Again, set up as a dependency, the next step "COUNT," runs a count on the BAM files, to check expression levels in the samples. The tool used for this is easyRNASeq, a package of the Bioconductor suite of RNA-seq analysis tools written in the R statistical programming language. Specifically, the subprogram/updated method of easyRNASeq called "simpleRNASeq" is run. This is done, similarly to previous steps, by calling a PBS scheduler ("simpleRNASeq.pbs") that then calls an R script ("simpleRNASeq.R") that runs simpleRNASeq on all the BAM files. The PBS script reads through the control file and derives the BAM soft-link file names for each sample, feeding these to the R script. The simpleRNASeq tool then finds expression levels of a feature (here, genes) in each sample. The output is an R object known as a "summarized experiment." Using the "assay" accessor function and the "write.table" function in R, a count table of feature (gene) expression levels is written to a text file: "count.table.txt". This file is used by the subsequent, differential expression step.

DIFFERENTIAL EXPRESSION: This final step of the pipeline (DIFFEXP) is also set up as a dependency, to run after the count step (COUNT) has completed. Here, the R/Bioconductor tool DESeq is used. Similar to previous steps, a scheduler ("DESeq.pbs") is called. This PBS script first calls a Perl script ("ConditionWriter.pl"). This script creates a table listing the headers of the count table (BAM filenames that are used to name each sample in the count table) with their appropriate conditions (treatment vs. control). This is done by reading through the count table headers, searching the original control file for the matching IDs, and then pairing those headers with the appropriate conditions as listed in the control file. (This is done to ensure that a header is not matched to the wrong condition. That is, if a later BAM file was completed before an earlier one due to parallelization, it would not be accurate to simply list the count table headers in order followed by the conditions in the order found in the control file.)

The pairs of ID-Condition are then fed to "DESeq.R", an R script that runs DESeq on the count table, comparing expression levels in treatment vs. control samples. This output is dumped into a file,"DESeq.output.txt". The information can be used to plot differential expression within R/DESeq and for other downstream analysis. As part of the DESeq.R script here, two basic plots are created and saved to a PDF file ("deseq_plots.pdf"): 1) the log2 fold changes against mean normalized counts and 2) a histogram of p values.

This is the final output for the main pipeline.

****

PREPROCESSING PIPELINE: At the users discretion, another pipeline may be run before the main RNASeq pipeline. This preprocessing pipeline validates the FASTQ files listed in the control file.

It operates as follows: This pipeline also takes information from the Configuration and Control files. The user must copy the files for the preprocessing pipeline into the same directory containing the Configuration and Control files. This pipeline is executed by launching the preprocessing master script ("PreProc_master.sh") by inputting to the terminal "bash PreProc_master.sh". This script reads in the SAMPLE_TABLE from the Configuration file, and retrieves the sample number from the control file. The master script then launches all validation scripts, beginning with "LengthFilter.pbs". This simple BASH PBS script loops through the FASTQ files and checks that all forward FASTQs are the same length as their reverse partners, and if any of the reads have been filtered by the Illumina QC filter. Any mismatched lengths or QC-filtered results are reported to a log file, "PreProcLog.txt".

The master script then calls an array job "FastQC.pbs", a scheduler that (similar to the Tophat step above) feeds the FastQ files to a BASH script ("FastQC.sh"). This BASH script runs the FastQC quality-check program on each FastQ. (As an array, this is done by pairing each sample to a corresponding array ID.) The FastQC results are stored in a directory corresponding to each FASTQ file (e.g. "FASTQ_R1_fastqc"). Finally, the master script calls the script "ReadID.pbs." This script loops through each read ID in each forward FASTQ and compares that read ID to the corresponding ID in the reverse FASTQ. If any do not match up, this is reported to the log file. This completes the preprocessing pipeline.

(Note: The Preprocessing Pipeline was finished the final week of the semester, as an additional optional component. It has not yet been reviewed by the Mentor, and the ReadID.pbs step is currently slow. A different algorithm may be desired.)

****

MATERIALS AND METHODS:

Work on the above pipelines was done using several tools and resources:

HARDWARE:
Programming work was done on my, the student's, home laptop computer, a MacBook Pro. The computer was connected remotely, via Secure SHell (SSH) to the NYU high-performance computing (HPC) center, on the Mercer cluster (https://wikis.nyu.edu/display/NYUHPC/Clusters+-+Mercer). Files and scripts were stored, and the pipeline scripts were tested, on that HPC system.

SCRIPTING LANGUAGES:
Several scripting languages were used: BASH scripting was used to program the pipeline's backbone (master scripts) and major components (PBS and BASH scripts to run individual tools). Perl (v 5.18.2) scripting was used at several points to accomplish specific tasks (e.g., to conduct validation of the control file and to write the condition table for use in the differential expression analysis). The R statistical programming language (R v 3.2.0) was used in the context of the Bioconductor RNA-seq R packages, specifically easyRNASeq for producing count tables and DESeq for conducting differential expression analysis (see below for Biconductor package details).

PROGRAMMING ENVIRONMENT:
Programming was done in at least one of three programming environments: All of Perl, BASH and R were coded using simple text files and the vim text editor in the NYU HPC's linux BASH environment, with scripts run from the command line. Perl coding was also done within the Eclipse integrated development environment. On the student's home computer, some scripting was also done using Xcode (Mac-based programming environment) for BASH and Perl scripts.

ANALYSIS TOOLS:

The following analysis and QC tools are called within the Main or PreProcessing pipelines:

* BowTie v 2.2.3 (http://bowtie-bio.sourceforge.net/index.shtml) is a short-read aligner and the first step within the Tophat aligner (see below), but is also called separately here in order to produce Bowtie indices, which are necessary for the Tophat run.

* Tophat v 2.0.12 (https://ccb.jhu.edu/software/tophat/index.shtml) is a splice-junction mapper that aligns RNA-seq reads to reference genomes. It runs the Bowtie short-read aligner and then performs splice-junction analysis on Bowtie results.

* Samtools v 0.1.19 (http://www.htslib.org/doc/samtools.html) is a set of tools for manipulating SAM and BAM alignment files. It also performs indexing of BAM files (its function in this pipeline) and other tasks.

* FastQC v 0.11.2 (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/) performs a quality check on sequence data, here performed on FASTQ files provided to the pipeline(s). The tool returns reports on base-sequence quality, GC content, sequence overrepresentation, and other measures.

BIOCONDUCTOR PACKAGES:
Within the R statistical programming environment, some of the suite of RNA-seq analysis packages within Bioconductor (v 31.: http://bioconductor.org/packages/release/data/experiment/html/RnaSeqTutorial.html) were employed in this pipeline, namely:

* easyRNASeq v 2.4.7 (http://www.bioconductor.org/packages/release/bioc/vignettes/easyRNASeq/inst/doc/easyRNASeq.pdf): This package contains tools for processing RNA-seq data, including producing a count table of a genic feature (here, genes). This is the function of interest to this pipeline, and the most up-to-date easyRNASeq method for that purpose was used here: simpleRANSeq. That tool produces a summarized experiment object containing information about the sequence data, most importantly a count table of the expression of a feature (genes).

DESeq v. 1.20.0 (http://bioconductor.org/packages/release/bioc/vignettes/DESeq/inst/doc/DESeq.pdf): This package's methods test for differential expression using various statistical techniques (negative binomial distribution and a shrinkage estimator). It applies these methods to a count table of feature expression (here, reads to a gene).

TEST DATA:
During development and testing, both the Main and PreProcessing pipeline scripts were executed on data from a paired-end (2x50) RNA-seq temperature-stress experiment on japonica rice (Azucena). The samples all come from chromosome 11 of the rice genome. There are two controls (maintained at 30 degrees Celsius for 3 hours) and two treatment samples (transferred to 40 degrees Celsius for 3 hours), as follows:

3 hrs at 30 C (controls)
ID012
ID156

3 hrs at 40 C (treatment)
ID371
ID803

The Control and Configuration file examples attached here (see Appendices A and B below) contain the paths to the relevant FASTQ and genome files used in this test experiment.

****

FLOWCHARTS:

Please see attached, graphical flow charts for Main Pipeline (Appendix W) and PreProcessing Pipeline (Appendix X).

****

FINAL OUTPUT:

Please see attached two plotting outputs for the test run of the Main Pipeline on the TEST DATA listed in Materials and Methods: 1) log2 fold changes against mean normalized counts (Appendix Y) and 2) a histogram of p values.
(Appendix Z).

****

REFERENCES/BIBLIOGRAPHY:

Here are the respective publications for the tools used in this pipeline.

Anders S, Huber W. 2010. Differential expression analysis for sequence count data. Genome Biology11:R106. [Online]. Available: http://www.genomebiology.com/content/pdf/gb-2010-11-10-r106.pdf

Andrews S. Last modified March 25, 2015. FastQC: A quality control tool for high throughput sequence data. Babraham Bioinformatics. [Online]. Available: http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/ [Babraham Bioinformatics, the maker of FastQC, does not publish in academic journals, but this is the documentation for their FastQC tool.]

Delhomme N, Ismael P, Furlong EE, et al. Oct. 1, 2012. easyRNASeq: a Bioconductor package for processing RNA-Seq data. Bioinformatics 28(19):2532-2533. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3463124/

Huber W, Carey VJ, Gentleman R, et al. Jan. 29, 2015. Orchestrating high-throughput genomic analysis with Bioconductor. Nature Methods 12:115-121. [Online]. Available: http://www.nature.com/nmeth/journal/v12/n2/abs/nmeth.3252.html

Langmead B, Trapnell C, Pop Mihai, et al. March 4, 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biology 10:R25. [Online]. Available: http://www.genomebiology.com/content/pdf/gb-2009-10-3-r25.pdf

Li H, Handsaker B, Wysoker A, et al. Aug. 15, 2009. The Sequence Alignment/Map format and SAMtools. Bioinformatics 25(16): 2078-2079. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2723002/

Trapnell C, Roberts A, Goff L, et al. March 1, 2012. Corrected online Aug. 7 2014. Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks. Nature Protocols 7:562-578. [Online]. Available: http://www.nature.com/nprot/journal/v7/n3/full/nprot.2012.016.html

****
CODE (APPENDICES):

Appendix A: "contrlfile.txt": This is an example control (sample) file, used in the testing of this pipeline

```
ID012   /scratch/courses/BI7653/project.RNAseq/ID012_R1.chr11.fastq.gz  /scratch/courses/BI7653/project.RNAseq/
ID012_R2.chr11.fastq.gz  control
ID156   /scratch/courses/BI7653/project.RNAseq/ID156_R1.chr11.fastq.gz  /scratch/courses/BI7653/project.RNAseq/
ID156_R2.chr11.fastq.gz  control
ID371   /scratch/courses/BI7653/project.RNAseq/ID371_R1.chr11.fastq.gz  /scratch/courses/BI7653/project.RNAseq/
ID371_R2.chr11.fastq.gz  treatment
ID803   /scratch/courses/BI7653/project.RNAseq/ID803_R1.chr11.fastq.gz  /scratch/courses/BI7653/project.RNAseq/
ID803_R2.chr11.fastq.gz  treatment
```

****

Appendix B: "Config.txt": This is an example configuration file, used in the testing of this pipeline

```
#This is the configuration file for the RNA seq pipeline.
#It contains variables for the reference genome (REF), the fasta index
#(FASTA_IND), the GFF file (GFF), the base of the genome's ID (GENOME_BASE),
#and the control file (SAMPLE_TABLE). Please assign values to these variables
#using the following syntax: VARIABLE=value.

REF=/scratch/courses/BI7653/project.RNAseq/rice7_Chr11.fa
FASTA_IND=/scratch/courses/BI7653/project.RNAseq/rice7_Chr11.fa.fai
GFF=rice7_Chr11.gff3
GENOME_BASE=rice7_Chr11

SAMPLE_TABLE=cntrlfile.txt
```

****

Appendix C: "RNAseq_master.sh": Master script, updated with incorporated simpleRNASeq call (final step)

```
#!/bin/bash

#This is the master shell script for the
# RNA-seq pipeline. It is configured for the
# Mercer cluster.

#Usage: (1) read the README file accompanying the
#           pipeline for details
#       (2) copy this script and all pipeline scripts
#           to a project directory on /scratch
#       (3) execute this script from a login node
#           using: bash RNAseq_master.sh
```

```
#set configuration file
CONFIG=Config.txt

#check that configuration file exists
if [ ! -f ${CONFIG} ]; then
        echo _ERROR_ configuration file not found
        exit 1
fi

#define configuration variables
source ${CONFIG}


#validate sample input table
# *** SAMPLE_TABLE variable defined in configuration file ***
perl Validator.pl ${SAMPLE_TABLE}
#Check if Validator failed--if so, end RNAseq_master script
val_return=$?
if [ $val_return -ne 0 ]; then
        echo "Validator test of Sample File failed, terminating RNAseq program"
        exit 1
fi

#create directory structure for project
bash DirStruct.sh

#get number of samples
# *** SAMPLE_TABLE variable defined in configuration file ***
NUMSAMPLES=$(wc -l < ${SAMPLE_TABLE})

#Make bowtie index
#Complete bowtie index step before running alignments
INDEX=`qsub BowtieIndex.pbs`

#Align the processed fastqs
# *** runs as an array job after completion of bowtie ***
ALIGN=`qsub -W depend=afterok:${INDEX} -t 1-${NUMSAMPLES} Tophat.pbs`

#Run bam softlink script on tophat-aligned BAM files
# *** runs as array job after completion of alignments ***
SOFTBAM=`qsub -W depend=afterokarray:${ALIGN} SoftBam.pbs`

#Run simpleRNASeq on tophat-aligned BAM files
# *** runs after completion of bam soft-link step ***
COUNT=`qsub -W depend=afterok:${SOFTBAM} simpleRNASeq.pbs`

#Run DESeq differential expression analysis on simpleRNASeq count table
# *** runs after completion of simpleRNASeq step ***
DIFFEXP=`qsub -W depend=afterok:${COUNT} DESeq.pbs`
```

****

Appendix D: "Validator.pl": This Perl script checks the formatting of the control (sample) file

```
#!/usr/bin/perl -w
#use warnings;
#use strict;

#This Validator checks the format of the control (sample) file for the RNAseq pipeline

print "This is the validator script\n";

#get control file and open for reading
my $file = shift @ARGV;
open (IN, $file) or die "Can't open file:$!";


#initialize variables for row number and row length (column number)
my $row_no = 1;
my $row_length =0;

#read in control file (sample table) line by line for validation
while (<IN>)    {
        #next two if statements check for bad end-of-line characters (\r\n and \r)
        if (/\r\n$/)    {
                die "Row $row_no uses the \\r\\n end-of-line character. Use only Unix-style \\n.\n";
        }
        if (/\r$/)      {
                die "Row $row_no uses the \\r end-of-line character. Use only Unix-style \\n.\n";
        }

        chomp; #remove trailing line character (AFTER checking for bad EOL chars)

        #check if line is empty
        if (/^\s*$/)    {
                die "Row $row_no is empty.\n";
                next;
        }

        #read row into array, split on tab
        my @row = split(/\t/,$_);
        $row_length = scalar @row;
                #check if row has exactly 4 columns
                if (@row != 4) {
                        die "Row $row_no has $row_length columns. All rows must have 4 columns.\n";
                        next;
                }
                #check if either sample in row does not exist
                if (!-e $row[1]) {
                        die "Sample $row[1] doesn't exist.\n";
                }
                if (!-e $row[2]) {
                        die "Sample $row[2] doesn't exist.\n";
                }

                # Check if Sample ID contains non-alphanumeric characters (other than underscore)
                if ($row[0] =~ /[^\w.]/) {
                        die "Sample ID name $row[0] contains non-alphanumeric characters.\nUse only alphanumeric chars,
underscore and period in Sample ID.\n";
                }
                # Next two if statements check if Sample paths contain non-alphanumeric, other than period, underscore,
forward-slash
                if ($row[1] =~ /[^\w.\/]/) {
                        die "Sample $row[1] contains non-alphanumeric characters.\nUse only alphanumeric chars, underscore,
period and forward-slash in Sample path.\n";
                }
                if ($row[2] =~ /[^\w.\/]/) {
                        die "Sample $row[2] contains nonalpha-numeric characters.\nUse only alphanumeric chars, underscore,
period and forward-slash in Sample path.\n";
                }
        $row_no++; #increment row number
        }
close (IN);
```

****

Appendix E: "DirStruct.sh": This BASH script makes the file directories for the pipeline

```
#/bin/bash

# This script creates the directory structure for the RNAseq pipeline, within the user-created 'project' directory

# source Config file to get sample table
source Config.txt

#check if 'alignments' directory exits for alignment step; create if not
if [ -d "alignments.1" ]
then
        echo "Can't make directory alignments.1; directory already exists."
else
        mkdir alignments.1
fi

#check if 'bam_links' subdirectory exits for alignment step; create if not
if [ -d "alignments.1/bam_links" ]
then
        echo "Can't make directory bam_links; directory already exists."
```

```
else
        mkdir alignments.1/bam_links
fi


# read tab-delimited Sample Table to get sample IDs
IFS=$'\t'
while read ID IN1 IN2 CD; do #check if Sample folders exist; create if not
        if [ -d "alignments.1/Sample_${ID}" ]
        then
                echo "Can't make directory Sample_${ID}; directory already exists."
        else
                mkdir "alignments.1/Sample_${ID}"
        fi
done <"${SAMPLE_TABLE}"
```

****

Appendix F: "BowtieIndex.pbs": This schedules call to Bowtie Index creator script

```
#!/bin/bash

#PBS -l nodes=1:ppn=4,walltime=4:00:00,mem=4gb
#PBS -N BowInd
#PBS -M mid224@nyu.edu
#PBS -m a
#PBS -j eo

cd ${PBS_O_WORKDIR}

#This script calls the Bowtie Index creator

module load bowtie2/intel/2.2.3

bash BowtieIndex.sh > BowtieIndex.${PBS_JOBID}.log 2>&1
```

****

Appendix G: "BowtieIndex.sh": Launches bowtie command to make bowtie indices

```
#/bin/bash

#This script creates the Bowtie index

#Get reference genome (REF) and genome base name (GENOME_BASE) from Config file
source Config.txt

# Make bowtie index
bowtie2-build ${REF} ${GENOME_BASE}
```

****

Appendix H: "Tophat.pbs": Array-job scheduler for the alignment (Tophat) step

```
#!/bin/bash

#PBS -l nodes=1:ppn=8,walltime=24:00:00,mem=8gb
#PBS -N Tophat
#PBS -M mid224@nyu.edu
#PBS -m a
#PBS -j eo


cd ${PBS_O_WORKDIR}

# source config file to get SAMPLE_TABLE path
source Config.txt

# Throw error if pbs script
if [ -z ${PBS_ARRAYID} ]; then
        echo _ERROR_ expecting an array job
        exit 1
fi
```

```
#copy reference genome to working directory
cp ${REF} ${PWD}

# Get fastqs paths from sample table for current subjob using ${PBS_ARRAYID}
IFS=$'\t'
line=$(sed -n "${PBS_ARRAYID}p" "${SAMPLE_TABLE}")
read -r ID FQ1 FQ2 CD <<< "$line"       #assign to variable for Sample ID, forward and reverse FastQ, and Condition

# Load modules
module load bowtie2/intel/2.2.3
module load tophat/intel/2.0.12
module load samtools/intel/0.1.19       #for index creation

#Call Tophat bash script on Sample for current array ID, pass in sample paths, ID
bash Tophat.sh ${FQ1} ${FQ2} ${ID} > tophat.${PBS_ARRAYID}.${PBS_JOBID}.log 2>&1
```

****

Appendix I: "Tophat.sh": This BASH script executes Tophat alignment on a given forward and reverse sample; also creates .bai index

```
#!/bin/bash

# This script runs tophat align on a forward and reverse FQ, which have not been trimmed

# Sample assignment to variables; untrimmed FastQ Files
FQ1=$1
FQ2=$2
ID=$3

if [ -z ${FQ1} ]; then
        echo _ERROR_ usage error
fi

if [ -z ${FQ2} ]; then
        echo _ERROR_ usage error
fi

# source CONGIF file
source Config.txt

# Load modules
#module load bowtie2/intel/2.2.3
#module load tophat/intel/2.0.12
#module load samtools/intel/0.1.19      #for index creation

# Tophat alignment step
tophat --no-coverage-search --num-threads 8 -G ${GFF} -o alignments.1/Sample_${ID} --no-novel-juncs ${GENOME_BASE} ${FQ1} $
{FQ2}

# Create bai index
samtools index alignments.1/Sample_${ID}/accepted_hits.bam
```

****

Appendix J: "SoftBam.pbs": Creates soft links to BAM and BAI files in bam_links directory for downstream use

```
#!/bin/bash
#
#PBS -l nodes=1:ppn=1,walltime=1:00:00,mem=2gb
#PBS -N SoftBam
#PBS -M mid224@nyu.edu
#PBS -m a
#PBS -j eo

cd ${PBS_O_WORKDIR}

#This script creates soft links to BAM files produced by the Tophat step for use in downstream easyRNAseq counts.
# It creates soft links to both bam and bai files in the bam_links directory
# They are named using Sample IDs and stored in the bam_links directory for easier downstream use

# source config file to get SAMPLE_TABLE
source Config.txt
```

```
#read through Sample Table to get sample ids
while IFS=$'\t' read ID FQ1 FQ2 CD;do
        # Create symbolic links to bam and bai files in directory structures bam_files folder
        # Check if they already exist first
        if [ -e "/${PWD}/alignments.1/bam_links/Sample_${ID}.bam" ]
        then
                echo "Can't make file Sample_${ID}.bam; file already exists."
        else
                ln -s /${PWD}/alignments.1/Sample_${ID}/accepted_hits.bam /${PWD}/alignments.1/bam_links/Sample_${ID}.bam
        fi

        if [ -e "/${PWD}/alignments.1/bam_links/Sample_${ID}.bam.bai" ]
        then
                echo "Can't make file Sample_${ID}.bam.bai; file already exists."
        else
                ln -s /${PWD}/alignments.1/Sample_${ID}/accepted_hits.bam.bai /${PWD}/alignments.1/bam_links/Sample_$
{ID}.bam.bai
        fi

done < ${SAMPLE_TABLE}
```

****

Appendix K: "simpleRNASeq.pbs": Scheduler script launches call to simpleRNASeq count tool

```
#!/bin/bash

#PBS -l nodes=1:ppn=8,walltime=24:00:00,mem=8gb
#PBS -N simpleRNASeq
#PBS -M mid224@nyu.edu
#PBS -m a
#PBS -j eo

cd ${PBS_O_WORKDIR}

#This script calls the simpleRNASeq R script

#get configuration file variables
source Config.txt

#initialize variable to hold list of bam file names
filelist=""

#read through Sample Table to derive bam file names, add to filelist
while IFS=$'\t' read ID FQ1 FQ2 CD;do
        filelist="${filelist}alignments.1/bam_links/Sample_${ID}.bam "

done < ${SAMPLE_TABLE}

#load R module
module load r/intel/3.2.0

#Call R script to run easyRNAseq on pipeline's BAM files; pass in filelist
R --file=simpleRNASeq.R --vanilla --args ${GFF} $filelist > simpleRNASeq.${PBS_JOBID}.log 2>&1
```

****

Appendix L: "simpleRNASeq.R": R script that runs simpleRNASeq count analysis on pipeline's BAM files

```
#!/usr/bin/env Rscript

#This R script runs simpleRNASeq on BAM alignments stored in the pipeline's directory structure of 'alignments.1'
#It produces a table of counts for features (genes) for the samples
#This script needs the Bioconductor core packages and easyRNASeq package to be installed in order to run.

#get arguments
args<-commandArgs(trailingOnly=TRUE)

#load easyRNASeq
library(easyRNASeq)

#get bam filenames from arguments
filenames<-c(args[2:length(args)])
```

```
#assing to bamFiles object
bamFiles<-getBamFileList(filenames)

#set parameters for annotaion, bam files
annotParam <- AnnotParam(datasource=args[1], type="gff3")
bamParam <- BamParam(stranded=FALSE)

#set parameters for RNASeq experiment; incorporates annotParam, bamParam
rnaSeqParam <- RnaSeqParam(countBy="genes", annotParam=annotParam, bamParam=bamParam)

#get summarized experiment object, which contains counts
sexp <- simpleRNASeq(bamFiles=bamFiles,
param=rnaSeqParam,
nnodes = 8,
verbose=TRUE
)

#use assay function to extract count table from summarized experiment; assign to count.table object
count.table<-assay(sexp)

#print preview of count table
print("Printing the head of assay of the count table")
head(count.table)

#'count.table' R object printed to a text file
write.table(count.table,file="count.table.txt",col.names=NA,quote=F,sep="\t")

*********************
```

Appendix M: "DESeq.pbs": Scheduler for DESeq differential expression arguments and call

```
#!/bin/bash

#PBS -l nodes=1:ppn=8,walltime=24:00:00,mem=8gb
#PBS -N DESeq
#PBS -M mid224@nyu.edu
#PBS -m a
#PBS -j eo

cd ${PBS_O_WORKDIR}

#This PBS scheduler script calls a scrip to create "condition" arguments for DESeq
#It then calls DESeq on those arguments and the count table produced by the pipeline

#source configuration file variables
source Config.txt

#call perl script to write Condition table using count table and control file
#this provides "condition" (treatment vs. control) arguments for DESeq R call
#checks to match headers from count table to appropriate conditions, listed in control file

#first check if condition table has already been created; call perl script if not
if [ -e "ConditionTable.txt" ]
then
        echo "Can't make file ConditionTable.txt; file already exists."
else
        perl ConditionWriter.pl count.table.txt ${SAMPLE_TABLE}
fi

#initialize variable to hold list of conditions
cdlist=""

#read through Condition Table to get sample conditions
while IFS=$'\t' read BAM CD;do
        cdlist="${cdlist}${CD} " #make list, separated by spaces, of conditions in correct order
done < ConditionTable.txt

#remove trailing space of conditions list
cdlist=${cdlist%?}

#load R module
module load r/intel/3.2.0

#Call R script to run easyRNAseq on pipeline's BAM files, passing in count table and conditions
```

```
R --file=DESeq.R --vanilla --args count.table.txt ${cdlist} > DESeq.${PBS_JOBID}.log 2>&1
```

****

Appendix N: "ConditionWriter.pl": Matches count table column headers to correct conditions in control file, prints pairs to condition table (used by DESeq)

```
#!/usr/bin/perl -w
#use warnings;
#use strict;

#open count table for reading, from arguments
my $count_file = $ARGV[0];
open (COUNT, "<$count_file") or die "Can't open file:$!";

#open text file to append condition info
open (OUT, '>>ConditionTable.txt');

#get first line of count table to get column headers
my $firstline = <COUNT>;
chomp $firstline; #chomp EOL character

#read row into array, split on tab
my @row = split(/\t/,$firstline);
my $id = "";    #initialize variable to hold id of column header

for($i=1; $i<(@row); $i++)      {
        #print "$row[$i]\n";
        $row[$i]  =~ /^(.*)\./; #remove the '.bam' from column header
        $id = $1;               #assign to id
        $id =~ s/Sample_//;     #remove 'Sample_' from column header

        #get control file name from arguments, and open for reading
        my $cntrl_file = $ARGV[1];
        open (CNTRL, $cntrl_file) or die "Can't open file:$!";
        #while loop goes through control file
        while (<CNTRL>) {
                my @cntrl_row = split(/\t/);    #split on tab
                #'if' statement finds ID (column 0) that matches id from count table headers
                #at match, print id and corresponding condition (column 3 of control file) to condition tableI
                if ($id eq $cntrl_row[0]) {
                        print OUT "$row[$i]\t$cntrl_row[3]";
                }
        }
        close(CNTRL);
}

close (COUNT);
close (OUT);
```

****

Appendix O: "DESeq.R": R script that applies DESeq differential expression analysis to count table of feature expression

```
#!/usr/bin/env Rscript

#This R script runs DESeq to evaluate differential expression on the count table produced by simpleRNASeq
#This script needs the Bioconductor core packages and DESeq package to be installed in order to run.

#get arguments
args<-commandArgs(trailingOnly=TRUE)

#load DESeq
library("DESeq")

#Get count.table produced by simpleRNASeq (R code)
datafile = args[1]
countTable = read.table( datafile, header=TRUE, row.names=1)

#get conditions from arguments
cds<-c(args[2:length(args)])

#create condition object
condition = factor( cds )
```

```
#Make data.frame with experimental-design information (R code)
experimentDesign = data.frame(
        row.names = colnames( countTable ),
        condition, libType = "paired-end")

#instantiate a CountDataSet ('central data structure in DESeq')
library ( "DESeq" )
cds = newCountDataSet( countTable, condition )

#Normalization (effective library size)
cds = estimateSizeFactors( cds ) #now can use 'normalized=TRUE' when getting counts

#Variance estimation
cds = estimateDispersions( cds ) #needed to call differential expression

#Calling differential expression
deseq_result = nbinomTest( cds, "control", "treatment" )

#perform basic plotting from deseq results, and print to PDF
pdf("deseq_plots.pdf") #open pdf writer for plots
plotMA(deseq_result)    #plot log2 fold changes against mean normalized counts
hist(deseq_result$pval, breaks=100, col="skyblue", border="slateblue", main="") #plot histogram of p values
dev.off()       #close plotting device

#Save output to file
write.table(deseq_result,"DESeq.output.txt")

********************
```

Appendix P: "PreProc_master.sh": Master script for the PreProcessing pipeline

```
#!/bin/bash

#This is the master shell script for the
#PreProcessing Pipeline for FastQ files.
#It checks for Illumina QC filtering, read IDs, and sample-file length.
#It also funs FASTQC
#It can be run before the RNAseq pipeline.
#It is configured for the Mercer cluster.


#set configuration file variable
CONFIG=Config.txt

#check that configuration file exists
if [ ! -f ${CONFIG} ]; then
        echo _ERROR_ configuration file not found
        exit 1
fi

#define configuration variables, including getting control file
source ${CONFIG}

#get number of samples
# *** SAMPLE_TABLE variable defined in configuration file ***
NUMSAMPLES=$(wc -l < ${SAMPLE_TABLE})

#call script to check FastQ lengths and Illumina filter
qsub LengthFilter.pbs

#call script to run FASTQC
qsub -t 1-${NUMSAMPLES} FastQC.pbs

#call PBS script to check read identifiers in forward samples match pairs in reverse samples
qsub ReadID.pbs

*********************************
```

Appendix Q: "LengthFilter.pbs": Checks lengths and Illumina QC filtering of FastQ files

```
#!/bin/bash
#PBS -l nodes=1:ppn=4,walltime=8:00:00,mem=4gb
#PBS -N LengthFilter
```

```
#PBS -M mid224@nyu.edu
#PBS -m a
#PBS -j eo

cd ${PBS_O_WORKDIR}

#This script checks FastQ file lengths and checks for Illumina QC filtering

#set configuration file variable
CONFIG=Config.txt

#check that configuration file exists
if [ ! -f ${CONFIG} ]; then
        echo _ERROR_ configuration file not found
        exit 1
fi

#get congiuration variables, including sample table
source ${CONFIG}

##get number of samples
# *** SAMPLE_TABLE variable defined in configuration file ***
NUMSAMPLES=$(wc -l < ${SAMPLE_TABLE})

#read in Sample FastQ.gz paths
#read through Sample file with Bash loop.
while IFS=$'\t' read ID FQ1 FQ2 CD;do
        #get lengths of forward and reverse files
        L1=`gunzip -c "$FQ1" | wc -l`
        L2=`gunzip -c "$FQ2" | wc -l`

        #check if forward and reverse files are not same length, report to log file
        if [ $L1 -ne $L2 ]; then
                echo "Forward read $FQ1 is not the same length as reverse read $FQ2." >> PreProcLog.txt
        fi

        #check for filtered reads and report to log file
        G1=`gunzip -c "$FQ1" | grep -c '^@.* [^:]*:Y:[^:]*:'`
        echo "There are $G1 filtered reads in $FQ1" >> PreProcLog.txt
        G2=`gunzip -c "$FQ2" | grep -c '^@.* [^:]*:Y:[^:]*:'`
        echo "There are $G2 filtered reads in $FQ2" >> PreProcLog.txt
done < ${SAMPLE_TABLE}

**************************************
```

Appendix R: "FastQC.pbs": Array job scheduler for running FastQC

```
#!/bin/bash
#PBS -l nodes=1:ppn=1,walltime=24:00:00,mem=8000mb
#PBS -N FastQC
#PBS -M jmf11@nyu.edu
#PBS -m a
#PBS -j eo

cd ${PBS_O_WORKDIR}

# This PBS script schedules the array jobs for running FastQC on all FastQ files

# source config file to get SAMPLE_TABLE path
source Config.txt

# Throw error if pbs script
if [ -z ${PBS_ARRAYID} ]; then
        echo _ERROR_ expecting an array job
        exit 1
fi

#load module
module load fastqc/0.11.2

# Get fastqs paths from sample table for current subjob using ${PBS_ARRAYID}
IFS=$'\t'
line=$(sed -n "${PBS_ARRAYID}p" "${SAMPLE_TABLE}")
read -r ID FQ1 FQ2 CD <<< "$line"       #assign to variable for Sample ID, forward and reverse FastQ, and Condition
```

```
#Call FastQC bash script on Sample for current array ID; pass in FastQ paths
bash FastQC.sh ${FQ1} ${FQ2} > fastqc.${PBS_ARRAYID}.${PBS_JOBID}.log 2>&1

*****************************************
```

Appendix S: "FastQC.sh": Runs FastQC check on forward and reverse FastQ files

```
#!/bin/bash

#This script runs FastQC check on on a forward and reverse FastQ file

#FastQ files are passed in from PBS array job; assign to variables
FQ1=$1
FQ2=$2

#check that sample paths exist
if [ -z ${FQ1} ]; then
        echo _ERROR_ usage error
fi

if [ -z ${FQ2} ]; then
        echo _ERROR_ usage error
fi

# FastQCcheck on current F and R FASTQ files
fastqc "${FQ1}"
fastqc "${FQ2}"

************************************
```

Appendix T: "ReadID.pbs": Checks the read IDs in FQ1 against their pairs in FQ2

```
#!/bin/bash
#PBS -l nodes=1:ppn=4,walltime=48:00:00,mem=4gb
#PBS -N ReadID
#PBS -M mid224@nyu.edu
#PBS -m a
#PBS -j eo

cd ${PBS_O_WORKDIR}

#This script checks the read IDs in the forward file against the reverse file, and logs any errors

#set configuration file variable
CONFIG=Config.txt

#check that configuration file exists
if [ ! -f ${CONFIG} ]; then
        echo _ERROR_ configuration file not found
        exit 1
fi

#define configuration variables, including getting control file
source ${CONFIG}

#read in Sample FastQ.gz paths
#read through Sample file with Bash loop
while IFS=$'\t' read ID FQ1 FQ2 CD; do

        #match counter, keeps track of what match (m'th match) of read ID line text ("@HWI")
        m=0

        #read through FQ1, grepping read ID lines
        gunzip -c "$FQ1" | grep -m 598195 '^@HWI' | while read ID1 BC1 file; do
                ((m++)) #increment match counter for FQ1
                #find m'th @HWI line in FQ2
                gunzip -c "$FQ2" | grep -m "$m" '^@HWI'| tail -n 1 | while read ID2 BC2 file; do
                        #check mth ID from FQ1 against mth ID from FQ2
                        if [ $ID1 != $ID2 ]; then
                                #Report to log file if mismatch
                                echo "Read identifiers did not match between forward and reverse." >> PreProcLog.txt
                                echo "Forward read $ID1 in $FQ1 does not have the same read identifier as reverse read $ID2
```

```
in $FQ2." >> PreProcLog.txt
                        fi
                done
        done

done < ${SAMPLE_TABLE}
```

****

Appendix U: "Config.txt": This configuration file lists a control file with locally saved FastQ files, for use with the PreProcessing pipeline
(Gunzipping FastQ files requires permissions, and student did not have these for where FastQ files for rice were stored on NGS course directory.)

```
#This is the configuration file for the RNA seq pipeline.
#It contains variables for the reference genome (REF), the fasta index
#(FASTA_IND), the GFF file (GFF), the base of the genome's ID (GENOME_BASE),
#and the control file (SAMPLE_TABLE). Please assign values to these variables
#using the following syntax: VARIABLE=value.

REF=/scratch/courses/BI7653/project.RNAseq/rice7_Chr11.fa
FASTA_IND=/scratch/courses/BI7653/project.RNAseq/rice7_Chr11.fa.fai
GFF=/scratch/courses/BI7653/project.RNAseq/rice7_Chr11.gff3
GENOME_BASE=rice7_Chr11

SAMPLE_TABLE=cntrlfile_local.txt
```

****

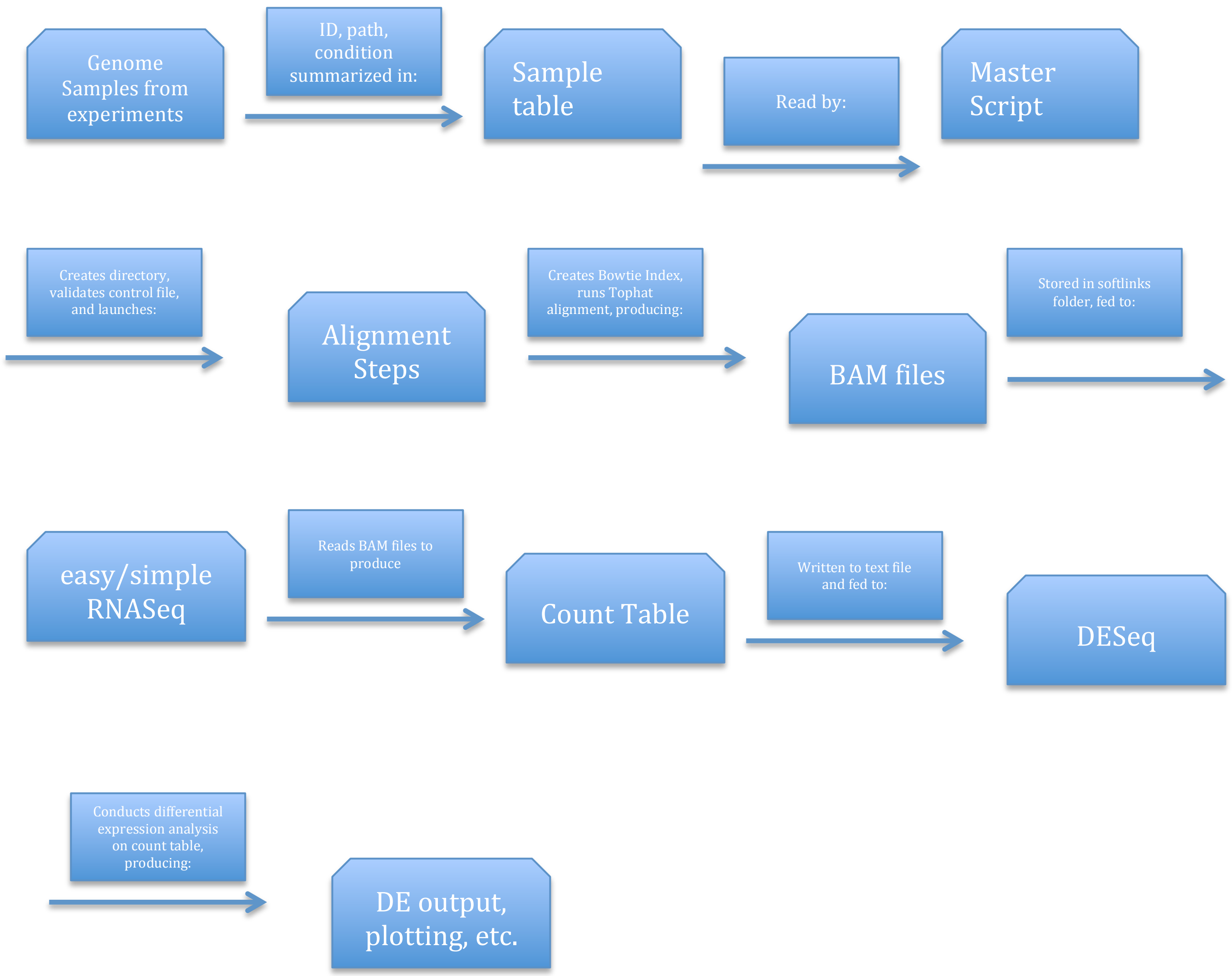
Appendix V: "cntrlfile_local.txt": See above. This control file contains paths to locally saved FastQ files.

```
ID012   ID012_R1.chr11.fastq.gz ID012_R2.chr11.fastq.gz control
ID156   ID156_R1.chr11.fastq.gz ID156_R2.chr11.fastq.gz control
```

****

Appendix W
Graphical Flow Chart for Main Pipeline
Michael Dhar
Guided Study
Spring/Summer 2015

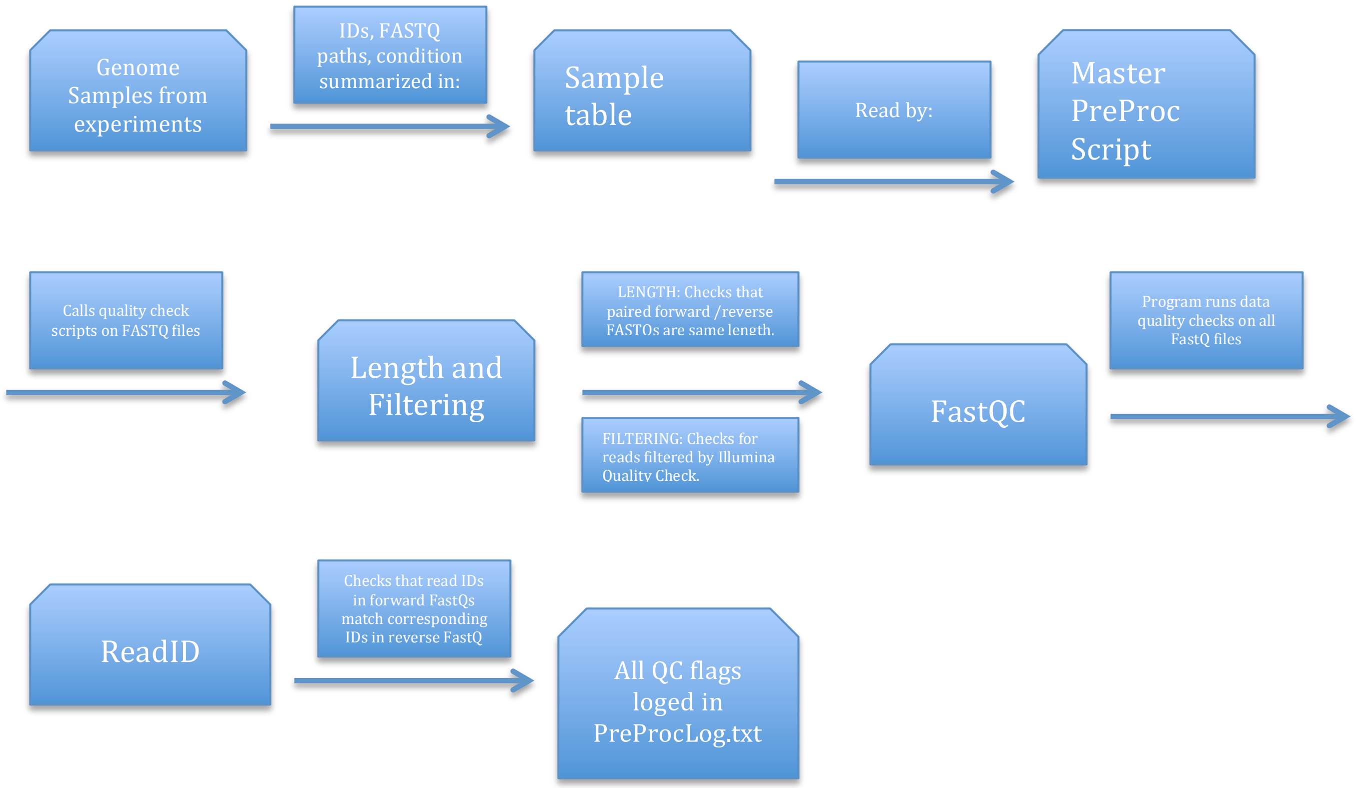
MAIN PIPELINE FLOWCHART:



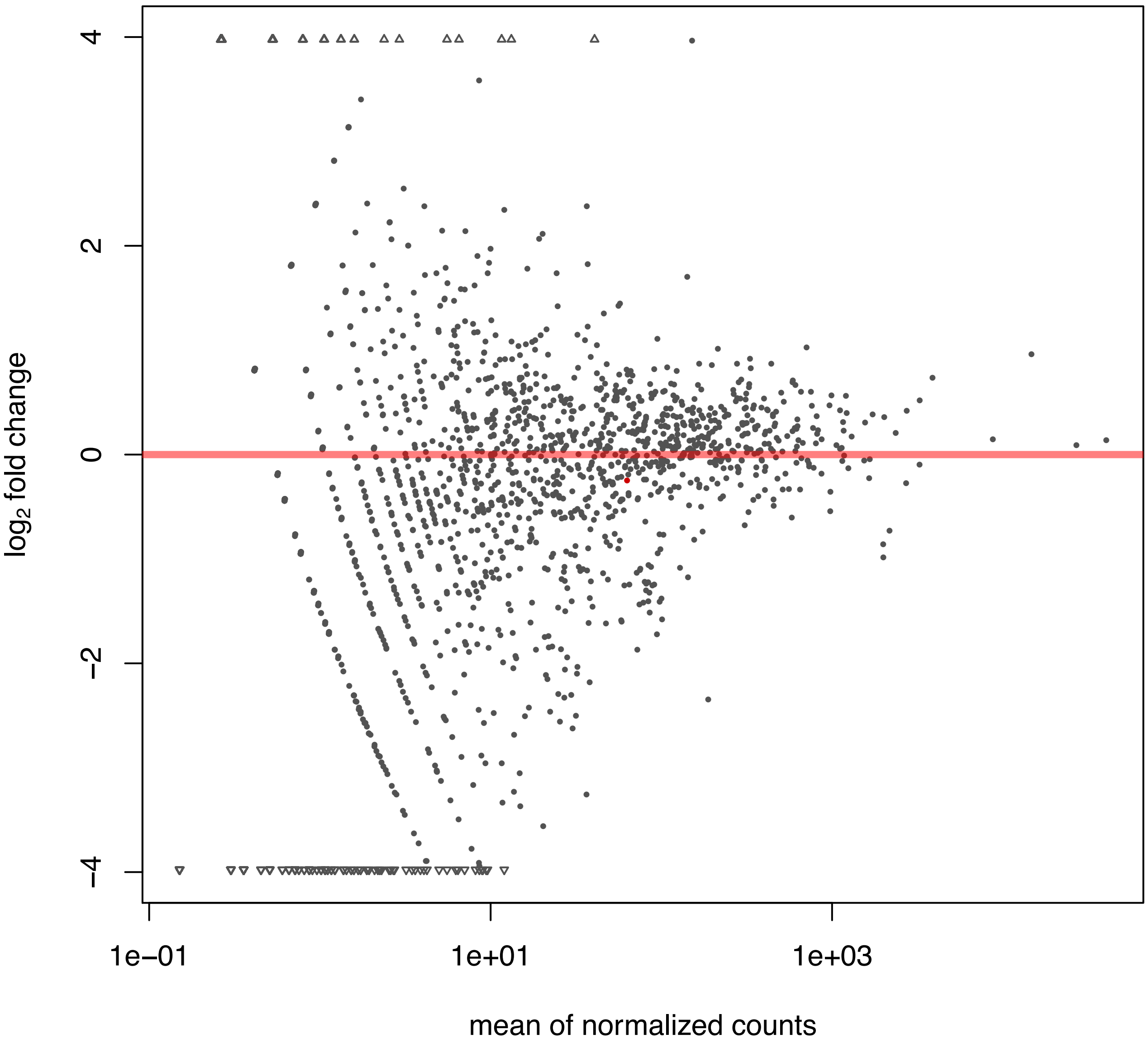
Genome Samples from experiments
ID, path, condition summarized in:
Sample table
Read by:
Master Script
Creates directory, validates control file, and launches:
Alignment Steps
Creates Bowtie Index, runs Tophat alignment, producing:
BAM files
Stored in softlinks folder, fed to:
easy/simple RNASeq
Reads BAM files to produce
Count Table
Written to text file and fed to:
DESeq
Conducts differential expression analysis on count table, producing:
DE output, plotting, etc.

Appendix X
Graphical Flow Chart for PreProcessing Pipeline
Michael Dhar
Guided Study
Spring/Summer 2015

PREPROCESSING PIPELINE FLOWCHART:

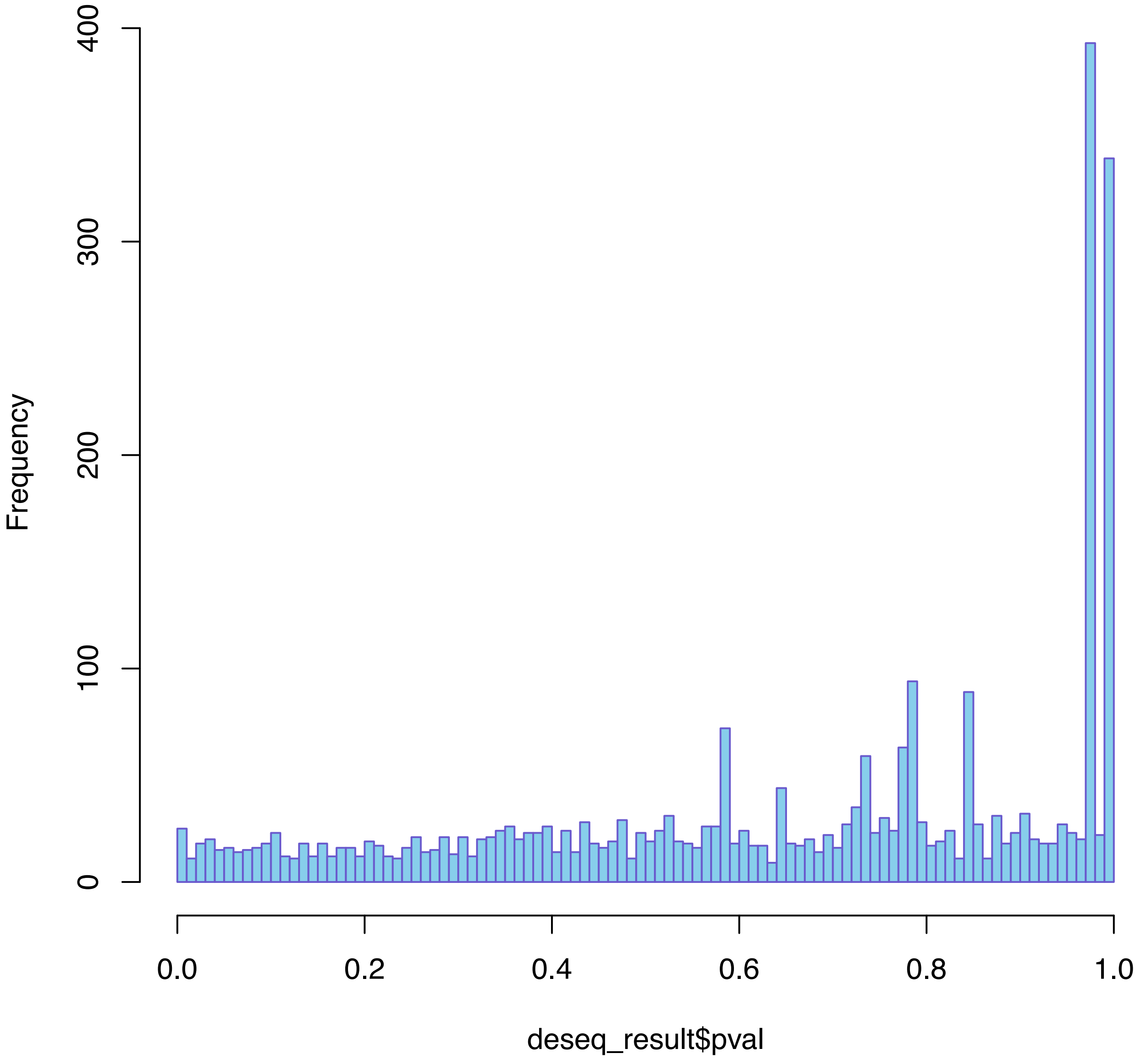

Genome Samples from experiments
IDs, FASTQ paths, condition summarized in:
Sample table
Read by:
Master PreProc Script
Calls quality check scripts on FASTQ files
Length and Filtering
LENGTH: Checks that paired forward /reverse FASTQs are same length.
FILTERING: Checks for reads filtered by Illumina Quality Check.
FastQC
Program runs data quality checks on all FastQ files
ReadID
Checks that read IDs in forward FastQs match corresponding IDs in reverse FastQ
All QC flags loged in PreProcLog.txt

Appendix Y



4
2
0
−2
−4
log2 fold change
1e−01
1e+01
1e+03
mean of normalized counts

Appendix Z


Frequency
deseq_result$pval