CSC Spring School in Bioinformatics 2014, Day 2

1.4.2014

A note on typography:

Some command lines are too long to fit a line in printed form. These are indicated by a backslash "\" at the end of line. It should not be included when typing in the command.

For example

```
example command \
continues \
and continues
```

Should be typed in as:

```
example command continues and continues
```

Exercise 1: Running simple analysis in Taito

Get files *hESC.fastq* and *hg19.tgz* from IDA

```
iget hESC.fastq
iget hg19.tgz
```

Unpack the genome index

```
tar zxf hg19.tgz
```

Setup the bioinformatics environment with command:

```
module load biokit
```

(Note, that there are several bioinformatics tools in Taito that have their own set up command or which don't need a set up command. You can check the software specific information from the bioscience program list http://research.csc.fi/bioscience_programs.)

Next run *prinseq* analysis for *hESC.fastq* dataset interactively:

```
prinseq-lite.pl -stats_all -fastq hESC.fastq
prinseq-lite.pl -trim_qual_right 20 -fastq hESC.fastq -out_good hESC_good
```

Create a batch job script that runs following TopHat2-command:

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

Open a new file called tophat2_run.sh with a text editor. You can use for example *nano* editor with command:

```
nano tophat2_run.sh
```

Then paste the sample tophat2 batch job script from the Tophat2 page of CSC (http://research.csc.fi/-/tophat) to this empty text file, and edit the *TopHat2* command to the correct form.

We have a resource reservation for the course on Taito. To use it add this line to your batch job script. (Note that you should only use this for this course, not for your own jobs.)

#SBATCH --reservation=bioinfo

Then submit the job with command:

```
sbatch tophat2_run.sh
```

After submission you can follow the status of your job with commands:

```
squeue -l
squeue -l -u trng<XX>
```

After the job has finished you check what resources it used with

```
sacct
```

To see details of a single job you can use

```
sacct –l –j <jobid>
```

sacct lists a rather long list of values, many of which are of interest mainly to sysadmins. To get a more manageable output you can use the –o option to define which fields you want to see.

```
sacct -o jobid,jobname,maxrss,maxvmsize,state,elapsed -j <jobid>
```

Here we have elected to show jobid (JobID), jobname (JobName), maximum used memory (MaxRSS), maximum used virtual memory (MaxVMSize), state of the job (State) and elapsed time (Elapsed). These can be helpful when we decide on the resource allocation parameters for our next similar job.

For details on the different available fields and other options see

```
man sacct
```

More information about running batch jobs in Taito:

http://research.csc.fi/taito-batch-jobs

Exercise 2: Running an array job

(http://research.csc.fi/taito-array-jobs)

In this exercise we will run 10 similar analysis using *array batch job* approach.

First download and unpack the input data set from IDA:

```
iget  samples1-10.tar.gz
gunzip samples1-10.tar.gz
tar xvf  samples1-10.tar
```

Now you should have ten new fastq files in your $WRKDIR directory.

(sample_1.fastq, sample_2.fastq, ..., sample_10.fastq)

Next modify the *tophat2_run.sh* batch job file so that it submits 10 jobs as a single job array. To do this add following line to the batch job definitions:

#SBATCH --array=1-10

In the actual command script, use the subjob specific environment variable $SLURM_ARRAY_TASK_ID to do following steps for each of the sample files:

1. create a separate directory for the input dataset

```
mkdir job_"$SLURM_ARRAY_TASK_ID"
```

2. copy the dataset to this new directory

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

3. run the *Prinseq* and *TopHat2* analysis for the dataset

```
cd job_"$SLURM_ARRAY_TASK_ID"

prinseq-lite.pl -trim_qual_right 20 –fastq sample_"$SLURM_ARRAY_TASK_ID".fastq \
-out_good sample_"$SLURM_ARRAY_TASK_ID"_good.fastq

tophat2 -p $SLURM_CPUS_PER_TASK --transcriptome-index ../hg19.ti -i 70 \
--no-novel-juncs ../hg19 sample_"$SLURM_ARRAY_TASK_ID"_good.fastq
```

And then submit the job with command *sbatch.*

Use *squeue* to check what the job looks like.

Extra exercise: Installing your own application

In this example we install MCL Markov cluster algorithm program to users own $USERAPPL directory in Taito.

Move to your $USERAPPL directory and create there a new directory called *mcl*

```
cd $USERAPPL
mkdir mcl
```

Go to the just created **mcl** directory and download the installation package with wget command

```
cd mcl
wget http://www.micans.org/mcl/src/mcl-latest.tar.gz
```

In this case the installation package is a tar-archive file that has been compressed with gzip program. You can unpack this file with commands

```
gunzip mcl-latest.tar.gz
tar xvf mcl-latest.tar
```

After unpacking, the **ls** command shows that a new directory called *mcl-12-068* has been created to your *mcl* directory. This directory contains the actual installation files and documentation of the software. Create a new empty directory called *version-12-068* to the mcl directory.

```
ls
mkdir version-12-068
```

After this go to the *mcl-12-068* directory and study its' content

```
cd
ls -l
```

Installation packages contain often a short installation instructions. Typically this instruction file is called as *INSTALL* or *README*. In this case you should read the *INSTALL* file to find out how the installation should be done.

```
less INSTALL
```

Many open source software tools are installed using following three steps:

1. Building up the so called *Makefile* with a **./configur**e command.
2. Running **make** command that compiles the source code according to the instructions in the *Makefile*
3. Installing the compiled executables with command **make install**

Normally the installation packages assume that the user has permissions to install the software to the locations where the standard linux commands and programs normally get installed. However, at CSC this is not the case. You can install software only to your own disk areas. Often you can use option **--prefix=*/path/*** to tell to the configure command, where to the program should be installed. In this case we wish to install the software to the *version-12-068* directory in you $USERAPPL area. Thus you must use following ./configure command:

```
./configure --prefix=$USERAPPL/mcl/version-12-068
```

The configure command checks that all the compilers and libraries, that the software needs, are available. It is not uncommon, that *./configure* reports about missing libraries or incorrect compilation options. In those cases you can check if the missing library or program can be taken in

use with the module system. CSC environment has several compiler and program versions available. In some cases you may for example need to use certain C-compiler or python version in order to install the software. If you still fail with the installation, ask help from the HelpDesk of CSC.

In the case of *mcl,* the *./configure* script runs without error messages when you use GNU-compilers. The GNU compilers are set up with command:

```
module switch intel/13.1.0 gcc
```

Next need to compile and install the software with commands:

```
make
make install
```

If *make* and *make install* commands don't give any error messages, you have successfully installed your software. Typically the executables, i.e. the compiled programs that can be launched, are stored to a sub directory called *bin*. In this case the bin directory is created to subdirectory *$USERAPPL/mcl/version-12-068.*

Running command

```
ls $USERAPPL/mcl/version-12-068/bin
```

now shows the programs you have installed:

clm clmformat mcl mclcm mclpipeline mcx mcxarray mcxassemble mcxdump mcxi mcxload mcxmap mcxrand mcxsubs

The name of the directory that contains the executables may vary between different software. In any case, to be able to use the programs you must tell the location of your own executables to the command shell. This can be done by adding the directory path of you executables to the **$PATH** environment variable. In this case we add path "**${USERAPPL}/mcl/version-12-068/bin**" to the $PATH variable. This is done with command:

```
export PATH=${PATH}\:${USERAPPL}/mcl/version-12-068/bin
```

Note that the first PATH word in the command above is without the dollar sign. Now you can launch the program you have installed. For example

```
mcl -h
```

Remember that also in the future, when you log in to CSC, the PATH variable must be set u before you can use **mcl** command. Also in the batch job files you need to run the correct *export **PATH*** command above before executing the program you have installed yourself.

If the software you have installed works correctly, you can remove the installation package and temporary directories that were used during the compilation. In this case we could remove the *mcl-latest.tar* file and the directory *mcl-09-308/*

```
cd $USERAPPL/mcl
rm mcl-latest.tar
rm -rf mcl-09-308/
```

Exercise 3: Working with data columns

We will first submit a BLAST search using the *pb blast* in Taito. We'll be using our own sequence as the database.

First move to the $WRKDIR and to the sub-directory where you have the *Pseudomonas aeruginosa* genome (NC_009656.fna) available (See Day 1, Exercise 1). Copy the query sequence set R.fasta into the same directory.

```
iget R.fasta
```

You should be able to submit the BLAST job with commands:

```
module load biokit
pb blastn -query R.fasta -dbnuc NC_009656.fna -out pb_blast_results -outfmt 7
```

By running command:

```
head -50 pb_blast_results
```

You can see that the result file contains columns, where the query sequences are in the first column and the hit sequences in the second column. You can now check, how many hits each query sequence got with command:

```
awk '{print $1}' pb_blast_results | grep -v "#" | sort | uniq -c | sort -k1n
```

Study how this command works by executing it step by step:

```
awk '{print $1}' pb_blast_results
awk '{print $1}' pb_blast_results | grep -v "#"
awk '{print $1}' pb_blast_results | grep -v "#" | sort
awk '{print $1}' pb_blast_results | grep -v "#" | sort | uniq -c
awk '{print $1}' pb_blast_results | grep -v "#" | sort | uniq -c | sort -k1n
```

You can check the number of query sequences in the input file with command:

```
grep -c ">" R.fasta
```

Alternative ways to count number of sequences with EMBOSS:

```
module load emboss
seqcount R.fasta
infoseq_summary R.fasta
```

If you compare that to the number query sequences in the result file:

```
awk '{print $1}' pb_blast_results | grep -v "#" | sort | uniq -c | wc -l
```

You can notice that the command used above shows no information for those query sequences that did not get any matches.

Extra task:

Use linux commands to sort out those query sequences that did not produce any hits. There is no single right way to do this task. You probably need several commands and temporary files to get the result. You can also try linux scripting.

Extra task 2:

Often it is necessary to change files slightly to use them in different analysis programs. This exercise simulates some typical changes you need to do.

Get file **ftp://mirbase.org/pub/mirbase/CURRENT/genomes/hsa.gff3**

```
wget ftp://mirbase.org/pub/mirbase/CURRENT/genomes/hsa.gff3
```

1. Remove comment lines (lines starting with #)
2. Remove all lines that include tag 'miRNA_primary_transcript'
3. Change chromosome names from format chr1, chr2, .. to format 1, 2,...
4. The 9. column is now format 'ID=MI0006363_1;accession_number=MI0006363;Name=hsa-mir-1302-2'. Change it to format 'gene_id "MI0006363_1"' You can omit the accession_number and Name entries.
5. Sort the file by chromosome and by miRNA start position (4. column). Make sure to sort the chromosomes in numerical order, not in alphabetical (i.e 1,2,3... not 1,10,11..)
6. Output result to a file
7. Make another file that only has entries from chromosome 2

It's possible to do the above in single command line, but you can use temporary files if you wish.

Exercise 4: Scripting

Write a script that has variable x and calculates the multiplication table from 1 to 10 for that variable. Use loop structures.

e.g for x=7 you should get

```
7
14
21
28
35
42
49
56
63
70
```

Hint: You can use command like: `echo $(( $y * $x ))` to calculate and output the multiplications.

Extra task 1.

Bash has a special variable $1, $2…$N to store command line arguments. Modify your script so that it takes a number from command line:

```
./mt_script 7
```

and prints out the multiplication table as above.

Extra task 2.

Write a script that uses nested loops to calculate all multiplication tables 1-10 and print them out as a table like this:

```
1       2       3       4       5       6       7       8       9       10
2       4       6       8       10      12      14      16      18      20
3       6       9       12      15      18      21      24      27      30
4       8       12      16      20      24      28      32      36      40
5       10      15      20      25      30      35      40      45      50
6       12      18      24      30      36      42      48      54      60
7       14      21      28      35      42      49      56      63      70
8       16      24      32      40      48      56      64      72      80
9       18      27      36      45      54      63      72      81      90
10      20      30      40      50      60      70      80      90      100
```

Hint: You can use command like: `echo -n $(( $y * $x ))$'\t'` to calculate and output the multiplications. (Option –n means no newline after echo and $'\t' outputs a tab character.)

Exercise 5: Scripting continued (a bit more complex this time)

Get file mcrA.list from IDA. It contains a list of EMBL accession numbers, one per line.

```
iget mcrA.list
```

Write a script that goes through the list line by line, and for each accession number:

1. Uses the EMBOSS *entret* command to fetch the sequence from the EMBL database.
2. Prints out the accession number, the organism and the strain separated by semicolons
3. Deletes the sequence file

Hint 1:

You can go through a list with a **for loop**. For example to print out each line in the file you could use

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

Hint 2:

When using variables as part of filenames you have to somehow indicate where the variable name ends. If you tried to for example use:

```
$line.txt
```

bash would understand this as variable named “line.txt”. Braces ({}) can be used to unambiguously identify a variable:

```
${line}.txt
```

Hint 3:

The command format for *entret* (you will need to load the biokit module for this to work):

```
entret dbfetch:embl:<accession_number> <output_file> -filter
```

Hint 4:

You can first run entret for one accession number to see the output file type.

```
entret dbfetch:embl:AY260438 AY260438.entret –filter
```

You can use commands like *grep* and *awk* like in Exercise 3 to parse out the desired information.

Exercise 1:

```
#!/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

```
#!/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
```

```
# Extra task 1

# First make temp files

awk '{print $1}' pb_blast_results | grep -v "#" | sort |uniq > found.list
grep ">" R.fasta | awk '{print $1}'|cut -c 2- > all.list

# with grep:

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

# With diff:

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




# Extra task 2

# oneliner:

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 |less
```