From bacterial RNA-seq data to read counts against genomic features

This is a quick guide to a walkthrough that starts from RNA-seq raw reads and an annotated bacterial genome (i.e. a file describing gene/transcript features in the genome) and ends with a table of counts that can be used as input for differential expression analysis.

There are many versions of such walkthroughs and it is worth trying alternatives, especially when new software is out and claiming to do better than the old stuff. This walkthrough is simply one of the possibilities that can be followed, if you have access to our server *icarus*. To try out Rsubread or carry out differential expression using some of the more popular tools, you will a working installation of **R/Rstudio** and the relevant **Bioconductor** packages. This walkthrough does not describe the differential expression analysis part.

Finally, note that recently a different approach is gaining popularity in the analysis of **eukaryotic** RNA-seq data. It is a pseudo-alignment approach that avoids the actual full mapping of reads to the genome and instead checks for compatibility of k-mers in a read with k-mers in known transcripts. Although the method has been developed mostly to deal with issues of carrying out differential expression at the gene level in the case of genes producing multiple transcripts, there is no obvious reason why the method couldn't be applied to bacterial RNA-seq data. In fact, because it is a lot faster than normal mapping methods, it may be useful in the case of very big datasets. However, note that this method does not produce bam files (i.e. files containing mapped reads that can be visualised in a genome browser) and requires an extra processing step for such files to be created. This approach will be covered in a different walkthrough.

IMPORTANT

Do not use this walkthrough blindly. Every command in every script should be checked. You need to ensure you understand what each step does (by looking up the relevant documentation) because a) as program versions change, there could be commands that no longer run or that need modifications and b) I may have made mistakes when writing this guide.

Preliminary note on running jobs on the departmental servers *icarus* or *thoth*

This walkthrough can be followed if you have access to our servers *icarus* or *thoth*. If you don't, it may still be useful as a series of steps that you can follow on another server or even a laptop, if the software has been installed and your memory and hard disk allow you to complete the pipeline.

When running programs on a server, I generally use a bash shell script that will run the same job for multiple samples. BEFORE I run the script that will work on multiple samples, I always run it once on a single sample to make sure it works as intended. There is no queueing system on icarus so the jobs will run sequentially as long as there are free CPUs available. If you run the script directly on the command line, it will hang once you exit the shell or you are logged out, even if you put the job in the background. This is why some of the commands below make use of `nohup` to ensure that the job will stay in the background after the shell has exited, e.g. the command:
`nohup ./run_fastqc.sh ./E-GEOD-69350/ > & run_fastqc.out &`
runs the bash shell script *run_fastqc.sh* using as input the directory E-GEOD-69350 and saving output that would have gone to standard out to the file run_fastqc.out.

If you work on *thoth*, you should start your session by typing: `module use -a /s/software/modules` or alternatively create a file called .modulerc in your HOME directory and put in it the following text:

```
#%Module -*- tcl -*-

module use -a /s/software/modules
```

Step 1: Organise your space!

When carrying out processing and analysis of data, it pays to be super-organised. I tend to have a parent directory for the project and a number of sub-directories, corresponding more or less to the steps required for the processing. You can choose your own directory structure but you should avoid mixing inputs/outputs of different programs (with many samples in a dataset, this can turn out to be very messy). For the description of steps below I assume that the following directories have been created (again, the names are of my own choice but they reflect well the processing steps I do):

- data
- genome
- fastqc
- multiqc
- fqtrim
- trimmomatic
- mapping
- rseqc

Clearly, you could have a different organisation. It doesn't matter HOW you organise your directories but it is important that some meaningful way of sorting your files is employed.

Step 2: Download the raw data

The starting point of workflows is usually raw data. In this walkthrough, the raw data is in the FASTQ format and can be obtained from databases of gene expression data (ArrayExpress, GEO) or general databases of reads (the sequence read archive (SRA), the european nucleotide archive (ENA) etc). Often bulk download is possible; in other cases you may have to ftp or download the files individually (especially if you don't know how to use the site's API to download in bulk).

On a UNIX server, you should be able to obtain files with `wget`. In the following example, I have found the link for this particular sample, under Samples and Data for the study E-GEOD-69350 and I download the fastq file using this link:
`wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR204/006/SRR2043956/SRR2043956.fastq.gz`
This is ok for small sample size datasets (e.g. 3 vs 3 samples) but when more samples are available, you should follow the link to SRA/ENA and use their facilities for bulk download.

Useful links:

- FASTQ format entry in wikipedia: https://en.wikipedia.org/wiki/FASTQ_format
- ArrayExpress at the EBI: https://www.ebi.ac.uk/arrayexpress/
- Gene Expression Omnibus at the NCBI: https://www.ncbi.nlm.nih.gov/geo/
- Sequence read archive: https://www.ncbi.nlm.nih.gov/sra
- European nucleotide archive: https://www.ebi.ac.uk/ena

Step 3: Download the genome annotation

When mapping RNA-seq reads to a genome, you need the genome FASTA file, containing all nucleotides in the order they appear in the genome of interest. It is important to note here that the version of the genome you use for mapping the data must be the version you will use for defining features (such as genes) and the version you will use for visualisation (with genome browsers). If different versions are used, the reads will most likely end up outside the genomic features of interest.

Once mapping of reads has been completed, you will need a genome annotation file (often ending in .gff3 or .gtf), i.e. a file that describes where each feature (gene, transcript, exon etc) starts and ends in the genomic coordinates. What type of file you need is determined by the program you use to do the counting of reads against

genomic features (see steps below for more info). However, for the mapping of reads using traditional mappers, you do not need an annotation file.

You can download genome FASTA files and annotations from Ensembl, the UCSC genome browser, the NCBI etc. I usually put both the genome fasta files and genome annotation files in the *genome* directory. This directory could be a sub-directory of the *data* directory (the data directory is meant to hold all external data that you do not make changes to after downloading).

Useful links:

- Ensembl bacteria main search page: https://bacteria.ensembl.org/index.html
- Ensembl bacteria genomes ftp download: https://bacteria.ensembl.org/info/website/ftp/index.html

Step 4: Check the raw data

Generally, it's a good idea to start with a quick quality assessment of the raw data. This is because data could be unsuitable for further processing, if it's very low in quality or displays biases that are incompatible with the questions being asked (e.g. where the 5' end is severely depleted but we are interested for some reason in the starts of transcripts).

Until recently, we used **fastqc** for quality control, followed by **multiqc** for easy summarisation across multiple samples.

- On *icarus* the latest fastqc version installed is under: `/d/in7/s/fastqc_0.11.3` This is quite likely not the latest version of the software (but we can update it as and when needed).

- On *thoth* you should type the following BEFORE calling the program fastqc on the command line or any script:

```
module load fastqc`
```

You can run fastqc on each sample but it makes more sense to run them all together using a script. Assuming the raw data files are under a directory called *data* that can be found in the parent directory, you can run:

```
nohup ./run_fastqc.sh ../data/E-GEOD-69350/ > & run_fastqc.out &
```

where the run_fastqc.sh script in my case contains the following (note: this is supposed to run fastqc on icarus and uses the installation under /d/in7/s ; on thoth, use simply "fastqc"):

```
#!/bin/bash
# Runs FASTQC on all fastq.gz files found in given directory
```

```
# Run as:
# ./run_fastqc dir_of_fastq_files

FASTQC_DIR="/d/in7/s/fastqc_0.11.3/FastQC/fastqc"

echo "Running FASTQC using executable: $FASTQC_DIR"

for fastq_file in $1/*.fastq.gz;
do
  echo "Fastq file: $fastq_file"
  $FASTQC_DIR $fastq_file -o .
done
```

Obviously, you will need to edit the script to suit your setup and needs.

NOTE: The *fastqc* program is not aware of paired reads and will treat forward and reverse reads as independent (both files will need to be processed).

Once *fastqc* is run, you can look at a summary of all output from all samples using the software *multiqc*. To run multiqc on icarus you need to do the following first (on the unix shell):

```
module use -s /s/mm/modules
module load python/v2
```

On *thoth*, do instead:

```
module load python/v3
```

Then, simply call the multiqc executive binary (`multiqc`) pointing it to the directory with all the *fastqc* output (here I assumed that this was in a directory called fastqc that is two directories up from the current directory - your setup may be different!):

```
multiqc ../../fastqc/E-GEOD-69350/
```

Assuming that the data looks reasonable and there are no outlier samples that should be removed, you can move on to the next step. What is reasonable data is a big question and cannot be answered easily. You can consult the fastqc documentation for some examples of what bad data looks like but it is likely that any RNA-seq dataset will fail some of the tests and this does not mean that you cannot use it.

Useful links:

- The fastqc program and documentation https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
- The multiqc program https://multiqc.info/

Step 5: Pre-process the raw data prior to mapping

In this step you need to pre-process the fastq files so that you get rid of adapters (bits of them sometimes are still attached to the reads) and trim polyA tails (these are

naturally present in eukaryotic transcripts but are also there in RNA-seq data of bacteria because the technology requires the transcripts to be artificially poly-adenylated). This step is required because without it, most reads that contain adapters or poly(A) tails would be thrown away as they would containg too many nucleotides not mapping to the reference genome. Quality control with fastqc generally reveals issues with adapters/poly(A) tails and can show what percentage of the data is affected by such issues.

In the past, we have generally used *Trimmomatic* for trimming adapters and *fqtrim* for trimming poly(A) tails. Currently, I would suggest the use of *BBduk* (not explained here yet) or *fastp* (see below) or *cutadapt* as used in the Systems Biology practicals (if you've taken those).

On *icarus*, we have a version of fqtrim here: `/d/in7/s/fqtrim/fqtrim-0.94/fqtrim` and a version of Trimmomatic here `/d/in7/s/trimmomatic/Trimmomatic-0.32/`
On *thoth* only Trimmomatic is currently installed, so run:

```
module load trimmomatic
```

To run *fastp* on either server run:

```
/d/in7/s/fastp/fastp
```

To run *cutadapt* on thoth do:

```
module load python/v3
```

Here is a bash script that runs Trimmomatic on single-end reads:

```
#!/bin/bash
# Runs Trimmomatic in SE mode for all sample names given as arguments
# Run as:
# ./run_trimmomatic.sh directory_of_samples list_of_samples

timestamp=`date "+%Y%m%d-%H%M%S"`
logfile="run_$timestamp.log"
exec > $logfile 2>&1  #all output will be logged to logfile

TRIM_EXEC="/d/in7/s/trimmomatic/Trimmomatic-0.32/trimmomatic-0.32.jar"
DIR=$1
shift

echo "Running Trimmomatic using executable: $TRIM_EXEC"

for sample in "$@";
do
  echo "Sample= $sample"
  java -jar $TRIM_EXEC SE -threads 8 -phred33 \
       -trimlog "$sample"_trim_report.txt \
        "$DIR$sample".fastq.gz \
       "$sample"_trimmed.fastq.gz \
       ILLUMINACLIP:/d/in7/s/trimmomatic/Trimmomatic-0.32/adapters/TruSeq2and3-
SE.fa:2:30:10 \
       LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36

  gzip "$sample"_trim_report.txt
```

```
done
```

To change the script for paired-end reads replace the java command as follows:

```
java -jar $TRIM_EXEC PE -threads 8 -phred33 \
       -trimlog "$sample"_trim_report.txt \
        "$DIR$sample"_1.fastq.gz "$DIR$sample"_2.fastq.gz \
       "$sample"_R1.trimmed_paired.fastq.gz "$sample"_R1.trimmed_unpaired.fastq.gz
\
       "$sample"_R2.trimmed_paired.fastq.gz "$sample"_R2.trimmed_unpaired.fastq.gz
\
       ILLUMINACLIP:/d/in7/s/trimmomatic/Trimmomatic-0.32/adapters/TruSeq2and3-
PE.fa:2:30:10 \
       LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

If you run fqtrim following the removal of adapters, you will need to use as input the output fastq files produced by Trimmomatic. An example script dealing with single-end reads might look like this:

```
#!/bin/bash
# Runs FQTRIM in SE mode for all sample names given as arguments
#
# Run as:
# ./run_fqtrim.sh directory_of_samples list_of_samples

timestamp=`date "+%Y%m%d-%H%M%S"`
logfile="run_$timestamp.log"
exec > $logfile 2>&1  #all output will be logged to logfile

FQTRIM_EXEC="/d/in7/s/fqtrim/fqtrim-0.94/fqtrim"
dir=$1
shift

echo "Running FQTRIM using executable: $FQTRIM_EXEC"

outSuffix="fqtrimmed.fastq.gz"   #this is added to the end of all output files
minLength=25 #min length threshold for reads after trimming
minPolyLength=3 #at least that many As/Ts are required for trimming
numProc=8 #num or CPUs to use

suffix="trimmed.fastq.gz"

for sample in "$@";
do
  echo "***Sample= $sample"

  echo "Trimming single-end reads  for sample $sample ..."
  $FQTRIM_EXEC -o $outSuffix -l $minLength -y $minPolyLength -p $numProc \
               -r "$sample".fqtrim_report.txt \
               "$dir/$sample"_"$suffix"

done
echo "All done!"
```

If paired-end reads are used, the script could be modified as follows:

```
#!/bin/bash
# Runs FQTRIM in PE mode for all sample names given as arguments
```

```
#
# Run as:
# ./run_fqtrim.sh directory_of_samples list_of_samples

timestamp=`date "+%Y%m%d-%H%M%S"`
logfile="run_$timestamp.log"
exec > $logfile 2>&1  #all output will be logged to logfile

FQTRIM_EXEC="/d/in7/s/fqtrim/fqtrim-0.94/fqtrim"
dir=$1
shift

echo "Running FQTRIM using executable: $FQTRIM_EXEC"

outSuffix="fqtrimmed.fastq.gz"   #this is added to the end of all output files
minLength=25 #min length threshold for reads after trimming
minPolyLength=3 #at least that many As/Ts are required for trimming
numProc=8 #num or CPUs to use

pairedReadsSuffix1="_R1.trimmed_paired.fastq.gz"
pairedReadsSuffix2="_R2.trimmed_paired.fastq.gz"

#First, we deal with paired-end unaligned reads
for sample in "$@";
do
  echo "***Sample= $sample"
  echo "Trimming paired reads for sample $sample ..."

  $FQTRIM_EXEC -o $outSuffix -l $minLength -y $minPolyLength -p $numProc \
               -r "$sample".paired_fqtrim_report.txt \
               "$dir/$sample$pairedReadsSuffix1","$dir/$sample$pairedReadsSuffix2"

done


echo "All done!"
```

An alternative approach

Very recently, a program called *fastp* was published. This is supposed to be the best and fastest program available currently for pre-processing of fastq data. A typical script to run fastp on all samples ending in .fastq.gz in a directory given on the command line might look like this (NOTE: this uses a file of adapters for Nextera taken from the Illumina website; other common adapters can be found on similar sites or you can copy TruSeq ones for single-end data here: `/d/in7/s/trimmomatic/Trimmomatic-0.32/adapters/TruSeq2and3-SE.fa` or paired-end data here: `/d/in7/s/trimmomatic/Trimmomatic-0.32/adapters/TruSeq2and3-PE.fa`).

```
#!/bin/bash
# Runs fastp in SE mode for all .fastq.gz files in a directory
# Run as:
# ./run_fastp.sh directory_of_samples

timestamp=`date "+%Y%m%d-%H%M%S"`
logfile="run_$timestamp.log"
exec > $logfile 2>&1  #all output will be logged to logfile
```

```
FASTP_EXEC="/d/in7/s/fastp/fastp"
DIR=$1
ADAPTER_FILE="/d/in16/u/ubcg71a/research/ASD/data/adapters/adapters.fa"

echo "Running fastp using executable: $FASTP_EXEC"

for file in `ls $DIR/*.fastq.gz`;
do
  sample=${file/$DIR\//}
  sample=${sample/.fastq.gz/}
  echo "Sample= $sample"
  $FASTP_EXEC -i  "$DIR/$sample".fastq.gz  \
         -o "$sample"_trimmed.fastq.gz \
         --adapter_fasta $ADAPTER_FILE \
         -l 30 --trim_poly_x \
         -h "$sample"_fastp.html -j "$sample"_fastp.json
done
```

Useful links

- Trimmomatic http://www.usadellab.org/cms/?page=trimmomatic
- fqtrim https://ccb.jhu.edu/software/fqtrim/
- The fastp paper: https://academic.oup.com/bioinformatics/article/34/17/i884/5093234

Step 6: Mapping of trimmed reads to the reference genome

Once the reads have been trimmed, we map them to the genome. This generally requires an indexed version of the genome you want to map to, which you have to create following the guidelines of the mapping program. In the past we used *TopHat* which then was replaced by *Hisat2* for eukaryotes. *STAR* is also a popular alternative. I've used ***bowtie2*** and ***bwa*** for bacteria but I'd be more keen now to use the ***Rsubread*** software available from Bioconductor in R, as sometimes it is advantageous to have all processing done in R, if possible. I was trying to find out if people use it for bacteria but most people still use bwa or bowtie2 or star. However, I don't see any reason why Rsubread's *align* function shouldn't be used for bacteria. It has the advantage that the mapping step can be done within R, together with the rest of your pipeline.

On *icarus* we have (slightly old) versions of hisat2 and bwa installed here (if you wish to try them):

`/d/in7/s/HISAT2_v2.1`
and here:

`/d/in7/s/bwa`

On *thoth*, do:

```
module load hisat2
```
or

```
module load bwa
```
You will almost certainly need to do some sorting/indexing of the mapped reads. For this, we have samtools available. On *icarus* under:

```
/d/in7/s/samtools/samtools-1.3.1/samtools
```
and on *thoth* we have a newer version that can be accessed with:

```
module load samtools
```

Mapping with *bwa*

Here, I show an example of how to use *bwa*. Like all mapping programs, bwa needs to index the genome first. This is a simple call to bwa with the index command:

```
bwa index NC_003888.fa
```
where NC_003888.fa is a genome fasta file downloaded from NCBI and saved in the *genome* directory (obviously, you need to download your genome of interest). Once the genome is indexed, we can use bwa-aln (used here because the reads were shorter than 70 nucleotides) in a bash script as follows (don't forget bwa-aln should be used on the **preprocessed** fastq files, not the original fastq). Again this script assumes you are running on *icarus*; on *thoth* you need to change the way you are calling the executables fo bwa, samtools etc.

```
#!/bin/bash

# Runs bwa in single end mode

# Run as:
# run_bwa.sh directory_of_fastq_files samples

timestamp=`date "+%Y%m%d-%H%M%S"`
logfile="run_$timestamp.log"
exec > $logfile 2>&1  #all output will be logged to logfile

dir=$1
shift

#set location of executables
BWA_EXEC="/d/in7/s/bwa-0.7.17/bwa"
SAMTOOLS_EXEC="/d/in7/s/samtools/samtools-1.3.1/samtools"

#set parameters
genomeFile="/d/in15/u/ubcg71a/research/alina/genome/NC_000913.2.fa" #index files
should be there too!
numProc=8

#extension for fastq files
suffix="_trimmed.fqtrimmed.fastq"
```

```
for sample in "$@";
do
   echo "Running bwa on sample $sample (single-end mode)..."

   unpairedFile="$dir$sample$suffix".gz
   if [ -f $unpairedFile ]
   then
      gzip -d $unpairedFile
      unpairedFile=$dir$sample$suffix
   else
      unpairedFile=$dir$sample$suffix
      if [ ! -f $unpairedFile ]
      then
         echo "File not found: $unpairedFile"
         exit $?
      fi
   fi
   tmpBinary="$sample"_sa.sai
   tmpSam="$sample"_se.sam
   tmpBam="$sample"_se.bam
   finalSortedBam="$sample"_sorted.bam

   #align
   $BWA_EXEC aln -t $numProc $genomeFile $unpairedFile > $tmpBinary
   #create sam file
   $BWA_EXEC samse $genomeFile $tmpBinary $unpairedFile > $tmpSam
   #create bam file
   $SAMTOOLS_EXEC view $tmpSam -Sbo $tmpBam
   $SAMTOOLS_EXEC sort $tmpBam -o $finalSortedBam
   $SAMTOOLS_EXEC index $finalSortedBam

   #cleanup
   /bin/rm $tmpSam $tmpBam
   /bin/rm $tmpBinary
   gzip -9 $unpairedFile

done


echo "All done!"
```

For paired-end reads we need to modify the script:

```
#!/bin/bash

# Runs bwa in paired-end  mode

# Run as:
# run_bwa.sh directory_of_fastq_files samples

timestamp=`date "+%Y%m%d-%H%M%S"`
logfile="run_$timestamp.log"
exec > $logfile 2>&1  #all output will be logged to logfile

dir=$1
shift

#set location of executables
```

```
BWA_EXEC="/d/in7/s/bwa-0.7.17/bwa"
SAMTOOLS_EXEC="/d/in7/s/samtools/samtools-1.3.1/samtools"

#set parameters
genomeFile="/d/in15/u/ubcg71a/research/alina/genome/Mycobacterium_smegmatis_str_mc
2_155_gca_000767605.ASM76760v1.dna.chromosome.Chromosome.fa " #index files should
be there too!
numProc=8

#extension for fastq files
suffix1="_R1.trimmed_paired.fqtrimmed.fastq"
suffix2="_R2.trimmed_paired.fqtrimmed.fastq"

for sample in "$@";
do
   echo "Running bwa on sample $sample (paired-end mode)..."

   pairedFile1="$dir$sample$suffix1".gz
   if [ -f $pairedFile1 ]
   then
      gzip -d $pairedFile1
      pairedFile1=$dir$sample$suffix1
   else
      pairedFile1=$dir$sample$suffix1
      if [ ! -f $pairedFile1 ]
      then
         echo "File not found: $pairedFile1"
         exit $?
      fi
   fi
   pairedFile2="$dir$sample$suffix2".gz
   if [ -f $pairedFile2 ]
   then
      gzip -d $pairedFile2
      pairedFile2=$dir$sample$suffix2
   else
      pairedFile2=$dir$sample$suffix2
      if [ ! -f $pairedFile2 ]
      then
         echo "File not found: $pairedFile2"
         exit $?
      fi
   fi

   tmpBinary1="$sample"_1_sa.sai
   tmpBinary2="$sample"_2_sa.sai
   tmpSam="$sample"_pe.sam
   tmpBam="$sample"_pe.bam
   finalSortedBam="$sample"_sorted.bam

   #align
   $BWA_EXEC aln -t $numProc $genomeFile $pairedFile1 > $tmpBinary1
   $BWA_EXEC aln -t $numProc $genomeFile $pairedFile2 > $tmpBinary2

   #create sam file
   $BWA_EXEC sampe $genomeFile $tmpBinary1 $tmpBinary2 $pairedFile1 $pairedFile2 >
$tmpSam
   #create bam file
   $SAMTOOLS_EXEC view $tmpSam -Sbo $tmpBam
```

```
    $SAMTOOLS_EXEC sort $tmpBam -o $finalSortedBam
    $SAMTOOLS_EXEC index $finalSortedBam

    #cleanup
    /bin/rm $tmpSam $tmpBam
    /bin/rm $tmpBinary1 $tmpBinary2
    #gzip -9 $pairedFile1 $pairedFile2

done


echo "All done!"
```

Useful links

- bwa documentation http://bio-bwa.sourceforge.net/bwa.shtml
- samtools documentation http://www.htslib.org/doc/samtools.html

Step 7: Check the mapping output for quality

Following mapping, you can do further quality control of the samples by checking the mapping results. This can be done with various scripts available from *RSEQC*. They are all python scripts and you can run them on *icarus* after you source this script:

```
source /s/mm/epd/setup.csh
```

On *thoth*, you need to do first:

```
module load python/v3
```

and then you can call each script directly by its name.

The most interesting scripts are probably the ones that show you mapping statistics (`bam_stat.py`), calculation of TIN values (a measure of RNA degradation; `tin.py`) and calculation of gene body coverage (which shows biases in the coverage of genes, usually at the 5′ or 3′ ends; `geneBody_coverage.py`).

Useful links

- RSEQC website and documentation: http://rseqc.sourceforge.net/

Step 8: Counting reads against genomic features

For differential expression, we need a table of counts against genomic features. These features are usually genes, but they could be transcripts, exons or any other type of feature defined in an annotation file. In our own pipeline called baerHunter,

we update the annotation file of bacterial genomes with non-coding features and counting can be done against those too (baerHunter will be described elsewhere).

For counting, we used to use *Htseq-count*. To use it on *icarus* you would need to do:

```
source /s/mm/epd/setup.csh
```

An example script for running htseq_count is given below:

```
#!/bin/bash
# Runs HTSEQ-COUNT for all sample names given as arguments
#
# Run as:
# ./run_htseq_count.sh directory_of_samples list_of_samples

timestamp=`date "+%Y%m%d-%H%M%S"`
logfile="run_$timestamp.log"
exec > $logfile 2>&1  #all output will be logged to logfile

#on icarus need to source: source /s/mm/epd/setup.csh
SAMTOOLS_EXEC="/d/in7/s/samtools/samtools-1.3.1/samtools"
dir=$1
shift

echo "Running htseq-count... "

bamSuffix=".paired_sorted.bam"
outSuffix="_paired_before_polyA_rem_htseq.txt"
gtfFile="/d/in9/u/ubcg71a/research/filipe_neurexin/genome/Caenorhabditis_elegans.W
Bcel235.90.gtf"

for sample in "$@";
do
  echo "***Sample= $sample"

  #sort by name to avoid overflow
  echo "Sorting bam by name...."
  tempBam="./"$sample"_n"$bamSuffix
  $SAMTOOLS_EXEC sort -n $dir$sample$bamSuffix -o $tempBam

  #only count pairs that are properly mapped
  echo "Counting..."
  $SAMTOOLS_EXEC view -f 0x0002 -h $tempBam | \
                htseq-count -s reverse \
                -t exon -i gene_id - $gtfFile >& $sample$outSuffix

  grep "WBGene" $sample$outSuffix > counts_$sample$outSuffix
  grep -v "WBGene" $sample$outSuffix > htseq_summary_$sample$outSuffix
  /bin/rm $sample$outSuffix
  /bin/rm $tempBam

done

echo "All done!"
```

However, htseq-count is painfully slow compared to *featureCounts* in *Rsubread* and in addition it doesn't run within R so I think I'd always go for featureCounts now.

Counting with Rsubread

Rsubread is an R package for dealing with NGS data. The Rsubread documentation is very reasonable and you should consult it for what options to use. You should always be aware of whether the data is **paired-end** or **single-end** and also whether it is **stranded** or not as these two options usually need to be set for all programs you use that handle these data.

An example command to count against a set of genomic features might look like this (see the Rsubread documentation for the meaning of each parameter):

```
fc <- featureCounts(bam_files, annot.ext = annotation_file, isGTFAnnotationFile = TRUE, GTF.featureType = feature_type, GTF.attrType = attribute_type,chrAliases = chromosome_alias_file, strandSpecific = strand_specific, isPairedEnd = paired_end)
```

where all parameters have already been set before calling featureCounts, e.g. bam_files is in this case a vector containing the names of files with BAM data (i.e. reads mapped to genomic coordinates) - each file representing one sample.

Note that Rsubread can also map your reads, not just count them, so you can perform all steps following pre-processing of reads within R. You can use the `align()` method to map reads to a reference genome.

Notes on mapping RNA-seq data:

- You may find a very large number of duplicates in the mapped reads (as well as the original fastq files). These are usually PCR artefacts but they can be true reads because of biases in the fragmentation process and sequencing. They are often not removed from RNA-seq but some people do analysis both with and without them to see the effect.
- If mapping rates are low you need to worry. You may have given the wrong strandedness option or something else may have gone wrong… With eukaryotes we often get over 80% mapping. I would think similar rates should be expected in bacteria. You can have contaminations too. One thing you can do is **blast** individual random reads that have not mapped to see where they are coming from.

Useful links

- htseq-count https://htseq.readthedocs.io/en/master/count.html
- Rsubread documentation https://bioconductor.org/packages/release/bioc/html/Rsubread.html