Compiled Date Palm RNA-Seq Pipeline Scripts

#Config.txt

```
#This is the configuration file for the RNAseq STAR pipeline, on
BUTINAH.
#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). It also contains variables for
two STAR
#index arguements:
#sjdbGTFfeatureExon (FEATURE) and sjdbGTFtagExonParentTranscript
(PARENT)
#Please assign values to these variables
#using the following syntax: VARIABLE=value.

REF=/scratch/phoenix/Fruit_RNAseq/NCBI_phoenix_Feb2016.modified.fa
GFF=/scratch/phoenix/Fruit_RNAseq/ref_DPV01_scaffolds.gff3
GENOME_BASE=NCBI_phoenix_Feb2016.modified

SAMPLE_TABLE=RNAseq_table
FASTQ_DIR=/scratch/phoenix/Fruit_RNAseq

FEATURE=exon
PARENT=Parent

#########################################################################
```

#SCRIPT 1: Validator.pl

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

#This Validator checks the format of the control (sample) file for the
STAR 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 3 columns
                if (@row != 3) {
                        die "Row $row_no has $row_length columns. All
rows must have 3 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);

print "Validator script completed.\n";


#########################################################################
####

#SCRIPT 2: FastQ_Check.pbs

#!/bin/bash

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

cd ${PBS_O_WORKDIR}



#This script loops through Datepalm sample table file

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

find ${FASTQ_DIR} -type f -name '*.gz' -exec md5sum "{}" + >
FastQ_md5sum.txt

while IFS=$'\t' read ID FQ1 FQ2 ;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}


#########################################################


#SCRIPT 3: RNAseq_master_Star.sh

#!/bin/bash

#This is the master shell script for the
# RNA-seq pipeline with STAR aligner. It is configured for the
# Butinah 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_Star.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 Star index
#Complete Star index step before running alignments
INDEX=`qsub StarIndex.pbs`

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

#Run bam softlink script on star-aligned BAM files
# *** runs b 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`

################################################

#SCRIPT 4: DirStruct.sh

#/bin/bash

# This script creates the directory structure for the RNAseq pipeline
(with STAR aligner), within the user-created 'project' directory

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

#check if 'starDir' directory (for STAR indexes) exists for alignment
step; create if not
if [ -d "starDir" ]
then
        echo "Can't make directory starDir; directory already exists."
else
        mkdir starDir
fi

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

############################################################

#SCRIPT 5: StarIndex.pbs

#!/bin/bash

#PBS -l nodes=1:ppn=4,walltime=24:00:00,mem=90gb
#PBS -q bigmem
#PBS -N StarIndex
#PBS -M mid224@nyu.edu
#PBS -m a
#PBS -j eo

cd ${PBS_O_WORKDIR}

#This script calls the Star Index creator

#module load star/intel/2.3.0e #using new star, via path below, now
```

```
#give star program path
PATH=$PATH:/scratch/gencore/software/STAR/2.5.0c/bin/
export PATH

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

#############################################################

#SCRIPT 6: StarIndex.sh

#/bin/bash

#This script creates the Star index

#Get reference genome (REF) and GFF file (GFF) from Config file
source Config.txt

STAR --runMode genomeGenerate \
--genomeDir starDir \
--genomeFastaFiles ${REF} \
--runThreadN 4 \
--sjdbGTFfile ${GFF} \
--sjdbGTFfeatureExon ${FEATURE} \
--sjdbGTFtagExonParentTranscript ${PARENT} \
--sjdbOverhang 100 \
--genomeChrBinNbits 17 \
--limitGenomeGenerateRAM 90000000000 \
--sjdbGTFtagExonParentGene gene


echo _ESTATUS_ [ STAR genomeGenerate ]: $?
echo _END_ $(date)

#################################################################

#SCRIPT 7: StarAlign.pbs

#!/bin/bash

#PBS -l nodes=1:ppn=4,walltime=48:00:00,mem=44gb
#PBS -N StarAlign
#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 <<< "$line"  #assign to variable for Sample ID,
forward and reverse FastQ, and Condition

# Load modules
#using new star, via path, below
#samtools loaded in StarAlign.sh

#give star program path
PATH=$PATH:/scratch/gencore/software/STAR/2.5.0c/bin/
export PATH

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

################################################

#SCRIPT 8: StarAlign.sh

#/bin/bash

# This script runs the STAR aligner on a forward and reverse FastQ

# 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

# Star alignment command
STAR --runMode alignReads \
--genomeDir starDir \
--runThreadN 4 \
--readFilesIn \
${FQ1} \
${FQ2} \
--readFilesCommand zcat \
--outSAMmapqUnique 255 \
--outFileNamePrefix alignments.1/Sample_${ID}/ \
--outSAMtype BAM SortedByCoordinate

echo _ESTATUS_ [ STAR alignReads]: $?
echo _END_ [ STAR alignReads ]: $(date)

#Samtools steps
#load Samtools
module load NYUAD/2.0 gcc zlib openssl ncurses samtools/1.2

#filter BAM for 255 mapq
samtools view -b -q 255 alignments.1/Sample_${ID}/
Aligned.sortedByCoord.out.bam > alignments.1/Sample_${ID}/
Aligned.sortedByCoord.filtered.out.bam

#make bai index
samtools index alignments.1/Sample_${ID}/
Aligned.sortedByCoord.filtered.out.bam

##################################################

#SCRIPT 9: SoftBam.pbs

#!/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 STAR 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=$'\t' read ID FQ1 FQ2;do
        # Create symbolic links to bam and bai files in directory
structures bam_files folder
        # Check if they already exist first
        if [ -e "/${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}/
Aligned.sortedByCoord.filtered.out.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_${ID}/
Aligned.sortedByCoord.filtered.out.bam.bai /${PWD}/alignments.1/
bam_links/Sample_${ID}.bam.bai
        fi

done < ${SAMPLE_TABLE}

############################################################

#SCRIPT 10: simpleRNASeq.pbs

#!/bin/bash

#PBS -l nodes=1:ppn=12,walltime=48:00:00,mem=46gb
#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=$'\t' read ID FQ1 FQ2;do
        filelist="${filelist}alignments.1/bam_links/Sample_${ID}.bam "

done < ${SAMPLE_TABLE}

#load R module
module load r/gnu/3.0.2
#load bioconductor/easyRNASeq packages
module load NYUAD/2.0 apps anyenv/1

#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


############################################################

#SCRIPT 11: simpleRNASeq.R

#!/usr/bin/env Rscript

#This R script runs simpleRNASeq on BAM alignments stored in the STAR
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)])
#passing 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 = 12,
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,se
p="\t")


##########################################################################
####

################################END#####################################
###
```