RNASeq2015 Course

CINECA, October 19-22 2015

RNA-seq Hands on Exercise

Fabrizio Ferrè, University of Bologna Alma Mater (fabrizio.ferre@unibo.it)

Hands-on tutorial based on the EBI teaching materials from Myrto Kostadima and Remco Loos at https://www.ebi.ac.uk/training/online/

Index

1. Introduction

The goal of this hands-on session is to perform some basic tasks in the preliminary and downstream analysis of RNA-seq data. You will start from RNA-seq reads originating from two zebrafish embryos at two different development stages. You will learn how to check read quality, perform read trimming, align the trimmed reads to the zebrafish genome, perform transcriptome reconstruction and compare the gene expression between the two different conditions, how to characterize the list of differentially expressed genes, how to reconstruct the transcriptome using a *de novo* assembly approach, and how to identify genomic variants in your data

1.1 RNA-Seq analysis tools

The table below provides the tools for RNA-Seq analysis that you will use in this tutorial.

Tool		URLs
FastQC	Home	http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
	Program	http://www.bioinformatics.babraham.ac.uk/projects/download.html#fastqc
	Manual	http://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/
Trimmomatic	Home	http://www.usadellab.org/cms/?page=trimmomatic
	Program	http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.33.zip
	Manual	http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/TrimmomaticManual_V0.32.pdf
Bowtie2	Home	http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
	Program	http://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.2.6/
	Manual	http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml
TopHat	Home	https://ccb.jhu.edu/software/tophat/index.shtml
	Program	https://ccb.jhu.edu/software/tophat/downloads/tophat-2.1.0.tar.gz
	Manual	https://ccb.jhu.edu/software/tophat/manual.shtml
Cufflinks	Home	http://cole-trapnell-lab.github.io/cufflinks/
	Program	http://cole-trapnell-lab.github.io/cufflinks/install/
	Manual	http://cole-trapnell-lab.github.io/cufflinks/manual/
STAR	Home	https://github.com/alexdobin/STAR
	Program	https://github.com/alexdobin/STAR/releases
	Manual	https://github.com/alexdobin/STAR/blob/master/doc/STARmanual.pdf
BWA	Home	http://bio-bwa.sourceforge.net/
	Program	http://sourceforge.net/projects/bio-bwa/files/
	Manual	http://bio-bwa.sourceforge.net/bwa.shtml
Velvet	Home	https://www.ebi.ac.uk/~zerbino/velvet/
	Program	https://www.ebi.ac.uk/~zerbino/velvet/velvet_1.2.10.tgz
	Manual	https://www.ebi.ac.uk/~zerbino/velvet/Manual.pdf
Samtools	Home	http://samtools.sourceforge.net/
	Program	http://sourceforge.net/projects/samtools/files/
	Manual	http://www.htslib.org/doc/samtools.html
Varscan	Home	http://varscan.sourceforge.net/
	Program	http://sourceforge.net/projects/varscan/files/
	Manual	http://varscan.sourceforge.net/using-varscan.html

For a more complete overview of next generation sequencing software tools required for basic pipelines and for more advanced analysis see:
http://seqanswers.com/wiki/Software/list

2. Your environment

In the *pico* environment, a scheduler system will queue your jobs and execute them as soon as there are free nodes, based on the resources (threads, memory, wall-time) that you specified. The commands for all the processes that you need to run should be written into a shell script file and submitted to the scheduler. You should open a file with a text editor (for example `vi`, `nano` or `pico`), and write at the beginning of the file a header similar to the following:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"
```

Then, in the body of the script you should first load the appropriate modules. In the *pico* environment, several bioinformatics modules are ready be loaded. You need first to load two profiles containing prerequisites for some of the available software, by typing in the script:

```
module load profile/bio
module load profile/advanced
```

After that, you could write Unix commands, calls to installed software, and so on. Alternatively, you can run commands and software from the command line. Unless you log in into *pico* using an interactive shell, the *pico* front end has stringent limits to the process that you run from command line. In this tutorial some short processes can be run from command line, while other requiring more resources will be run through the scheduler.
Once you log in into the *pico* front end, you can load the needed profiles by typing the following commands (`$` is the shell prompt, press `enter` after each command):

```
$ module load profile/bio
$ module load profile/advanced
```

Now all available modules are ready to be loaded.

3. Your data

You will use a dataset derived from sequencing of mRNA from zebrafish (*Danio rerio*) embryos in two different developmental stages. Sequencing was performed on the Illumina platform and generated 76bp paired-end sequence data using polyA-selected RNA. Due to the time constraints of the practical you will only use a subset of the reads, but the steps that you are going to follow are the same whether you are using the whole data or only a subset. Original data can be found here:
`http://www.ebi.ac.uk/ena/data/view/ERR022484` and
`http://www.ebi.ac.uk/ena/data/view/ERR022485`

You need first to create a directory called `zebrafish` (or whatever other name you like) as a subdirectory of your home, to store all data and results. The sequencing data are in another folder in *pico* called `data`. You need to copy this folder into the folder you just created. In this tutorial, each step will be written explicitly, but you should be able by now to perform most of the above and the following steps using Unix commands, so try by yourself before resorting to help. Don't worry about doing something wrong, you should not be able to do major damage. Below is the list of commands; as before, `$` is the shell prompt, press `enter` after each command, and everything after the `#` is a comment ignored by the shell, and you don't need to type it:

```
$ pwd                          # where are you
$ ls                           # what's in your home
$ mkdir zebrafish              # create the work folder
$ cd zebrafish                 # and move into it
$ cp -r /pico/home/userexternal/fferre00/tutorial/data .
$ ls -l data                   # what's in the folder
```

Note that the `-r` parameter for the `cp` command instructs `cp` to copy entire folders. In the `data` folder you will find the following data files:

- `2cells_1.fastq` and `2cells_2.fastq`: these files are derived from paired-end RNA-seq data of a 2-cell zebrafish embryo, the first file containing the forward reads and the other the reverse reads;
- `6h_1.fastq` and `6h_2.fastq` : these files are derived from paired-end RNA-seq data of zebrafish embryos 6 hours post-fertilization, the first file containing the forward reads and the other the reverse reads.

Let's look at a fastq file. These are text files that you could open with any text editor (`vi`, `emacs`, `pico`, etc.), but these files are generally so large that trying to open them will cause lots of problems. Remember the commands `more` (to look at a file one page at the time), `head` (to see the file beginning) and `tail` (to see its end).

In this case let's look at the beginning of a fastq file with `head <filename>` (choose any file in the data folder). For example:

```
$ head data/2cells_1.fastq
```

By default, head shows the first ten rows of the file. How many complete reads are you looking at? Remember that in the fastq file each read is represented by four rows (read name, read sequence, spacer, phred scores). The file in the example contains the forward reads; let's look at the corresponding reverse reads:

```
$ head data/2cells_2.fastq
```

Check the name of the first read in the forward file and of its mate in the corresponding reverse file. In the two files, paired reads occupy the same position within the files (e.g. the mate of the first read in the forward file is the first also in the reverse file, and so on). Hence, the two files must contain the same number of reads. How can you quickly known how many reads are in the fastq file? Run the following commands:

```
$ wc -l data/2cells_1.fastq
$ wc -l data/2cells_2.fastq
```

You will get the number of rows in the two files, which must be identical. Dividing this number by four gives you the number of reads. If the number of rows in the forward and reverse files is not the same, and these numbers are not multiples of four, then something is very wrong.
Can you tell which quality encoding our fastq formatted reads are in? Look at the first few reads of the file `2cells_1.fastq` using the `head` command, and then compare the quality strings with the table found at:

`http://en.wikipedia.org/wiki/FASTQ_format#Encoding`

Knowing the encoding of the quality scores is required for some steps of the RNA-seq pipelines. Some tools can try to guess the encoding, while others require the user to specify it. The Wikipedia page provides lots of information on how the encodings changed over the years.
You are now ready for checking how good your reads are.

4. Quality control

You are now going to use **FastQC** to verify read quality. First, you need to create a folder where to store the **FastQC** output files. Let's call it **Fastqc** or any other name you like:

```
$ mkdir Fastqc
```

Now, let's see the **FastQC** options. In the *pico* environment, several bioinformatics modules are ready be loaded. So let's first load the **FastQC** module and then see its command line options:

```
$ module load fastqc
$ fastqc -h
```

Most options are needed only for particular cases, and you can generally ignore them. Let's run **FastQC** on the forward reads fastq file of the 2-cells sample:

```
$ fastqc -o Fastqc -t 2 --extract --nogroup data/2cells_1.fastq
```

-o: the output folder
-t: the number of threads (**FastQC** is fast, you don't need to dedicate lots of resources to run it)
--extract: instruct FastQC to extract the compressed output files
--nogroup: show output data for each position in the read, instead of grouping neighboring positions.

The last argument is the full path to the fastq file. More than one fastq file can be analyzed in one run, but let's do it one file at the time. Now let's look at the **FastQC** output:

```
$ ls -l Fastqc
```

You should see two files (a zip archive and an html file, and a folder). The html file can be opened by any browser, and will show plot reports similar to those that you saw during the presentation, with quality distributions, GC contents and so on. To look at these plots you must copy these files on your laptop by **sftp** or **scp**. Alternatively, in the folder there are two text files that you can look. The **summary.txt** reports the outcome of all the performed tests. Let's look at it:

```
$ more Fastqc/2cells_1_fastqc/summary.txt
```

You should see printed on screen something like this:

```
PASS    Basic Statistics    2cells_1.fastq
PASS    Per base sequence quality   2cells_1.fastq
PASS    Per tile sequence quality   2cells_1.fastq
PASS    Per sequence quality scores 2cells_1.fastq
FAIL    Per base sequence content   2cells_1.fastq
WARN    Per sequence GC content 2cells_1.fastq
```

```
PASS    Per base N content  2cells_1.fastq
PASS    Sequence Length Distribution    2cells_1.fastq
FAIL    Sequence Duplication Levels2cells_1.fastq
PASS    Overrepresented sequences   2cells_1.fastq
PASS    Adapter Content 2cells_1.fastq
FAIL    Kmer Content    2cells_1.fastq
```

Most tests were successful, and we can ignore the failed ones. The file `fastqc_data.txt` contains a more detailed report, equivalent of the plots in the html output file but in text format. Either case, let's check whether in the *Per base sequence quality* report there are positions where the mean or median quality drops below 20.

Now repeat the same steps for the three remaining fastq files in the `data` folder:

```
$ fastqc -o Fastqc -t 2 --extract --nogroup data/2cells_2.fastq
$ fastqc -o Fastqc -t 2 --extract --nogroup data/6h_1.fastq
$ fastqc -o Fastqc -t 2 --extract --nogroup data/6h_2.fastq
```

You should see in the end of the runs that all four files seem to be of good quality without any glaring problem. In any case, it is always a good idea to perform some trimming, since even if the overall quality is good, there could be individual reads having low quality.

5. Trimming

We will use **Trimmomatic** for read trimming and filtering, and adapters removal. First, let's load the module (load also the profiles bio and advanced if you have not done it before):

```
$ module load profile/bio
$ module load profile/advanced
$ module load autoload trimmomatic
```

Let's see its general usage. NB: for some reasons even after importing the module, to run **trimmomatic** you need to specify its full path: **/cineca/prod/applications/trimmomatic/0.33/binary/trimmomatic-0.33.jar**):

```
$ java -jar
/cineca/prod/applications/trimmomatic/0.33/binary/trimmomatic-
0.33.jar -h

Usage:
       PE [-threads <threads>] [-phred33|-phred64] [-trimlog
<trimLogFile>] [-basein <inputBase> | <inputFile1>
<inputFile2>] [-baseout <outputBase> | <outputFile1P>
<outputFile1U> <outputFile2P> <outputFile2U>] <trimmer1>...

   or:

       SE [-threads <threads>] [-phred33|-phred64] [-trimlog
<trimLogFile>] <inputFile> <outputFile> <trimmer1>...
```

For paired end reads (PE), after specifying some options, the arguments are the input forward and reverse fastq files, the names of the paired and unpaired remaining reads after trimming for the forward reads, and the names of the paired and unpaired remaining reads after trimming for the reverse reads. Finally, you need to specify the parameters for the trimming operations. Since the **Trimmomatic** output files are fastq files, you must be careful of not mixing them with the original un-trimmed fastq files, or to overwrite the un-trimmed fastq files. You can create a specific folder for the trimmed files, or use specific names that will remind you what they are. Let's try for the forward 2-cells fastq file:

```
$ java -jar
/cineca/prod/applications/trimmomatic/0.33/binary/trimmomatic-
0.33.jar PE -threads 2 -phred33 data/2cells_1.fastq
data/2cells_2.fastq data/2cells_1.trim.fastq
data/2cells_1.trim.unpaired.fastq data/2cells_2.trim.fastq
data/2cells_2.trim.unpaired.fastq LEADING:20 TRAILING:20
AVGQUAL:20 MINLEN:25
```

The arguments are:
PE: specify that reads are paired end
-threads: number of threads
-phred33: quality scale
paired forward reads output file
unpaired forward reads output file
paired reverse reads output file
unpaired reverse reads output file
LEADING: quality threshold for removing nucleotides from the 5' end
TRAILING: quality threshold for removing nucleotides from the 3' end
AVGQUAL: mean read quality threshold
MINLEN: minimum read length

At the end of the run, you should get a report on screen telling how many reads passed the filters, for example:

```
Input Read Pairs: 786742 Both Surviving: 770010 (97.87%)
Forward Only Surviving: 14831 (1.89%) Reverse Only Surviving:
1596 (0.20%) Dropped: 305 (0.04%)
TrimmomaticPE: Completed successfully
```

With these parameters, all output files are written in the **data** folder, where the original fastq files are, but you can chose a different setting. The files **2cells_1.trim.fastq** and **2cells_2.trim.fastq** contain trimmed reads that after the filtering have maintained their mate, while the files **2cells_1.trim.unpaired.fastq** and **2cells_2.trim.unpaired.fastq** contain reads that have lost their mate. Using the Unix commands that you used before, count the number of reads in the trimmed fastq files, and verify that the forward and reverse fastq files have the same number of reads. Moreover, by looking at the beginning of the forward and reverse file, verify that the pairing order of reads has been maintained (i.e. the first read of the forward file must be the mate of the first read of the reverse file).

Since paired end read files must have the same number of reads and with the same order in the forward and reverse files, unpaired reads must kept separated.
Count the number of the reads that have lost their mate for the forward and reverse reads.
In this tutorial, unpaired reads will be ignored from now on, but they can be used in principle by treating them as single end reads.
If you want to verify that the trimming procedure worked, you could run again FastQC on the trimmed fastq files and compare the output reports against the raw fastq files, but let's skip this step.

Now try yourself repeating the same steps for the other sample (the 6-houts embryo) in the **data** folder (NB: be careful on the input and output file names), and once finished check that in the **data** folder now you have all trimmed fastq files, paired and unpaired (8 trimmed files, 4 for the 2-cells embryo and 4 for the 6-hours embryo).

6. The tuxedo pipeline

6.1 Genome indexing for bowtie

To map reads to a reference genome, all mapping tools require the genome to be converted into particular structures for quick access and to keep the memory footprint small. In this tutorial we will use **tophat2** for the transcriptome reconstruction, which in turn requires **bowtie2** to run for the read mapping to the genome. **Bowtie2** needs the genome to be indexed using the Burrows-Wheeler transform, and provides a tool (**bowtie2-build**) to obtain this transformation starting from the genome sequence stored in a text file in fasta format. You will also try another mapper, **STAR**, which performs quite well and can be easily integrated with **cufflinks**/**cuffdif**, as you will see later.

Let's first look at the **bowtie2-build** options:

```
$ module load bowtie2
$ bowtie2-build --help
```

The general usage is:

```
Usage: bowtie2-build [options]* <reference_in> <bt2_index_base>
```

You need to specify the genomic sequence file (**reference_in**) and a label to identify the index (**bt2_index_base**), which will be the prefix of all files written by **bowtie2-build**. Copy a folder containing the genomic sequence with the following command:

```
$ cp -r /pico/home/userexternal/fferre00/tutorial/genome .
```

Check the content of the **genome** folder that you just copied:

```
$ ls -l genome
```

Have a look at the file with the commands **more** or **head**. This is a file in fasta format, widely used for storing nucleotide or amino acid sequences. The first row, and any row starting with **>**, is the name of the sequence. For this tutorial, only the zebrafish chromosome 12 is in this file, and all the reads were already filtered for mapping on it.
Try to count the number of nucleotides in the chromosome 12: knowing that each row in the fasta file contains 60 nucleotides, how would you know the chromosome size?
To keep things well organized, you can create a folder to store the index (name it for example **bt2Index** or any other name you like):

```
$ mkdir bt2Index
```

Once you created the folder, let's open a text file with a text editor. You can call the file any name you want, for example **bowtie2index.sh**. Write at the beginning of the file the header for the scheduler, and then load the required modules:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load bowtie2
```

Remember to change `<your account>` to the account name that was given to you (without the `<>` characters). Now write the Unix command to instruct the script to go in the folder in which to execute the commands. For my account, I would write in the script:

```
cd /pico/home/userexternal/fferre00/zebrafish/bt2Index
```

If you are not sure the full path, i.e. where your folder is, check the path of the folder using `pwd`.
Now you can write the command to index the genome:

```
bowtie2-build -f ../genome/Danio_rerio.Zv9.66.dna.fa ZV9
```

`-f`: specify that the genome is in fasta format
`../genome/Danio_rerio.Zv9.66.dna.fa:` path to the genome fasta file (remember that the command `..` indicates the parent folder, i.e. the one right above that in which you currently are)
`ZV9`: the prefix that I chose for all files written by `bowtie2-build` (but you can chose a different one)

Save and close the file. Now you can submit it to the scheduler using the `qsub` command, like this:

```
$ qsub bowtie2index.sh
```

The job is submitted to the scheduler, and the job identifier is written on screen. You can check the status of the job using the `qstat` command. This command returns the status of all jobs from all users, hence to check only your job you can write something like this:

```
$ qstat | grep <your user name>
```

Remember that the operator `|` pipes the output of a command (`qstat` in this case) to another command (`grep` in this case). The command `grep` will filter the `qstat` output to select only those lines containing your user name.
Once the job is finished, look at the content of the folder (with `ls -l`). The bunch of files having extension `.bt2` and prefix `ZV9` contain the index of the genome (or in your case only of the chromosome 12) in a format that `bowtie2` can use.
Now we are ready to map the reads onto the chromosome 12.

6.2 Read mapping using `tophat`

There are numerous tools performing short read alignment and the choice of aligner should be carefully made according to the analysis goals/requirements. Here you will use `tophat2`, a widely used ultrafast aligner that performs spliced alignments.
Let's first load the `tophat` module so that you can see its several parameters:

```
$ module load tophat
$ tophat --help
```

The general format of the `tophat` command is:

```
$ tophat [options]* <index_base> <reads_1> <reads_2>
```

where the last two arguments are the fastq files of the paired end trimmed reads, and the argument before is the prefix of the indexed genome which in your case is ZV9 (unless you choose a different one).
Let's create a folder in which you will run and store the `tophat` output:

```
$ mkdir tophat_out
```

Now open a text file with a text editor that you will use to align the 2-cells zebrafish embryo sample, calling the file with a meaningful name (for example `tophat_2cells.sh`), and write the header for the scheduler as you did before for running `bowtie2-build`. Load the needed modules and point the script to the folder you just created. You script should look something like this:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load tophat

cd /pico/home/userexternal/fferre00/zebrafish
```

Here you need to take an extra step to solve a compatibility issue between `tophat` and some versions of `samtools` (a suite of utilities for handling SAM/BAM files). You will need to add to your path an older version of `samtools`. Write the following in your script (in one line):

```
export     PATH=//cineca/prod/applications/samtools/0.1.19/gnu--
4.8.3/bin/:$PATH
```

Now write in the script file the **tophat** command (all in one line):

```
tophat --solexa-quals -g 2 --library-type fr-unstranded  -o
tophat_out/2cells bt2Index/ZV9 data/2cells_1.trim.fastq
data/2cells_2.trim.fastq
```

The parameters are:
--solexa-quals: quality scale of the reads
-g: maximum number of multihits allowed. Short reads are likely to map to more than one locations in the genome even though these reads can have originated from only one of these regions. In general, only a restricted number of multihits is tolerated, and in this case you are asking **tophat** to report only reads that map at most onto 2 different loci.
--library-type: type of the sequencing (in your case paired end forward/reverse unstranded)
-o: output folder name. Given that for every run the name of the output files is the same, you need to specify different folders for each run, lest the output files will be overwritten.

Now save and close the file, and submit it using **qsub** as you did before. This is going to be the longest step of the pipeline, requiring few minutes (10/20 minutes at most). While the job is running, you can see the files that **tophat** is writing (using **ls -l**). A temporary folder will be created where large files will be written, then deleted after completion.

In the meantime, let's prepare the script file for the 6-hours post-fertilization sample. Open a file (call it for example **tophat_6h.sh**), write the header, load the modules, point to the right folder, and add the PATH extension to the older **samtools** version, identically as you did for the 2-cells sample. Then write the **tophat** command for the 6 hours sample (remember to change the output folder name, otherwise the second run will overwrite the first run files). You should know now how to do that, so try by yourself before looking at the command below (all in one line):

```
tophat --solexa-quals -g 2 --library-type fr-unstranded  -o
6hours ../bt2Index/ZV9 ../data/6h_1.trim.fastq
../data/6h_2.trim.fastq
```

Save and close, run it with **qsub**, check the status with **qstat**. When the two jobs are finished, look into the output folders. The file that we need is the **accepted_hits.bam**, where the read alignments are stored in a binary version of the Sequence Alignment Map (SAM) format. For more information regarding the SAM format please see **http://samtools.sourceforge.net/SAM1.pdf**

Another interesting file is the **align_summary.txt** file, where some statistics on the run outcome are reported. For the 2-cells embryo sample, your **align_summary.txt** file should look something like this:

```
Left reads:
          Input     :    770010
           Mapped   :    716310 (93.0% of input)
            of these:     52123 ( 7.3%) have multiple
```

```
alignments (22408 have >2)

Right reads:
          Input      :    770010
           Mapped    :    715438 (92.9% of input)
            of these:     52189 ( 7.3%) have multiple
alignments (22401 have >2)

93.0% overall read mapping rate.

Aligned pairs:    662966
     of these:     49472 ( 7.5%) have multiple alignments
                    6893 ( 1.0%) are discordant alignments

85.2% concordant pair alignment rate.
```

The BAM files cannot be looked, but their SAM counterparts can. You will see later that the `samotools` suite of tools offers several utilities for conversion and analysis of SAM/BAM files. For example, to convert a BAM into SAM and check the first lines of the SAM file you can write (but don't do it now):

```
$ samtools view file.bam | head
```

6.3 Transcriptome reconstruction and expression using `cufflinks`

There are a number of tools that perform reconstruction of the transcriptome and for this workshop you are going to use `cufflinks`, which can do transcriptome assembly with and without a reference annotation. It also quantifies the isoform expression in FPKMs. Let's import the cufflinks module and check its (rather long) list of parameters:

```
$ module load cufflinks
$ cufflinks --help
```

You are going to use `cufflinks` with the Ensembl annotation file for zebrafish limiting the transcriptome reconstruction strictly to the available annotations. You could also use the annotations as a guide but letting the algorithm look also for transcribed loci not included in the annotations. In the first case `cufflinks` will only report isoforms that are included in the annotation, while in the latter case it will report novel isoforms as well. First, copy the annotation file from Ensembl for chromosome 12 of *Danio rerio*:

```
$ cp -r /pico/home/userexternal/fferre00/tutorial/annotations .
```

Then let's create a folder for the `cufflinks` output storage:

```
$ mkdir cuff_out
```

Now you are ready to run `cufflinks`. The general format of the `cufflinks` command is:

```
cufflinks [options]* <aligned_reads.(sam/bam)>
```

where the input is the aligned reads (either in SAM or BAM format).

Prepare a script file to launch `cufflinks`, by opening it with a text editor (you can call it for example `cufflinks_2cells.sh`). Copy the same header for the scheduler that you used previously, use the commands to load the required modules, and point the script to your working space. Your script should look something like this:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load cufflinks

cd /pico/home/userexternal/fferre00/zebrafish
```

Then you can write (all in one line) the command:

```
cufflinks -o cuff_out/2cells -G
annotations/Danio_rerio.Zv9.66.gtf -b
genome/Danio_rerio.Zv9.66.dna.fa -u
--library-type fr-unstranded
tophat_out/2cells/accepted_hits.bam
```

The options that you used above are:
`-o`: output directory
`-G`: tells `cufflinks` to use the supplied annotation in order to estimate isoform annotation.
`-b`: instructs `cufflinks` to run a bias detection and correction algorithm which can significantly improve accuracy of transcript abundance estimates (the genome sequence is required to perform this).
`-u`: tells `cufflinks` to do an initial estimation procedure to more accurately weight reads mapping to multiple locations in the genome.
`--library-type:` specify the employed sequencing strategy

Save, close and run using `qsub`. With your small dataset, it should not take more than few minutes. While waiting, you can prepare and run a similar file for the 6-hours embryo sample (again, remember to change the output folder name, otherwise the second run will overwrite the first run files). You should know now how to do that, so try by yourself before looking at the command below:

```
cufflinks -o cuff_out/6hours -G
annotations/Danio_rerio.Zv9.66.gtf -b
genome/Danio_rerio.Zv9.66.dna.fa -u
--library-type fr-unstranded
tophat_out/6hours/accepted_hits.bam
```

When both have finished, you can take a look at the output folders that have been created. The results from cufflinks are stored in 4 different files:

`genes.fpkm_tracking`: This file contains the estimated gene-level expression values in the generic FPKM Tracking Format.
`isoforms.fpkm_tracking`: This file contains the estimated isoform level expression values in the generic FPKM Tracking Format.
`transcripts.gtf`: This GTF file contains the assembled isoforms.
`skipped.gtf`: This GTF file contains the skipped loci (it's often empty)

Let's have a look at the file `genes.fpkm_tracking`, where gene expression is stored. The first column reports the Ensembl identifier of the gene, the fifth its common name, the seventh its genomic coordinates, and the tenth its expression in FPKM. You would like to know which are the most expressed genes in the sample. This can be done in many way, for example opening the file as a spreadsheet with Excel, but let's do it using Unix commands. You can use the `sort` command, which can order a file based on the values in a specific column:

```
$ sort -t$'\t' -r -g -k 10 cuff_out/2cells/genes.fpkm_tracking
> cuff_out/2cells/genes.sorted.fpkm_tracking
```

If you want to better understand the command that you just typed, check the manual page for the `sort` command using:

```
$ man sort
```

The first parameter of `sort` tells the command that the file is divided in columns separated by tabs; the `-k` parameter specify which column you are interested in; the `-g` tells `sort` that the column contains numbers (the default `sort` behavior is to order by lexicographic order); the `-r` tells `sort` to order from the largest to the smallest value. The `sort` output is redirected into a new file. If you open this new file, the genes are now ordered by decreasing expression. If you now want to know, for example, which is the most expressed gen, you can copy the Ensembl identifier in the first row of the sorted file, and use Ensembl or the UCSC genome browser to get more information about it (if everything went well, the most expressed gene in the 2-cell sample should be the U6 spliceosomal RNA). Do that for a few genes.

Now do the same also for the 6-hours embryo sample. Which are the most expressed genes in this other sample? Are they in common with the 2-cells embryo?

Now that you have all the strongly expressed genes in the two embryos, you might wonder in which biological processes they participate. Let's see how you can use an automatic web-based functional annotation tool (DAVID) to help you in the

rationalization of which are the most represented process in a biological sample. To use DAVID you need to submit a list of gene identifiers. In the `genes.sorted.fpkm_tracking`, the first column contains the Ensembl gene ids of the genes in the annotation GTF file, all starting with the prefix ENSDARG. You need to extract these identifiers from this file. Let's take the top 100 of the genes sorted by FPKM. Let's use the command `head` to select the top 100 genes, and the command `cut` to extract only the first column from these rows (if you want, use `man` to have more details on how these commands work):

```
$ head -100 cuff_out/2cells/genes.sorted.fpkm_tracking

| cut -f 1 > cuff_out/2cells/top100byFpkm.txt
```

Remember that the `>` operator redirects the output of a command into a file. Now open the `top100byFpkm.txt` file, verify that it's a single-column file with the Ensembl gene identifiers, and copy its content. Open any browser and go to:

```
https://david.ncifcrf.gov/home.jsp
```

Click the `Start Analysis` tab at the top of the screen, click the `Upload` tab on the left of the screen and paste the list of genes into the box labeled `Step 1: Enter Gene List`. In `Step 2: Select Identifiers` choose `ENSEMBL_GENE_ID` from the list, and mark `Gene List` in `Step 3: List Type`, then click `Submit List` in `Step 4`. After the page ends the processing, click `Functional Annotation Tool` to be redirected to the result page. Here you can expand all tabs and click on the different functional categories to se if there is some enrichment in your gene list. The one that might interest you the most is `GOTERM_BP_FAT` under `Gene_Ontology`; do these categories make sense given the samples you are studying?

Now repeat for the 6-hours embryo sample. Are there differences between the two samples?

6.4 Differential Expression using `cuffdiff`

One of the stand-alone tools that are part of the `cufflinks` package, and that performs differential expression estimation, is `cuffdiff`. You can use this tool to compare between two conditions; for example control and disease, or wild-type and mutant, or, as in your case, we want to identify genes that are differentially expressed between two developmental stages. You don't need to load `cuffdiff` since it is part of the `cufflinks` module that you loaded previously. So let's just create an output folder to store its results:

```
$ mkdir cdiff_out
```

The general format of the `cuffdiff` command is:

```
cuffdiff [options]* <transcripts.gtf>
<sample1_replicate1.sam[,...,sample1_replicateM]>
<sample2_replicate1.sam[,...,sample2_replicateM.sam]>
```

where the input is an annotation file and the aligned reads (either in SAM or BAM format) for the two conditions to be compared. You can now prepare the script for running `cuffdiff`, by opening a text file (let's call it `cuffdiff.sh`), and writing the header, the loading module commands, and pointing to your working space:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load cufflinks

cd /pico/home/userexternal/fferre00/zebrafish
```

Now write the command to run `cuffdiff` (all in one line):

```
cuffdiff -o cdiff_out -L ZV9_2cells,ZV9_6h -b
genome/Danio_rerio.Zv9.66.dna.fa -u
--library-type fr-unstranded
annotations/Danio_rerio.Zv9.66.gtf
tophat_out/2cells/accepted_hits.bam
tophat_out/6hours/accepted_hits.bam
```

The options that you used above are:
`-o`: output directory,
`-L`: labels for the different conditions,
`-b`, `-u`, `--library-type`: same meaning as described before for `cufflinks`

The output folder (check its contents with `ls -l`) contains a number of files reporting differential usage for different genomic entities. You are mostly interested in the differential expression at the gene level. This is reported in the file `gene_exp.diff`. Look at the first few lines of the file using the following command:

```
$ head cdiff_out/gene_exp.diff
```

You would like to see which are the most significantly differentially expressed genes. To do this, you can sort the above file according to the q-value (corrected p-value for multiple testing). Let's use a `sort` command to sort the file and write the sorted file in a different one called `gene_exp_sorted.diff`. Write the following command (all in one line):

```
$ sort -t$'\t' -g -k 13 cdiff_out/gene_exp.diff  >
cdiff_out/gene_exp_sorted.diff
```

Look again at the first few lines of the sorted file. If everything went as it should have, the

first 18 genes should have q-value < 0.05, chosen as significance threshold, and are therefore differentially expressed between the zebrafish 2-cells embryo and the 6 hours embryo.

Now you have all the genes that change between the two conditions. In your example, the number of these genes is quite small, but in more realistic cases you might end up with hundreds of differentially expressed genes. Let's use again DAVID to help you in the rationalization of which biological functions change between the two analyzed conditions. Let's take the top 100 of the genes sorted by q-value from the **gene_exp_sorted.diff** file (in more realistic cases you would have taken only genes having q-value < 0.05, but in your example the 18 genes satisfying this condition are too few to have a significant functional characterization). As you did before, use the command **head** to select the top 100 genes, and the command **cut** to extract only the first column from these rows:

```
$ head -100 cdiff_out/gene_exp_sorted.diff | cut -f 1 > top100genes.txt
```

Remember that the **>** operator redirects the output of a command into a file. Now open the **top100genes.txt** file, verify that it's a single-column file with the Ensembl gene identifiers, and copy its content. Open any browser and go to:

```
https://david.ncifcrf.gov/home.jsp
```

The procedure is the same that you did for the **cufflinks** output. Click the **Start Analysis** tab at the top of the screen, click the **Upload** tab on the left of the screen and paste the list of genes into the box labeled **Step 1: Enter Gene List**. In **Step 2: Select Identifiers** choose **ENSEMBL_GENE_ID** from the list, and mark **Gene List** in **Step 3: List Type**, then click **Submit List** in **Step 4**.
After the page ends the processing, click **Functional Annotation Tool** to be redirected to the result page. Here you can expand all tabs and click on the different functional categories to se if there is some enrichment in your gene list. The one that might interest you the most is **GOTERM_BP_FAT** under **Gene_Ontology**; do these categories make sense given the samples you are studying?

7 Using STAR for read mapping

STAR is a fast and accurate read mapper that can handle spliced reads, and can therefore be included in the RNA-Seq analysis pipeline instead of the bowtie/tophat step. Let's see how it works and how to use its mapping for differential expression analysis.

7.1 Genome indexing for STAR

First, you need to create a folder for storing the STAR indexed genome:

```
$ mkdir starIndex
```

Then, you need to write a script to run the indexer. Let's call it starIndex.sh. Write the usual header and load the appropriate modules, then point to your working directory:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load autoload star

cd /pico/home/userexternal/fferre00/zebrafish
```

Write in the file the STAR indexing command for the zebrafish chromosome 12:

```
STAR --runThreadN 2 --runMode genomeGenerate --genomeFastaFiles
genome/Danio_rerio.Zv9.66.dna.fa --genomeDir starIndex
```

The employed parameters are:
--runThreadN: number of threads
--runMode: tells STAR to create the index
--genomeFastaFiles: where is the genome sequence
--genomeDir: where to write the index

Close and save the file, and run it using qsub.

7.2 Read mapping using STAR

Now you will use the index to map reads onto it. STAR includes several parameters that you will find described in the documentation. Some of these parameters must be set to ensure compatibility with cuffdiff, which you will use later on the STAR-mapped reads. Let's create a folder to store the results of the read mapping:

```
$ mkdir starMapping
```

Open a text file, call it `star_2cells.sh` (or whatever you like), write header, module loads and path to the working folder:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load autoload star

cd /pico/home/userexternal/fferre00/zebrafish
```

Now write the long `STAR` mapping command for the trimmed 2-cell embryo data (all in one line):

```
STAR --outSAMstrandField intronMotif --outFilterIntronMotifs
RemoveNoncanonical --runThreadN 2 --genomeDir starIndex
--sjdbGTFfile annotations/Danio_rerio.Zv9.66.gtf
--outSAMtype BAM Unsorted --outFileNamePrefix
starMapping/2C.star
--readFilesIn data/2cells_1.trim.fastq data/2cells_2.trim.fastq
```

The employed parameters are:
`--outSAMstrandField`: annotates spliced alignment with strand information, required by `cuffdiff`
`--outFilterIntronMotifs`: remove non-canonical junctions, required for `cuffdiff`
`--runThreadN`: number of threads
`--genomeDir`: where is the genome index
`--sjdbGTFfile:` path to the annotation file in GTF format. STAR will extract splice junctions from this file and use them to improve mapping accuracy.
`--outSAMtype`: format of the output mapping file
`--outFileNamePrefix`: path and prefix of the output files
`--readFilesIn`: the input fastq files

Now save and close, and run using `qsub`. While the script is running, prepare a similar file for the 6-hours embryo. Remember to change the output file prefix (i.e. the `--outFileNamePrefix` parameter, for example into `starMapping/6H.star`).

After both runs are completed, in the output folder you will find some files, two of them in BAM format (`2C.starAligned.out.bam` and `6H.starAligned.out.bam`, if you followed the previous instructions). You cannot use these as input for `cuffdiff`, which requires the BAM files to be sorted by genomic coordinates. Let's use `samtools` to get this. First, import the `samtools` module and check its usage:

```
$ module load samtools
$ samtools
```

You can see that `samtools` offers a number of different tools for handling of BAM/SAM files. To know the usage of a particular module, you can type `samtools` followed by the name of the module. For example, since we need to sort the BAM files, type:

```
$ samtools sort
```

To sort the BAM file for the 2-cells embryo data, you can type:

```
$ samtools sort -O bam -o starMapping/2C.starAligned.out.sorted.bam -T tmp starMapping/2C.starAligned.out.bam
```

the parameters are:
`-O`: specifies the output format
`-o`: the output file name
`-T`: the prefix for the temporary files that `samtools` will write and then delete.

For the 6-hours sample:

```
$ samtools sort -O bam -o starMapping/6h.starAligned.out.sorted.bam -T tmp starMapping/6h.starAligned.out.bam
```

that you will use as input for `cuffdiff`. Take a look at the files ending in `starLog.final.out`, which contain information on the runs.

7.3 Using `cuffdiff` on `STAR` mapped reads

The sorted BAM file reporting the read alignment can now be given as input for `cuffdiff`. First, create an output folder:

```
$ mkdir cdiff_star
```

Prepare a script file containing the usual stuff:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load autoload cufflinks

cd /pico/home/userexternal/fferre00/zebrafish
```

And write the **cuffdiff** command (all in one line):

```
cuffdiff -o cdiff_star -L ZV9_2cells,ZV9_6h -b
genome/Danio_rerio.Zv9.66.dna.fa -u
--library-type fr-unstranded annotations/Danio_rerio.Zv9.66.gtf
starMapping/2C.starAligned.out.sorted.bam
starMapping/6h.starAligned.out.sorted.bam
```

Save and run with **qsub**. As before, you would like to see which are the most significantly differentially expressed genes. Once **cuffdiff** has finished, let's use a **sort** command to sort the file and write the sorted file in a different one called **gene_exp_sorted.diff**. Write the following command (all in one line):

```
$ sort -t$'\t' -g -k 13 cdiff_star/gene_exp.diff  >
cdiff_star/gene_exp_sorted.diff
```

Look at the first few lines of the sorted file. If everything went as it should have, the first 14 genes should have q-value < 0.05, chosen as significance threshold, and are therefore differentially expressed between the zebrafish 2-cells embryo and the 6 hours embryo. As you can see, the number is slightly different to that obtained using the full **Tuxedo** pipeline. Ho many genes resulted differentially expressed by both pipelines?

Now, repeat the same steps described for the functional characterization of the **cuffdiff** ouput at the end of section 6.4. Let's use the command **head** to select the top 100 genes and the command **cut** to extract only the first column from these rows:

```
$ head -100 cdiff_star/gene_exp_sorted.diff | cut -f 1 >
star_top100genes.txt
```

Open the **star_top100genes.txt** file and copy its content. Go to DAVID (**https://david.ncifcrf.gov/home.jsp**) and proceed with the annotation. Are there major differences with the results you obtained previously?

8. *De novo* assembly with `Velvet`

`Velvet` is an assembly algorithm developed for genomic assembly, but it can be applied to transcriptome assembly as well. A more recent algorithm from the same authors, called `Oases`, was developed specifically for un-guided transcriptome reconstruction, but for our purposes the original `velvet` is enough. Other *de novo* transcriptome reconstruction algorithms exist, for example `Trinity`, which is generally more accurate but has very large computational requirements, especially for memory. `Trinity` offers also modules for gene expression and differential expression estimation, while `velvet` is limited to transcript reconstruction.

`Velvet` is composed by two modules, `velveth` and `velvetg`, that need to be run one after the other. The first module analyzes the reads, decomposes them in sub-sequences of fixed length *k* (called k-mers) and builds an index of all k-mers. The second module takes as input the output of `velveth` and builds structures in which reads overlap, called contigs, corresponding to transcripts when the input contains RNA-seq reads.
Let's open a script for running `velvet`, and write the header, load the modules and point to the working directory:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load autoload velvet

cd /pico/home/userexternal/fferre00/zebrafish
```

The general usage of the `velveth` module is:

```
Usage:
./velveth directory hash_length {[-file_format][-read_type][-
separate|-interleaved] filename1 [filename2 ...]} {...}
[options]
```

where `directory` is the name of the output folder and `hash_length` is the size of the k-mers in which reads are decomposed (must be an even number). Now let's write the `velveth` command to assemble reads from the 6 hours zebrafish embryo (all in one line):

```
velveth velvet_out 29 -fastq data/6h_1.trim.fastq
data/6h_2.trim.fastq
```

The parameters are:
`velvet_out`: the output folder
`29`: the value of *k*
`-fastq`: the input format
The second module, `velvetg`, has many options, but can also simply run just specifying the folder with the `velveth` output. Let's now add the `velvetg` command in the file that you are writing:

```
velvetg velvet_out
```

Now save and close the file and run with `qsub`. When it is finished, have a look at the output folder (using for example `ls -l velvet_out`). Some files (e.g. `Roadmaps`, `Sequences`) report the reads decomposition and indexing. The file with the reconstructed transcript is `contigs.txt`. Open it, and you will see the sequence of all transcripts in fasta format.

Let's now see if `velvet` was able to correctly assemble zebrafish transcripts. Open `contigs.txt`, choose one transcript sufficiently long and copy it. Now open your browser and go to:

```
http://blast.ncbi.nlm.nih.gov/Blast.cgi
```

Click on `nucleotide blast` on the left size of the screen. Paste the chosen sequence from the `velvet` output into the text box, select `Nucleotide collection (nr/nt)` from `Choose Search Set`, then click the `BLAST` button and wait until completion. Verify that the best match in the BLAST output is a zebrafish gene. If you want, try again with other reconstructed transcripts, trying some large ones and some small ones. Do all of them have a good match with zebrafish genes?

9. Using bwa for read mapping

Another popular mapping tool is **bwa**, which many prefer to **bowtie**. As an optional exercise, you will learn how to perform the **bwa** mapping, but you will not use it for the later steps of the pipeline, since **bwa** is not able to handle spliced reads and as such is not suitable for transcriptome analysis. RNA-seq analysis pipeline require a mapper able to handle reads coming from exon-exon junctions, which would present a very large gap (corresponding to an intron) when mapped onto the genome. The tool **bwa** is not a spliced aligner, and cannot be used directly for RNA-seq analysis. Nevertheless, it works quite well (as alignment accuracy and computational speed) and learning its usage could help anyone interested in working with NGS data. In this tutorial, you will use the **bwa** alignment for SNP calling, i.e. to identify genomic variations between the reference genome and your sample. SNP calling using RNA-Seq data is obviously limited to gene products and cannot identify variations in promoters and other regulatory regions, moreover is heavily affected by the gene expression levels, nevertheless in some cases can offer valuable insights. You will use a SNP calling algorithm, **varscan**, on the output of the **bwa** mapping.

9.1 Genome indexing for bwa

Let's first look at the **bwa** options:

```
$ module load bwa
$ bwa
```

The **bwa** suite provides a number of commands for various steps of the alignment procedure. The general usage is:

```
bwa <command> [options]
```

Among the available commands there is **index** for genome indexing, and **aln**, **samse** and **sampe** for read mapping. To see the general usage of a particular command type **bwa** followed by the name of the command:

```
$ bwa index
```

```
Usage:   bwa index [-a bwtsw|is] [-c] <in.fasta>
```

You need to specify the genomic sequence file (**in.fasta**) and a label to identify the index (**index**), which will be the prefix of all files written by **bwa index**. You already copied the folder containing the genomic sequence of the zebrafish chromosome 12.
To keep things well organized, you can create a folder to store the index (name it for example **bwaIndex** or any other name you like):

```
$ mkdir bwaMapping
```

Once you created the folder, let's open a text file with a text editor. You can call the file any name you want, for example **bwaindex.sh**. Write at the beginning of the file the

header for the scheduler, and then load the required modules:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load bwa

cd /pico/home/userexternal/fferre00/zebrafish/bwaMapping
```

Now you can write the command to index the genome:

```
bwa index -p ZV9 ../genome/Danio_rerio.Zv9.66.dna.fa
```

Parameters:
`../genome/Danio_rerio.Zv9.66.dna.fa:` path to the genome fasta file
`-p`: the prefix that I chose for all files written by `bwa index` (but you can chose a different one)

Save and close the file. Now you can submit it to the scheduler using the `qsub` command, like this:

```
$ qsub bwaindex.sh
```

Once the job is finished, look at the content of the folder (with `ls -l`). The bunch of files having prefix ZV9 contain the index of the genome (or in your case only of the chromosome 12) in a format that `bwa` can use.

9.2 Read mapping using `bwa`

Read mapping with `bwa` is peculiar in which it requires two steps: in the first, `bwa` produces the mapping using an internal format (called *sai* format) that is different from the BAM/SAM that you saw before. The *sai* files need then to be converted in SAM. Moreover, in case of paired end reads, you will need to process them separately then join the outputs. You will do both steps in one script. Let's open a text file called, for example, `bwamapping.sh`, in which you can write the usual stuff:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=16GB
#PBS -A <your account>
#PBS -W group_list="<your account>"

module load profile/bio
module load profile/advanced
module load bwa
```

```
cd /pico/home/userexternal/fferre00/zebrafish/bwaMapping
```

Now let's write the command to map the forward and reverse fastq files for the 2-cells embryo sample (each command in one line):

```
bwa aln -n 2 -t 4 ZV9 ../data/2cells_1.trim.fastq > 2cells_1.sai

bwa aln -n 2 -t 4 ZV9 ../data/2cells_2.trim.fastq > 2cells_2.sai
```

The parameters are:
`-n`: allowed mismatches
`-t`: number of threads
`ZV9`: the index prefix

In the same file you will also write the command to combine the two *sai* files into one SAM:

```
bwa sampe ZV9 2cells_1.sai 2cells_2.sai ../data/2cells_1.trim.fastq ../data/2cells_2.trim.fastq > 2cells.bwa.sam
```

Save and run with `qsub`. Once finished, check the folder content: you should see two *sai* files and one large SAM file. The SAM can be converted into BAM format using `samtools`, and be used for further analyses.

9.3 SNP calling using `varscan`

First, you need to sort the SAM file, and then generate a so-called pileup file, which describes how the reads align to the reference genome position by position. You will use `samtools` for both steps. Go into the `bwaMapping` folder and type the following command (in one line):

```
$ samtools sort -T tmp -O bam -o 2cells.sorted.bam 2cells.bwa.sam
```

Now let's create the pileup:

```
$ samtools mpileup -f ../genome/Danio_rerio.Zv9.66.dna.fa 2cells.sorted.bam > 2cells.pileup
```

the `-f` parameter specifies the path to the genomic sequence. Now open the `2cells.pileup` file with the `more` command (is generally a very large file, so do not open it with a text editor) and scroll few pages to see its format.
You are now ready for the identification of SNP. `Varscan` has several parameters for setting the reliability of the SNP identification. All SNP calling algorithms face the problem of discriminating between true variations and sequencing errors.
Open a text file called for example `varscan.sh`, and write the header like this:

```
#/bin/bash
#PBS -l walltime=4:00:00
#PBS -l select=1:ncpus=4:mpiprocs=1:mem=8GB
#PBS -A IscrC_RepiTOP
#PBS -W group_list="IscrC_RepiTOP"

cd /pico/home/userexternal/fferre00/zebrafish/bwaMapping
```

Now write the `varscan` command (all in one line):

```
java -jar /cineca/prod/applications/varscan2/2.3.7/jre--1.7.0_72/varscan2-2.3.7.jar pileup2snp 2cells.pileup --p-value 0.01 --min-coverage 50 --min-reads2 20 --min-avg-qual 20 > 2cells.varscan.out
```

the parameters are:
`--p-value`: p-value threshold for calling a SNP
`--min-coverage`: minimum number of reads covering a genomic position
`--min-reads2`: minimum number of reads supporting the variant
`--min-avg-qual`: minimum average quality at the position

Now sort the output file by increasing p-values:

```
$ sort -t$'\t' -g -k 12 2cells.varscan.out > 2cells.varscan.sorted.out
```

Open the sorted file. The first two columns indicate the genomic coordinate of the SNP, the third is the nucleotide in the reference genome in that position, and the fourth is the nucleotide found there in the aligned reads. Copy the coordinate of the first (or of any) SNP, go to the UCSC Genome Browser and point it to the SNP coordinate (remember to set `Jul 2010 (Zv9/danRer7)` as genome assembly). In which gene the SNP falls into?
Once you have the SNPs list, you can try to assess the potential effect of the variation, for example a SNP changing an amino acid into another with different characteristics can be deleterious.

10. Some useful references

Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, Batut P, Chaisson M, Gingeras TR. STAR: ultrafast universal RNA-seq aligner. Bioinformatics. 2013 29(1):15-21.

Koboldt DC, Zhang Q, Larson DE, Shen D, McLellan MD, Lin L, Miller CA, Mardis ER, Ding L, Wilson RK. VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. Genome Res. 2012 22(3):568-76.

Langmead, B., Trapnell, C., Pop, M. & Salzberg, S. L. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. 10, R25 (2009).

Li H. and Durbin R. Fast and accurate short read alignment with Burrows-Wheeler Transform. Bioinformatics 2009, 25:1754-60.

Roberts, A., Pimentel, H., Trapnell, C. & Pachter, L. Identification of novel transcripts in annotated genomes using RNA-Seq. Bioinformatics 27, 2325–2329 (2011).

Roberts, A., Trapnell, C., Donaghey, J., Rinn, J. L. & Pachter, L. Improving RNA-Seq expression estimates by correcting for fragment bias. Genome Biol. 12, R22 (2011).

Trapnell, C., Pachter, L. & Salzberg, S. L. TopHat: discovering splice junctions with RNA-Seq. Bioinformatics 25, 1105–1111 (2009).

Trapnell, C. et al. Transcript assembly and quantification by RNASeq reveals unannotated transcripts and isoform switching during cell differentiation. Nat. Biotechnol. 28, 511–515 (2010).

Zerbino DR, Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res. 2008 18(5):821-91

Zerbino DR. Using the Velvet de novo assembler for short-read sequencing technologies. Curr Protoc Bioinformatics. 2010 Chapter 11:Unit 11.51