# CSC Spring School in Bioinformatics 2014, Day 2

1.4.2014

**Exercise 1:**

An example batch job script. Note that if you ran *prinseq-lite* command as specified in the original exercise sheet, your input file would be “hESC_good.fastq.fastq”.

If using this as template, remember that the “--reservation=bioinfo” was only for the course and won’t normally work.

Tophat has argument “-p” that tells the program how many threads to use. We use here variable $SLURM_CPUS_PER_TASK . We could use “-p 4” instead, but then we would have to remember to change that if we change “--cpus-per-task” parameter in the batch job script.

```
#!/bin/bash -l
#SBATCH -J TopHat
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 24:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=4000
#SBATCH --reservation=bioinfo
#

module load biokit

tophat2 -p $SLURM_CPUS_PER_TASK --transcriptome-index hg19.ti -i 70 --no-novel-juncs \
hg19 hESC_good.fastq
```

## Exercise 2

An example batch job script for an array job.

```
#!/bin/bash -l

#SBATCH -J TopHat

#SBATCH -o output_%j.txt

#SBATCH -e errors_%j.txt

#SBATCH -t 24:00:00

#SBATCH -n 1

#SBATCH --nodes=1

#SBATCH --cpus-per-task=4

#SBATCH --mem-per-cpu=4000

#SBATCH --reservation=bioinfo

#SBATCH --array=1-10

#


mkdir job_"$SLURM_ARRAY_TASK_ID"

cp sample_"$SLURM_ARRAY_TASK_ID".fastq job_"$SLURM_ARRAY_TASK_ID"

cd job_"$SLURM_ARRAY_TASK_ID"

prinseq-lite.pl -trim_qual_right 20 -fastq sample_"$SLURRM_ARRAY_TASK_ID".fastq \

-out_good  sample_"$SLURM_ARRAY_TASK_ID"_good


tophat2 -p $SLURM_CPUS_PER_TASK --transcriptome-index ../hg19.ti -i 70 \

--no-novel-juncs ../hg19 sample_"$SLURM_ARRAY_TASK_ID"_good.fastq
```

## Exercise 3, extra task 1

There are many ways to solve this problem. I chose to make two temporary files: one with all the accession numbers from the original fasta file and the other with accession numbers for the sequences that found a hit.

```
awk '{print $1}' pb_blast_results | grep -v "#" | sort |uniq > found.list

grep ">" R.fasta | awk '{print $1}'|cut -c 2- > all.list
```

You could then use *grep*:

```
grep -w -v -f hits.list all.list
```

We specify –w = whole word matches only, -v = print those lines that do NOT match and –f = inputs are files. (See grep man pages for details.)

Another solution is to use *diff* (see diff man pages for details):

```
diff hits.list all.list |grep ">" |awk '{print $2}'
```

## Exercise 3, extra task 2

Here is a one-liner that does steps 1-6:

```
grep -v "#" hsa.gff3 | grep -v "miRNA_primary_transcript" | cut -c 4- | \

cut -d ";" -f 1 | sed s/ID=/'gene id "'/ | awk '{print $0"\""}' | sort \

-k1n,1 -k4n,4 > hsa.edited
```

Again, there are many ways to do this. Here I chose to *awk*, *sed* and *cut* to demonstrate their use.

For step 7 you could do:

```
grep -w "^2" hsa.edited > hsa_chr2
```

"^" = beginning of line, so "^2" searches for lines starting with 2. Again, we use the –w parameter so we only get lines for chromosome 2 and not also those for 20, 21 etc.

**Exercise 4**

Again, many ways of doing this. I chose to use ***while loops***.

Sample script:

```
#!/bin/bash

x=7
y=1

while [ $y -le 10 ] ; do
        echo $(( $y * $x ))
        let y=y+1
done
```

**Extra task 1**

Here we used the variable $1 to get the command line argument.

Sample script

```
#!/bin/bash

x=$1
y=1

while [ $y -le 10 ] ; do
        echo $(( $y * $x ))
        let y=y+1
done
```

**Extra task 3**

Sample script

```
#!/bin/bash


x=1

y=1


while [ $x -le 10 ] ; do

        while [ $y -le 10 ] ; do

                echo -n $(( $x * $y ))$'\t'

                let y=y+1

        done

        echo

        y=1

        let x=x+1

done
```

**Exercise 5**

Here we use a ***for loop*** with *cat* command to go through a file line by line.

For demonstration purposes I have used two different ways to identify the variable (lines 4-5): “$line”.entret and ${line}.entret. The choice is largely a matter of taste. The {} notation may make the code easier to read, especially since double quotes are also used for strings etc.

We use *grep* and *awk* again to parse out the necessary data from the files.

```
#!/bin/bash

for line in $(cat mcrA.list); do

    entret dbfetch:embl:$line $line.entret -filter

    organism=$(grep "/organism="  "$line".entret | awk -F "=" '{print  $2}')

    strain=$(grep "/strain="  ${line}.entret | awk -F "=" '{print  $2}')

    rm -f ${line}.entret

    echo $line";"$organism";"$strain

done
```