RNA-seq example

(transferred from the old wiki)

For CentOS 7 users...

you need to make two changes when applying this pipeline, due to system upgrade from CentOS 6 to CentOS 7 in October 2018.

1. Job submission script. We now use SLURM as the job scheduler. Therefore, all the qsub scripts given here have to be adapted to SBATCH scripts on your own. We've provided some SLURM tutorial on our wiki site. In addition, **you should constrain the jobs only to intel16 or intel18 compute nodes**. To do so, submit your job as **`sbatch --constraint="intel16|intel18" <your slurm script>`**
2. Module loading for each software. All the "module load" commands presented in this tutorial no longer work for CentOS 7. Please follow commands below for loading each of them. In each step of the pipeline, **always begin with "module purge"** so as to start off with a clean slate.

Using bowtie2
```
module purge
module load GCC/5.4.0-2.26 OpenMPI/1.10.3
module load Bowtie2/2.3.2
```
Using tophat (note that tophat calls bowtie and samtools so we load these two as well)
```
module purge
module load GCC/5.4.0-2.26 OpenMPI/1.10.3
module load TopHat/2.1.1 Bowtie2/2.3.2 SAMtools/1.5
```
Using cufflinks
```
module purge
module load GCC/5.4.0-2.26 OpenMPI/1.10.3
module load Cufflinks/2.2.1
```
Using samtools
```
module purge
module load GCC/5.4.0-2.26 OpenMPI/1.10.3
module load SAMtools/1.5
```
Using htseq ("htseq-count" and "htseq-qa" are built-in with Python 3.6.4)
```
module purge
module load GCC/6.4.0-2.28 OpenMPI/2.1.2 Python/3.6.4
```
Using R
```
module purge
module load GCC/8.3.0 OpenMPI/3.1.4 R/4.0.2
```

Workflow and data files

The pipeline for a RNA-seq analysis typically includes the following steps:

1. map (align) reads to genome;

2. assemble transcripts for each sample (possibly guided by reference annotation), and merge assemblies from multiple samples into a master transcriptome;
3. quantify expressions of transcripts or genes;
4. detect genes differentially expressed under different experimental conditions.

Below we list the locations of data files we will be using:

Reference genome (GRCh38, unmasked): **`/mnt/research/common-data/Bio/Ensembl_GRCh38_unmasked_ref/GRCh38.fa`**

Reference GTF (GRCh38): **`/mnt/research/common-data/Bio/Ensembl_GRCh38_GTF/GRCh38.gtf`**

RNA-seq fastq files for 4 samples (paired-end reads): **`/mnt/research/common-data/Examples/RNASeq-Model/data/`** where we have, for example, sample ERR315382's data in **`ERR315382_1.fastq`** and **`ERR315382_2.fastq`**

Notes

- Quality Check (QC) of raw fastq files prior to alignment is not covered here; you can find such information in another tutorial.
- Parameters used in each software tool are not explained. For the purpose of demonstration, they've been chosen to minimize computational burden. You will want to modify commands illustrated in the example to meet your own need.
- RNA-seq analysis has many different protocols; in each step of the following pipeline, alternative software tools can be chosen. That said, our primary purpose is to provide guidance on setting up batch submission scripts through PBS, and how much computing resource one needs to request for analyzing a human transcriptome data set.

Pipeline code

Step 1: map reads to reference genome using `Tophat`

First, we need to build Bowtie indexes for reference genome.

Submit the following job in `/mnt/research/common-data/Bio/Ensembl_GRCh38_unmasked_ref/` (i.e., run "`qsub bowtie-build.qsub`"):

bowtie-build.qsub
```
#!/bin/bash --login
```

```
#PBS -l walltime=3:50:00
#PBS -l mem=15gb
#PBS -l nodes=1:ppn=4
#PBS -m abe
#PBS -j oe

module swap GNU GNU/4.4.5
module load bowtie2/2.3.1

cd $PBS_O_WORKDIR

bowtie2-build --threads 4 GRCh38.fa GRCh38 > bowtie2-build.log 2>&1

qstat -f $PBS_JOBID
```

By looking at qsub output (i.e., `*.qsub.o*`) file, we notice

```
resources_used.cput = 04:14:23
resources_used.energy_used = 0
resources_used.mem = 9474548kb
resources_used.vmem = 11326804kb
resources_used.walltime = 01:40:39
```

This information should help you request an appropriate amount of memory and wall time next time.

Second, we need to build bowtie transcriptome indexes for reference GTF. This need be done only once, and the built indexes can be reused. Tophat v2.1.1 is needed because this version incorporated Luca Venturini's code to support large bowtie2 indexes (.bt2l).

Submit the following job in `/mnt/research/common-data/Bio/Ensembl_GRCh38_GTF/`:

build-gtf-index.qsub
```
#!/bin/bash --login

#PBS -l walltime=3:50:00
#PBS -l mem=15gb
#PBS -l nodes=1:ppn=1
#PBS -m abe
#PBS -j oe

module swap GNU GNU/4.4.5
module load TopHat2/2.1.1

cd $PBS_O_WORKDIR

tophat2 -G /mnt/research/common-data/Bio/Ensembl_GRCh38_GTF/GRCh38.gtf \
  --transcriptome-index=/mnt/research/common-data/Bio/Ensembl_GRCh38_GTF/GRCh38 \
  /mnt/research/common-data/Bio/Ensembl_GRCh38_unmasked_ref/GRCh38

qstat -f $PBS_JOBID
```

Resource used:

```
resources_used.cput = 00:24:06
resources_used.energy_used = 0
resources_used.mem = 675676kb
resources_used.vmem = 1172192kb
resources_used.walltime = 00:33:03
```

Now we are ready to do the alignment. Prepare a job submission file, `tophat2.qsub`, in your working directory, as shown below:

tophat2.qsub
```
#!/bin/bash --login

#PBS -l walltime=15:00:00
#PBS -l mem=30gb
#PBS -l nodes=1:ppn=8
#PBS -m abe
#PBS -j oe

module swap GNU GNU/4.4.5
module load TopHat2/2.1.1

cd $PBS_O_WORKDIR

# The shell variable "sample" is provided by qsub command line through "-v xxx"
sampleFastq1=/mnt/research/common-data/Examples/RNASeq-Model/data/${sample}_1.fastq
sampleFastq2=/mnt/research/common-data/Examples/RNASeq-Model/data/${sample}_2.fastq

if [ ! -d $sample/map ]; then
  mkdir -p $sample/map
else
  if [ -e $sample/map/tophat.log ]; then
    rm $sample/map/tophat.log
  fi
fi

# Run Tophat
tophat2 -p 8 -o $sample/map \
  --transcriptome-index=/mnt/research/common-data/Bio/Ensembl_GRCh38_GTF/GRCh38 \
  --read-realign-edit-dist 0 \
  -g 1 -x 1 -m 2 -r 80 \
  --library-type fr-unstranded \
  /mnt/research/common-data/Bio/Ensembl_GRCh38_unmasked_ref/GRCh38 $sampleFastq1
$sampleFastq2 > $sample/map/tophat.log 2>&1

qstat -f $PBS_JOBID
```

Above, we create sample specific directories within the working directory, and then put Tophat's output files in a subdirectory named "`map`".

You should know what scale is used to encode base quality in your fastq. Normally it's phred+33. Otherwise, you probably need to add an extra option "`--solexa1.3-quals`" in the Tophat command line. Check Tophat manual. We also have a script to help you figure out the scale:

Check quality encoding of fastq
```
awk 'NR % 4 == 0' [your fastq file] | /mnt/research/common-
data/Bio/scripts/identify_phred_scale/guess-encoding.py -n 1000
```

Now, submit `tophat2.qsub` for each sample in your working directory:

Bash script for submitting jobs
```
for S in ERR315325 ERR315326 ERR315382 ERR315424
do
  qsub -v sample="$S" tophat2.qsub
done
```

The actual wall time would vary from sample to sample, and so it's better to request a sufficient long time.

Step 2: assemble transcripts for each sample using `cufflinks`

First, we will generate a mask GTF so `cufflinks` can ignore reads that could have come from transcripts in this mask GTF file:

Generate mask GTF
```
awk '$1=="MT" || $0 ~/\<rRNA\>/ || $0 ~/\<tRNA\>/' /mnt/research/common-
data/Bio/Ensembl_GRCh38_GTF/GRCh38.gtf > mask.gtf
```

Then, prepare a job submission script, `cufflinks.qsub`, in the working directory, as shown below (in this tutorial the working directory is set to `/mnt/research/common-data/Examples/RNASeq-Nanye/`):

cufflinks.qsub
```
#!/bin/bash --login

#PBS -l walltime=3:50:00
#PBS -l mem=5gb
#PBS -l nodes=1:ppn=4
#PBS -m abe
#PBS -j oe

module swap GNU GNU/4.4.5
module load cufflinks/2.2.1

cd $PBS_O_WORKDIR

# "sample" is provided by qsub command line through "-v xxx"
sample_bam=/mnt/research/common-data/Examples/RNASeq-
Nanye/${sample}/map/accepted_hits.bam

if [ ! -d $sample/cufflinksOut ]; then
  mkdir -p $sample/cufflinksOut
else
  if [ -e $sample/cufflinksOut/cufflinks.log ]; then
    rm $sample/cufflinksOut/cufflinks.log
  fi
fi


# Run cufflinks
cufflinks -p 4 -o $sample/cufflinksOut \
  -M mask.gtf -u -N \
  --max-bundle-frags 50000 \
  -g /mnt/research/common-data/Bio/Ensembl_GRCh38_GTF/GRCh38.gtf \
  -b /mnt/research/common-data/Bio/Ensembl_GRCh38_unmasked_ref/GRCh38.fa \
  $sample_bam > $sample/cufflinksOut/cufflinks.log 2>&1

qstat -f $PBS_JOBID
```

Above, we create sample specific directories within the working directory, and then put cufflinks' output files in a subdirectory named "`cufflinksOut`".

The choice of --`max-bundle-frags` (50K) is unrealistically small here, for consideration of speed.

Similar to what we did in the alignment step, we now submit `cufflinks` job for each sample in the working directory:

Bash script for submitting jobs

```
for S in ERR315325 ERR315326 ERR315382 ERR315424
do
  qsub -v sample="$S" cufflinks.qsub
done
```

For the four sample considered here, the execution wall time with ppn=4 is short (less than 4 hours).

Step 3: merge assemblies from multiple samples into a master transcriptome using `cuffmerge`

We will run `cuffmerge` (a tool that comes along with `cufflinks`) in the working directory. `cuffmerge` can be run directly on our developer node, as shown below (with more samples, you should submit the job to the cluster, because on the dev-node, jobs running over 2 hours will be terminated by the system):

Run cuffmerge

```
# Create a list of all samples' GTF files: toMerge_list.txt
for sample in ERR315325 ERR315326 ERR315382 ERR315424
do
  echo /mnt/research/common-data/Examples/RNASeq-
Nanye/${sample}/cufflinksOut/transcripts.gtf >> toMerge_list.txt
done

if [ -d cuffmergeSummary ]; then
  rm -r cuffmergeSummary
fi

module load cufflinks/2.2.1

# Run cuffmerge
nohup cuffmerge -p 4 -o cuffmergeSummary \
  -g /mnt/research/common-data/Bio/Ensembl_GRCh38_GTF/GRCh38.gtf \
  -s /mnt/research/common-data/Bio/Ensembl_GRCh38_unmasked_ref/GRCh38.fa \
  toMerge_list.txt > cuffmerge.log 2>&1 &
```

The final output is a merged master GTF: `your_working_dir/cuffmergeSummary/merged.gtf`

Step 4: quantify gene expressions using `htseq-count`

In the main working directory, create a new subdirectory "`read_counts`"; for this task we will execute all commands in "`read_counts`".

`htseq-count` requires sorted bam files, therefore we need to sort each sample's alignment file (i.e. `accepted_hits.bam`) obtained from `Tophat`. See below, where we use `samtools sort`:

sortbam.qsub
```
#!/bin/bash --login

#PBS -l walltime=1:00:00
#PBS -l mem=2gb
#PBS -l nodes=1:ppn=1
#PBS -m abe
#PBS -j oe

module load SAMTools/1.5

cd $PBS_O_WORKDIR

# "sample" is provided by qsub command line through "-v xxx"
sample_bam=/mnt/research/common-data/Examples/RNASeq-
Nanye/${sample}/map/accepted_hits.bam

# Run samtools sort
samtools sort -n -o ${PBS_O_WORKDIR}/${sample}.nameSorted.bam -T
${PBS_O_WORKDIR}/${sample}_tmp $sample_bam

qstat -f $PBS_JOBID
```

Again, submit this bam-sorting job sample by sample:

Bash script for submitting jobs
```
for S in ERR315325 ERR315326 ERR315382 ERR315424
do
  qsub -v sample="$S" sortbam.qsub
done
```

Bam sorting is fast; it finished within 20 minutes for these samples.

Now we are ready to do the read counting job. That is, for a specific sample, we want to count how many reads map to each gene in the merged GTF.

count_reads.qsub

```
#!/bin/bash --login

#PBS -l walltime=1:00:00
#PBS -l mem=800mb
#PBS -l nodes=1:ppn=1
#PBS -m abe
#PBS -j oe

module load HTSeq/0.6.1

cd $PBS_O_WORKDIR

# "sample" is provided by qsub command line through "-v xxx"
htseq-count --format=bam --minaqual 50 --stranded=no --idattr=gene_id
${sample}.nameSorted.bam /mnt/research/common-data/Examples/RNASeq-
Nanye/cuffmergeSummary/merged.gtf > ${sample}.genecounts

qstat -f $PBS_JOBID
```

As usual, we submit the job for each sample, to get their gene counts.

Bash script for submitting jobs

```
for S in ERR315325 ERR315326 ERR315382 ERR315424
do
  qsub -v sample="$S" count_reads.qsub
done
```

Read counting is fast; it finished within 30 minutes for these samples.

Step 5: run differential gene expression analysis using `R {edgeR}`

We still stay in the "`read_counts`" directory for this step. We assume two samples are in the control group whereas the other two in the treatment group. Then, for each gene, we evaluate read count difference between control and treatment groups to see if the difference is statistically significant.

First, launch R 3.3.2:

launch R 3.3.2
```
module swap GNU GNU/4.9
module swap OpenMPI OpenMPI/1.10.0
module load R/3.3.2
```

In R, we can perform the following analysis. For more details, see edgeR's user guide.

edgeR example
```
library(edgeR)

# Read each sample individually
ctrl1 <- read.table('ERR315325.genecounts', sep='\t', as.is=T, header=FALSE)
ctrl2 <- read.table('ERR315326.genecounts', sep='\t', as.is=T, header=FALSE)
trt1 <- read.table('ERR315382.genecounts', sep='\t', as.is=T, header=FALSE)
trt2 <- read.table('ERR315424.genecounts', sep='\t', as.is=T, header=FALSE)


# Combine data sets into a matrix
geneCounts <- data.frame(ctrl1[,2], ctrl2[,2], trt1[,2], trt2[,2])
row.names(geneCounts) = ctrl1[,1]

toRmv <- which(grepl("__", ctrl1[,1]))
geneCounts <- geneCounts[-toRmv,] # ignore the few special lines such as "__no_feature"
(cf. http://htseq.readthedocs.io/en/release_0.9.0/count.html)

condition <- c(rep('control',2), rep('treatment',2))
colnames(geneCounts) <- c('control1','control2','treatment1','treatment2') # use sample
names as column names


y <- DGEList(counts=geneCounts, group=condition)
y <- calcNormFactors(y)
design <- model.matrix(~condition)
y <- estimateDisp(y,design)


# Perform quasi-likelihood F-tests
fit <- glmQLFit(y,design)
qlf <- glmQLFTest(fit, coef=2)
topTags(qlf)


# Perform likelihood ratio tests
fit <- glmFit(y,design)
lrt <- glmLRT(fit,coef=2)
topTags(lrt)
```