RNASeq2017 Course

Salerno, September 27-29, 2017

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 and advanced 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, compute gene expression, compare the gene expression between the two different conditions, how to characterize the list of differentially expressed genes, and how to reconstruct the transcriptome ignoring the reference zebrafish annotations or by using a *de novo* assembly approach, and how to identify genomic variants in your data. For each task, you will find a step-by-step guide throughout this tutorial.

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
Samtools	Home	http://samtools.sourceforge.net/
	Program	http://sourceforge.net/projects/samtools/files/
	Manual	http://www.htslib.org/doc/samtools.html
Trinity	Home	https://github.com/trinityrnaseq/trinityrnaseq/wiki
	Program	https://github.com/trinityrnaseq/trinityrnaseq/releases
	Manual	https://github.com/trinityrnaseq/trinityrnaseq/wiki/Running-Trinity
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:
`https://omictools.com/rna-seq-category`

2. 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 Genome Analyzer II platform and generated 76bp paired-end sequence data from poly(A)-selected RNA. Due to the time constraints of the practical, you will only use a subset of the reads, i.e. only reads that can be mapped to genes in the zebrafish chromosome 12. Nevertheless, the steps that you are going to follow are the same whether you are using the whole data or only a subset. Original complete 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 `tutorial` (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 `/home/studente/Scrivania/Dataset_Corso`. 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 tutorial          # create the work folder
$ cd tutorial             # and move into it
```

Now let's check which data we have:

```
$ ls -l ../Scrivania/Dataset_Corso
```

Remember that `..` indicate the parent folder, i.e. the one right above that in which you currently are. In the `Dataset_Corso` 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.

All these four files contain reads in the standard *fastq* format, that reports, for each read, its sequence and the quality score of each nucleotide; a description of this format can be found at `https://en.wikipedia.org/wiki/FASTQ_format`.
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 ../Scrivania/Dataset_Corso/2cells_1.fastq
```

By default, `head` shows the first ten rows of the file. How many complete reads are you looking at? Remember that four lines describe each read: i) read identifier; ii) read sequence; iii) a spacer row (containing only a `+` in our case); iv) the *Phred* score (quality score) for each read nucleotide. Notice how all read identifiers end with `/1`. This is because this file contains all the forward reads for the 2-cells embryo; let's now look at the corresponding reverse reads:

```
$ head ../Scrivania/Dataset_Corso/2cells_2.fastq
```

Check the name of the first read in the reverse file and notice how it ends with `/2`. You might also notice that the first forward read and the first reverse read share the same identifier (except the `/1` and `/2`). This means that these two reads were sequenced at the two ends of the same cDNA fragment. The reads in the forward and reverse files have the same order, i.e. the first forward read is paired with the first reverse read, the second with the second, and so on. 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 ../Scrivania/Dataset_Corso/2cells_1.fastq
$ wc -l ../Scrivania/Dataset_Corso/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.
Then, do the same also for the `6h_1.fastq` and `6h_2.fastq` files. Do you have more reads for the 2-cells or for the 6-hours embryo?
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 will also find in the `Dataset_Corso` folder a file containing the zebrafish genomic sequence. This file contains a shortened genome containing only the sequence of one particular chromosome, to save time in the read mapping step, which is usually the most time-consuming. The file name is `Danio_rerio.Zv9.66.dna.fa`. The extension `fa` stands for *fasta* format, which is the most common file format for biological sequences.
Have a look at the file with the commands `more` or `head`. In the *fasta* format 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?

Finally, you will also find a file specifying the zebrafish genome annotations, i.e. the coordinates and type of genomic features such as exons, CDS, and so on. Using these information facilitates the transcriptome reconstruction and the expression levels estimation. Obviously, using the genome annotations as a strict guide will make it impossible to identify unknown transcriptional units or unknown splicing variants of known genes. The file name is `Danio_rerio.Zv9.66.gtf`. The extension `gtf` stands for *General Transfer Format*, which is one of the most common formats for transferring genome annotations and features. Other popular formats are *bed* (*Browser extensible Data*) and gff (*General Feature Format*). All formats tend to be similar, associating a set of genomic coordinates (usually defined as chromosome, start, position, end position, strand) with a genomic feature. For a description of the most commonly used file formats for genome annotations, see `https://genome.ucsc.edu/FAQ/FAQformat.html`.
This file contains only genes in the chromosome 12, sorted by start position coordinate. The 9 file columns of the file are: i) chromosome name; ii) feature source; iii) feature type; iv) start position in base-pairs from the 5' end of the + strand; v) end position; vi) score (it's not used in our case); vii) the strand; viii) the reading frame; ix) attributes (for example the gene common name). The third column (Feature) describes the type of annotation at that coordinates (for example whether it is an exon, a start codon, a stop codon, and so on).
Let's count how many CDS are reported in chromosome 12. You can use the `grep` shell command that extracts from a file all rows containing a certain string. You can try the following:

```
$ grep CDS ../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.gtf |
wc -l
```

Remember that the pipe (`|`) symbol redirects the output of a command to another command (in this case `wc`, since we want to count all rows containing the string CDS). The number printed in the terminal then is the number of exons in the CDS of zebrafish genes in the chromosome 12.

You are now ready for checking how good your reads are.

3. 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* command line options:

```
$ fastqc -h
```

The command line parameter `-h` or `--help` is generally used to retrieve a brief help on how to use the tool. Most tools that we are going to use provide this brief description of usage and parameters using this flag.
For *FastQC*, 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
../Scrivania/Dataset_Corso/2cells_1.fastq
```

The parameters are:

- `-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 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. Alternatively, in the folder `2cells_1_fastqc` 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 Levels	2cells_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. Open the file `fastqc/2cells_1_fastqc/ fastqc_data.txt`. Look near the beginning of the file to the report *Per base sequence quality.* Here, the first column reports the read position, while the second reports the mean quality at that position. Scroll down till the end of the report and check whether the mean quality is above 20 at each position.

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

```
$ fastqc -o fastqc -t 2 --extract --nogroup ../Scrivania/Dataset_Corso/2cells_2.fastq
$ fastqc -o fastqc -t 2 --extract --nogroup ../Scrivania/Dataset_Corso/6h_1.fastq
$ fastqc -o fastqc -t 2 --extract --nogroup ../Scrivania/Dataset_Corso/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.

4. Trimming

We will use *Trimmomatic* for read trimming and filtering, and adapters removal. Several other equally good trimming and adapter removal tools are available, but *Trimmomatic* is one of the most popular, being effective, fast and simple to use.

Let's first see its general usage:

```
$ java -jar /usr/share/java/trimmomatic.jar -h

Usage:
       PE [-version] [-threads <threads>] [-phred33|-phred64] [-
trimlog <trimLogFile>] [-quiet] [-validatePairs] [-basein
<inputBase> | <inputFile1> <inputFile2>] [-baseout <outputBase> |
<outputFile1P> <outputFile1U> <outputFile2P> <outputFile2U>]
<trimmer1>...
   or:
       SE [-version] [-threads <threads>] [-phred33|-phred64] [-
trimlog <trimLogFile>] [-quiet] <inputFile> <outputFile>
<trimmer1>...
```

You can see that there are two different ways of using *Trimmomatic*, one for paired-end reads (PE) and the other for single-end reads (SE).
For paired-end reads, 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, or both. In our case, we need to specify output file names that will remind you that these are trimmed *fastq* files, and that differentiate between reads that remained paired after trimming and those that do not.

First, verify that you are in the `tutorial` folder (using `pwd`). If you are somewhere else, go back to `tutorial`, for example using the command `cd $HOME/tutorial`. Now, to keep everything tidy, let's first create a folder for the trimmed *fastq* files:

```
$ mkdir trimmed_fastq
```

Now let's run the trimming for the 2-cells *fastq* files:

```
$ java -jar /usr/share/java/trimmomatic.jar \
PE \
-threads 2 \
-phred33 \
../Scrivania/Dataset_Corso/2cells_1.fastq \
../Scrivania/Dataset_Corso/2cells_2.fastq \
trimmed_fastq/2cells_1.trim.paired.fq \
trimmed_fastq/2cells_1.trim.unpaired.fq \
trimmed_fastq/2cells_2.trim.paired.fq \
trimmed_fastq/2cells_2.trim.unpaired.fq \
ILLUMINACLIP:/usr/share/trimmomatic/TruSeq3-PE.fa:2:30:10 \
SLIDINGWINDOW:4:20 \
LEADING:20 \
TRAILING:20 \
MINLEN:40 \
AVGQUAL:20
```

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
- `ILLUMINACLIP`: Instruct to remove adapter sequences, if found, and provides the path to a file containing adapter sequences.
- `SLIDINGWINDOW`: This command specifies that the average quality is computed for a window of a given length (4 nucleotides), starting from the read end and shifting one nucleotide at the time, and if this average is lower than a threshold (20) the first nucleotide of the window is removed.
- `LEADING`: quality threshold for removing nucleotides from the 5' end (set to 20)
- `TRAILING`: quality threshold for removing nucleotides from the 3' end (set to 20)
- `MINLEN`: minimum read length (set to 40 nucleotides)
- `AVGQUAL`: mean read quality threshold (set to 20)

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: 767134 (97,51%)
Forward Only Surviving: 15735 (2,00%) Reverse Only Surviving:
3398 (0,43%) Dropped: 475 (0,06%)
TrimmomaticPE: Completed successfully
```

Check the content of the output folder:

```
$ ls -l trimmed_fastq
```

Trimmomatic generates 4 files: one for the trimmed forward reads and one for the trimmed reverse reads, for all the cases where both members of the forward/reverse pair survived the trimming (these are the two files containing the word `paired`), plus two files for the forward and reverse reads that remained unpaired, meaning that one member of the pair was discarded (these are the two containing the word `unpaired`). Since many tools require that 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. We will use only the paired files, but remember that also the unpaired reads could be employed by treating them as single end reads.

For the 2-cells embryo data, the files `2cells_1.trim.paired.fastq` and `2cells_2.trim.paired.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). You should obtain the vast majority of reads survived the trimming, and most of them remained paired. This is not always the case, and the trimming can discard up to two-thirds, or even half, of the initial dataset. Nevertheless, it is better to work on fewer but reliable reads than on lots of reads that can possibly contain many sequencing errors.

It is always a good practice to check whether trimming has indeed removed some quality issues of the datasets. To verify that the trimming procedure worked, try to run again *FastQC* on the 2-cells embryo trimmed *fastq* files and compare the output reports against the raw *fastq* files. You can use a command like this:

```
$ fastqc -o fastqc -t 2 --extract --nogroup
trimmed_fastq/2cells_1.trim.paired.fq
```

Try also for the reverse reads from the 2-cells embryo sample. Note that now the reads do not have all the same length anymore.

Now try yourself: repeat the same steps for the other sample (the 6-houts embryo) in the `../Scrivania/Dataset_Corso` folder (NB: be careful on the input and output file names), and once finished check that in the `trimmed_fastq` 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).

In any case, the command line should look like this:

```
$ java -jar /usr/share/java/trimmomatic.jar \
PE \
-threads 2 \
-phred33 \
../Scrivania/Dataset_Corso/6h_1.fastq \
../Scrivania/Dataset_Corso/6h_2.fastq \
trimmed_fastq/6h_1.trim.paired.fq \
trimmed_fastq/6h_1.trim.unpaired.fq \
trimmed_fastq/6h_2.trim.paired.fq \
trimmed_fastq/6h_2.trim.unpaired.fq \
ILLUMINACLIP:/usr/share/trimmomatic/TruSeq3-PE.fa:2:30:10 \
SLIDINGWINDOW:4:20 \
LEADING:20 \
TRAILING:20 \
MINLEN:40 \
AVGQUAL:20 \
```

5. The tuxedo pipeline

The Tuxedo pipeline is the *de facto* standard for RNA-Seq data analysis. Several other tools and procedures have been proposed in the recent past, some more accurate and others more fast, but this pipeline is by far the most commonly employed, since it offers a good compromise between speed and reliability. The pipeline employs three software suites, *Bowtie*, *TopHat* and *Cufflinks*, each composed by a number of tools. *Bowtie* is a read mapper, that is a software that aligns reads to a genomic sequence. *TopHat* is a spliced aligner, that rescues reads corresponding to splicing junctions, that would be lost by a simple *Bowtie* run. *Cufflinks* use the *TopHat/Bowtie* mapping to estimate expression levels and perform differential expression tests.

5.1 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 *TopHat* for the transcriptome reconstruction, which in turn requires *Bowtie* to run for the read mapping to the genome. *Bowtie* 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/cuffdiff*, as you will see later.

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

```
$ 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`), which you can find in the `../Scrivania/Dataset_Corso` folder, and a label to identify the index (`bt2_index_base`), which will be the prefix of all files written by `bowtie2-build`. From the `tutorial` folder, 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):

```
$ cd $HOME/tutorial
$ mkdir bt2Index
$ cd bt2Index
```

Once you created and entered the folder, you can enter the command to index the genome:

```
$ bowtie2-build \
-f \
../../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.dna.fa \
ZV9
```

Parameters:

- `-f`: specify that the genome is in fasta format
- `../../Scrivania/Dataset_Corso/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, and, since you moved into the `bt2Index` folder, now the folder with the genome is two levels above)
- `ZV9`: the prefix that I chose for all files written by `bowtie2-build` (but you can chose a different one)

While the process is running, several log messages are written in the screen. Once the job is finished (it might take few minutes), 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.

5.2 Insert size and mean inner distance

In the case of paired-ended sequencing, the correctness and reliability of the read alignment to the genome is estimated by evaluating if paired reads align on the genome at the right distance and with the right orientation. For example, if the sequencing was made on genomic or cDNA fragments of around 300bp in length, and the reads are 100 nt long, then one can expect that when aligned to genome, the distance between the closest ends of the paired reads should be 300-100*2= 100nt.

The *insert size* is the average length of the sequenced fragments (300bp in this example), minus the adapters length, while the difference between the average insert size and twice the length of the reads is called the *mean inner distance* (100nt in this example). In the alignment step, all paired reads that do not align coherently with the insert size or the inner distance are generally discarded, since their alignment could be wrong. Often the insert size is known, since it depends on the sample preparation and fragment size selection, but can also be estimated by aligning a subset of paired reads to the genome, then computing the mean inner distance assuming that the majority of reads will align correctly. Let's do this.

To estimate insert size and mean inner distance, let's align the trimmed reads to the zebrafish genome using *Bowtie2*. This step might require some time (in our case around 5 minutes). Since at this point we don't know yet the insert size, we will tell bowtie to consider alignments as good in which read pairs can be at very close or very far (up to 1000 bp). Moreover, we will use parameter settings that will produce a fast (but likely less accurate) output.

As usual, let's first check the *Bowtie2* usage:

```
$ bowtie2 -h
```

You should get the following message:

```
Usage:
  bowtie2 [options]* -x <bt2-idx> {-1 <m1> -2 <m2> | -U <r> | -
-interleaved <i>} [-S <sam>]

  <bt2-idx>  Index filename prefix (minus trailing .X.bt2).
             NOTE: Bowtie 1 and Bowtie 2 indexes are not
compatible.
  <m1>       Files with #1 mates, paired with files in <m2>.
             Could be gzip'ed (extension: .gz) or bzip2'ed
(extension: .bz2).
  <m2>       Files with #2 mates, paired with files in <m1>.
             Could be gzip'ed (extension: .gz) or bzip2'ed
(extension: .bz2).
  <r>        Files with unpaired reads.
             Could be gzip'ed (extension: .gz) or bzip2'ed
(extension: .bz2).
  <i>        Files with interleaved paired-end FASTQ reads
             Could be gzip'ed (extension: .gz) or bzip2'ed
(extension: .bz2).
  <sam>      File for SAM output (default: stdout)

  <m1>, <m2>, <r> can be comma-separated lists (no whitespace)
and can be
  specified  many  times.    E.g.  '-U  file1.fq,file2.fq  -U
file3.fq'.
```

From this message, you can see that you need to specify where the genome index is, and which is the prefix of the index files. Then, you need to provide the paths of the paired-end reads. Several other options are available, but we will use just few of them.
Go back to the `tutorial` folder, and create, using `mkdir`, a folder where this analysis will be done, you can call it `insert_size` or any other name you like, then move into the folder using the `cd` shell command.

To run the process, type on the command line:

```
$ bowtie2 \
--minins 0 \
--maxins 1000 \
--very-fast \
-x ../bt2Index/ZV9 \
-1 ../trimmed_fastq/2cells_1.trim.paired.fq \
-2 ../trimmed_fastq/2cells_2.trim.paired.fq \
-S 2cells.sam
```

The parameters are:

- `--minins`: the minimum fragment length (set to 0 bp)
- `--maxins`: the maximum fragment length (set to 1000 bp)
- `--very-fast`: This choice is a preset of the many *Bowtie2* parameters to obtain

a fast, albeit not particularly accurate alignment.

- `-x:` path and prefix of the genome index
- `-1:` the forward reads file
- `-2`: the reverse reads file
- `-S`: the output file name

At the end of the run, some statistics will be printed on screen. The output file is in Sequence Alignment Map (*SAM*) format, which is the standard format to represent the alignment of reads to a genome in a compact way. The *BAM* format is the binary version of the (*SAM*) format, which is more often used since it has a smaller size. Let's have a look at a *SAM* file, using the `head` or `more` commands. After a few header lines (starting with @), each row describes a read alignment to the reference genome, reporting the read sequence and a string in a particular encoding (called *CIGAR*) describing how the read aligns to the genome (for example, if there are insertion, deletion or mismatches). For more information regarding the *SAM* format you can see `http://samtools.sourceforge.net/SAM1.pdf`

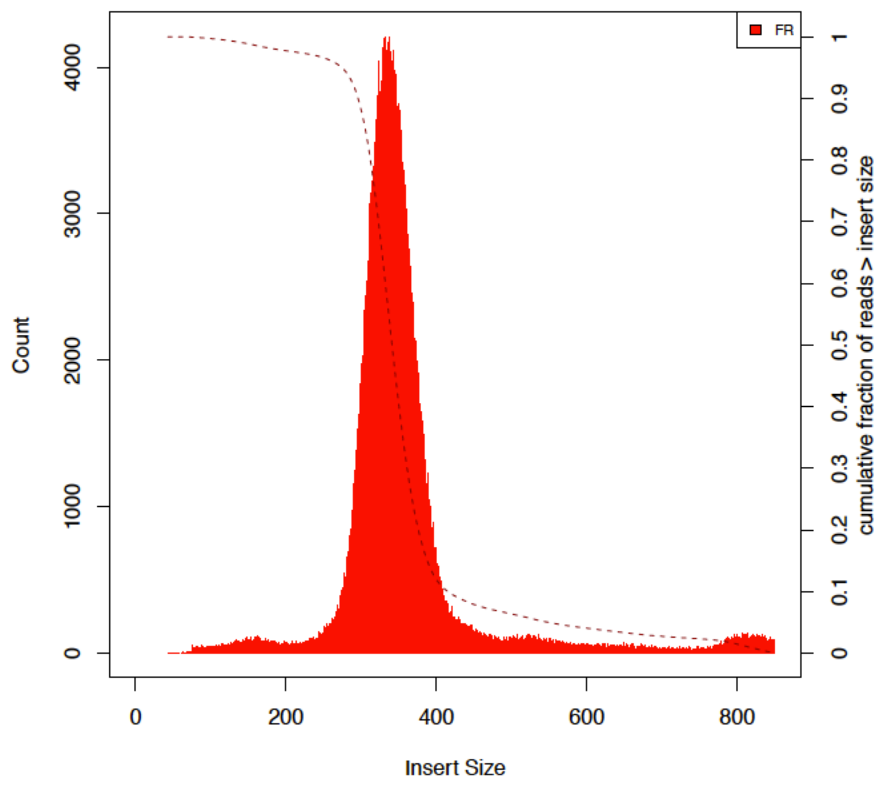
Now we need to compute the distribution of the distance between paired reads, its mean and standard deviation. Starting from the alignment, the insert size can be computed from the distance at which paired reads align on the genome. To do that, we will use a tool, called `CollectInsertSizeMetrics`, which is part of the *picard* suite, which is a very popular suite containing many utilities for handling and analyzing sequencing data. On your computer, *picard* can be found in a folder on your Desktop. First, we need to sort the *SAM* file by genomic coordinates, that is ordering the reads from the beginning of the chromosome to its end. To do that we will use another tool from *picard*, called `SortSam`. In the terminal, type the following:

```
$     java     -jar     /home/studente/Scrivania/Elixir-RNA-Seq-
Tools/picard.jar \
SortSam \
INPUT=2cells.sam \
OUTPUT=2cells.sorted.sam \
SORT_ORDER=coordinate
```

The parameters specify the input *SAM* file that you just obtained, the name of the output file, which is another file in *SAM* format, and the type of sorting (by genomic coordinate). Once finished, we are ready to compute the insert size statistics. Type the following:

```
$     java     -jar     /home/studente/Scrivania/Elixir-RNA-Seq-
Tools/picard.jar \
CollectInsertSizeMetrics \
I=2cells.sorted.sam \
O=2cells.insert_size_metrics.txt \
H=2cells.insert_size_histogram.pdf
```

The parameters specify the input file (the sorted *SAM* file), and the name of the two output files, one reporting the statistics, the other is a pdf file plotting the insert size distribution, which should look like this:



You should see a monomodal distribution (with only one main peak). The narrower the peak, the better. Some paired reads have a large insert size (> 800bp), but this can be due to the fact that we aligned the reads to the genome and not directly to the mature transcript sequences, hence these large insert sizes could be due to reads mapping to the sides of an intron.
Open the `2cells.insert_size_metrics.txt` file with a text editor. Here you can see the computed metrics and the values that are used to plot the histogram of the insert size. In the `#METRIC CLASS` section, you will see a list of metrics and below a list of values for these metrics. The mean insert size is the fifth of these values, while its standard deviation is the sixth. To compute the mean inner distance from the mean insert size, we need to subtract from this number the read length. Remember that our reads are 76 nucleotides long, The mean inner distance is equal to mean insert size – mean read length * 2. You should get a value around 206 nucleotides.

Now repeat the same procedure for the 6-hours embryo sample. You should get in the end an estimate of the mean inner distance and the standard deviation.

5.3 Read mapping using `TopHat2`

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. *TopHat2* employs *Bowtie2* to align the reads to the genome. The difference between *Bowtie* and *TopHat* is that, when reads come from RNA sequencing, you have to remember that splicing events can join exon sequences. When sequencing a mature mRNA, all reads corresponding to an exon-exon junction cannot be easily mapped on the genome, since one part of the read could map into an exon, and the rest of the read could map into a different exon, possibly separated by hundreds or thousands of nucleotides. *Bowtie2* cannot handle these cases, which are instead rescued by *TopHat2* and used to identify splicing junctions and, from these, splicing events.

Let's first check the tophat2 several parameters:

```
$ tophat2 --help
```

The general format of the tophat command is:

```
tophat [options] <bowtie_index> <reads1[,reads2,...]>
[reads1[,reads2,...]]
```

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).
Go back to the tutorial folder in your home (you should be now in the insert_size folder, hence you should go up one level), and there create a folder for the tophat2 output, and call it tophat_out. Now type the tophat2 command line for the 2-cells embryo data:

```
$ tophat2 \
-o tophat_out/2cells \
-N 1 \
-g 2 \
--library-type fr-unstranded \
-p 2 \
--mate-inner-dist 206 \
--mate-std-dev 99 \
--solexa-quals \
bt2Index/ZV9 \
trimmed_fastq/2cells_1.trim.paired.fq \
trimmed_fastq/2cells_2.trim.paired.fq
```

The parameters are:

- `-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.
- `-N:` number of allowed mismatches between the read and the reference genome
- `-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 tophat2 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)
- `-p`: number of threads
- `--mate-inner-dist`: the mean inner distance that you just computed (it should be 206 nucleotides for the 2-cells embryo sample reads)
- `--mate-std-dev`: the standard deviation of the insert size (that should be 99

bp for the 2-cells embryo sample reads; we should have computed the standard deviation of the inner distance distribution, but let's skip this)
- `--solexa-quals:` quality scale of the reads
- path and prefix of the genome index
- trimmed forward reads file
- trimmed reverse reads file

This is going to be the longest step of the pipeline, requiring few minutes with our data, but that could increase to many hours for larger datasets. While the job is running, a temporary folder will be created where large files will be written, then deleted after completion.
When the job is finished, look into the output folder. The file that you need is the `accepted_hits.bam`, where the read alignments are stored in a binary version (*BAM*) of the Sequence Alignment Map (*SAM*) format. The *BAM* files cannot be opened with a text editor, but, as we saw earlier, 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
```

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     :    726329
           Mapped   :    642863 (88.5% of input)
            of these:     38812 ( 6.0%) have multiple
   alignments (11212 have >2)
Right reads:
          Input     :    726329
           Mapped   :    640243 (88.1% of input)
            of these:     38695 ( 6.0%) have multiple
   alignments (11225 have >2)
88.3% overall read mapping rate.

Aligned pairs:     572712
     of these:      35158 ( 6.1%) have multiple alignments
                     3752 ( 0.7%) are discordant alignments
78.3% concordant pair alignment rate.
```

Now launch the `tophat2` 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:

```
$ tophat2 \
-o tophat_out/6h \
-N 1 \
-g 2 \
--library-type fr-unstranded \
-p 2 \
--mate-inner-dist 231 \
--mate-std-dev 53 \
--solexa-quals \
bt2Index/ZV9 \
trimmed_fastq/6h_1.trim.paired.fq \
trimmed_fastq/6h_2.trim.paired.fq
```

Once finished, check the `align_summary.txt` file, to verify the run outcome statistics.

5.4 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. You are going to provide to *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. Let's check its (rather long) list of parameters:

```
$ cufflinks --help
```

The general format of the `cufflinks` command is:

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

where the input is a file containing aligned reads (either in *SAM* or *BAM* format).
You are going to use `cufflinks` with the Ensembl annotation file for zebrafish, thus 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.

Go back to the `tutorial` folder and create a folder for the `cufflinks` output storage:

```
$ mkdir cufflinks_out
```

Now you are ready to run `cufflinks`. Run the following command to compute expression values for the 2-cells embryo sample:

```
$ cufflinks \
-o cufflinks_out/2cells \
-G ../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.gtf \
-b ../Scrivania/Dataset_Corso/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 annotations in order to estimate isoform expression.
- `-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 those reads mapping to multiple locations in the genome.
- `--library-type:` specify the employed sequencing strategy
- the reads alignment file

With your small dataset, it should not take more than few minutes. Then, try to run the same analysis for the 6-hours embryo data by yourself, before looking at the command below:

```
$ cufflinks \
-o cufflinks_out/6h \
-G ../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.gtf \
-b ../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.dna.fa \
-u \
--library-type fr-unstranded \
tophat_out/6h/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, by open it with a text editor. 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. Run the following command (all in one line):

```
$ sort -t$'\t' -r -g -k 10 \
cufflinks_out/2cells/genes.fpkm_tracking > \
cufflinks_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 gene, 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 cufflinks_out/2cells/genes.sorted.fpkm_tracking |
cut -f 1 > cufflinks_out/2cells/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. If DAVID complains that not all identifiers were recognized, go on anyway using the recommended option (using the Gene conversion tool). 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?

5.5 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. Go back to tutorial and create an output folder to store its results:

```
$ mkdir cuffdiff_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.
Now type the command to run cuffdiff:

```
$ cuffdiff \
-o cuffdiff_out \
-L ZV9_2cells,ZV9_6h \
-b ../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.dna.fa \
-u \
--library-type fr-unstranded \
--library-norm-method classic-fpkm \
../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.gtf \
tophat_out/2cells/accepted_hits.bam \
tophat_out/6h/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`
- `--library-norm-method`: the normalization method (FPKM is not the best method but it's faster)
- the annotation file
- the read mapping for the 2-cells embryo sample
- the read mapping for the 6-hours embryo sample

The run should take few minutes. At the end, 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 cuffdiff_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`. Run the following command (all in one line):

```
$ sort -t$'\t' -g -k 13 cuffdiff_out/gene_exp.diff  >
cuffdiff_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 15 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 the estimate and statistical significance of all genes expression changes 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 15 genes satisfying this condition are too few to have a meaningful 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 cuffdiff_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`. If DAVID complains that not all identifiers were recognized, go on anyway using the recommended option (using the `Gene conversion tool`).
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?

5.6 Transcriptome reconstruction without genome annotations

We previously run *Cufflinks* providing the zebrafish genome annotations, thus limiting the transcript reconstruction and expression estimation to the known genes and splicing isoforms. Yet, for many species the annotations can be incomplete or completely lacking. Also for well-studied and well-annotated species, such as human or zebrafish, there is always the possibility that some unknown gene or splicing variant can be expressed in your samples. To identify novel expression units you can run *Cufflinks* without providing it the current annotations, or asking the software to use the annotations as a guide only. Then, you might want to know how many novel genes or splicing variants *Cufflinks* was able to identify. To obtain this, you can compare the *Cufflinks* output with the reference annotations and retrieve all the novel features. Finally, you might also want to characterize what these hypothetical new genes are. You might then fetch these gene and transcripts sequences, and use *BLAST* or other database search tools to try to identify homologs of these novel genes. Let's count how many hypothetical novel genes can be found in our samples. First, create a folder for this, so you will not mix the results with the previous *Cufflinks* run. Call this folder `cufflinks_noref` or something like that. Then, run the following:

```
$ cufflinks \
-o cufflinks_noref/2cells \
-b ../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.dna.fa \
-u \
--library-type fr-unstranded \
tophat_out/2cells/accepted_hits.bam
```

The options that you used above are the same of the previous Cufflinks run, we are just not providing the GTF file containing the annotations. Once the run is finished, check the content of the output folder, which contains the 4 standard Cufflinks output files. The `genes.fpkm_tracking` file is the table storing gene expression values in the 2-cells embryo. As we saw before, the table reports gene name, its coordinates and the expression value measured in FPKM units (in the tenth column), among other things. The difference between this run and the previous one, in which we used the zebrafish annotation file, is that *Cufflinks* assign to the reconstructed genes a custom identifier in

the form `CUFF.#` for genes and `CUFF.#.#` for transcripts. This is because *Cufflinks* does not know the zebrafish gene coordinates, therefore cannot assign a gene identity to the reconstructed genes.
We would like to know how many of the reconstructed transcripts were already known and present on the zebrafish *GTF* annotation file, and how many are novel. To obtain this, we will use the `Cuffcompare` tool from the *Cufflinks* suite, which can compare two *GTF* files. In the output folder there is a file, `transcripts.gtf`, containing the coordinates of the transcripts reconstructed by *Cufflinks*. We will compare this file with the official zebrafish annotation file, to see what is in common and what it's not.

```
$ cuffcompare \
-r ../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.gtf \
-R \
-M \
-o cufflinks_noref/cuffcompare \
cufflinks_noref/2cells/transcripts.gtf
```

The parameters are:

- `-r`: the reference annotation file
- `-R`: ignore reference transcripts that are not overlapped by any reconstructed transcript
- `-M`: ignore single exon transcripts, both in the reference and in the reconstructed transcriptome (these are often artifacts)
- `-o`: output folder and prefix for the output files
- the reconstructed transcriptome

In the output folder `cufflinks_noref/2cells` you will get 6 *Cuffcompare* output files:

- `cuffcompare_stats`: This file contains a report of how similar are the unguided reconstructed transcripts to the reference annotations..
- `cuffcompare.transcript.gtf.refmap`: This file lists, for each reference transcript, which *Cufflinks* transcripts either fully or partially match it. The first and second column is the reference gene name and Ensembl identifier, the fourth column is the *Cufflinks* identifier for the reconstructed transcript. The third column (class code) reports if there full overlap (`=`) or partial overlap (`c`) between reference and reconstructed transcripts.
- `cuffcompare.transcript.gtf.tmap`: This file lists the most closely matching reference transcript for each *Cufflinks* transcript. The class codes in the third column that indicates the potential new features that the unguided Cufflinks run identified are: (`j`) novel splicing variant; (`u`) novel gene. The file also report the estimated expression levels (in FPKM) of the Cufflinks transcripts.
- `cuffcompare.combined.gtf`: This *GTF* file contains the union of the reference annotations and the *Cufflinks* unguided reconstructed transcripts. Have a look at all these files.
- `cuffcompare.loci`: This file reports the names and coordinates of all loci identified by the intersection of the reconstructed and reference genes.
- `cuffcompare.tracking`: This file reports the matching transcripts between the reconstructed and reference annotations.

If you want to identify and count how many potential new genes *Cufflinks* identified, you can look for those indicated by a `u` character in the `cuffcompare.transcript.gtf.tmap` file. To do that, we will use `awk`, which is a shell tool for the manipulation of text files. Run the following command (all in one line):

```
$ awk '$3=="u"'
cufflinks_noref/2cells/cuffcompare.transcripts.gtf.tmap >
cufflinks_noref/2cells/novel_genes
```

Here we instructed `awk` to look in the third column of the file (indicated as `$3`) selecting all fields identical (indicated by the `==` notation) to the character `u`. The output is redirected (using the `>` notation) to an output file. Open the output file to verify that the filtering produced the desired selection, and use the `wc -l` command to count how many they are. Remember that you can get more information about a particular shell tool by using the `man` command (run `man awk` at the prompt in this case).

As an optional exercise, you can repeat the whole procedure for the 6-hours embryo sample, and count how many potential new genes can be identified in this sample.

6. *De novo* assembly with `Trinity`

Trinity is a *de novo* transcriptome assembly algorithm. *Trinity* offers also modules for gene expression and differential expression estimation. *Trinity* is composed by three different modules: *Inchworm*, *Chrysalis* and *Butterfly*.
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 generates groups of reads sharing k-mers, which correspond to different genes, and organize them into a graph. The third one scans through these graphs, seeking paths corresponding to the different splicing variants encoded by the gene.
Other *de novo* transcriptome reconstruction algorithms exist, for example *Trans-Abyss*, but *Trinity* is generally more accurate even though has very large computational requirements, especially for memory. For our small dataset we should nevertheless be able to run *Trinity* in a reasonable amount of time.

First, go back to `tutorial` and create a folder for the `Trinity` run, and move into it:

```
$ mkdir trinity
$ cd trinity
```

The *Trinity* input can be one, two or more reads files in fastq or fasta format. Due to some *Trinity* requirements, we need to manipulate a bit the original fastq files, since the read names do not follow the conventions that *Trinity* expects. Mostly, we need to remove a space character from the read names, which in fact it's not usually present. To do this, we will employ `sed`, a shell tool that can manipulate text files, similar to `awk` that we used previously. Type and run the following commands:

```
$ sed -e 's/ /_/g' ../../Scrivania/Dataset_Corso/2cells_1.fastq
> 2cells_1.nospace.fq

$ sed -e 's/ /_/g' ../../Scrivania/Dataset_Corso/2cells_2.fastq
> 2cells_2.nospace.fq
```

Without going into many details, these commands remove a space character from the read names and change it into an underscore "`_`" character. Remember that you can get more information about a particular shell tool by using the `man` command (run `man sed` at the prompt in this case).

A copy of each file from the 2-cells embryo sample will be created into the folder in which you currently are, with this modification. To differentiate between these new files and the original ones, the new names contain the "`nospace`" string.

Now we are ready to run *Trinity*. Run the following command:

```
$ Trinity \
--seqType fq \
--max_memory 2G \
--left 2cells_1.nospace.fq \
--right 2cells_2.nospace.fq \
--CPU 2 \
--SS_lib_type FR
```

The parameters are:

- `--seqType`: specifies that the input is in fastq format
- `--max_memory`: the maximum amount of memory that can be used by Trinity (for more realistic cases, this value should be larger, depending also on how much RAM your computer has)
- `--left`: the forward reads file
- `--right`: the reverse reads file
- `--CPU`: number of threads (the larger this parameter is, the faster will the run be)
- `--SS_lib_type`: the type of paired-end sequencing (forward-reverse in our case)

Once the run is finished (it should take around fifteen minutes), check the content of the output folder `trinity_out_dir`. You will see many files, but the one that we are interested in is that containing the assembled transcripts in fasta format (`Trinity.fasta`). Let's now check whether *Trinity* assembled transcripts are real zebrafish mRNAs, or are just *Trinity* artifacts. Open `Trinity.fasta`, using any text editor, choose one transcript sufficiently long and copy (`Ctrl+c`) its sequence. Now open your internet 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?

7. Variant calling

In this part of the tutorial, you will use the read alignment for variant calling, i.e. to identify genomic variations (SNPs, small insertion and deletions) between the reference zebrafish genome and your sample. Variant 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, and can be confounded by RNA editing events, nevertheless in some cases it can offer valuable insights using data that you already have at hand. You will use a variant calling algorithm, *VarScan*, on the output of the *bowtie* mapping.

First, you need to go back to `tutorial` and create a new folder for this analysis (call it `variant_calling`, for example) and move into it. Then, we need a BAM file sorted by genomic coordinate. We already have a sorted alignment of reads to the genome, which we did for computing the insert size, but it is in SAM format. Let's then convert this SAM file to BAM using the `view` tool of the `samtools` suite:

```
$ samtools view \
-b \
-o 2cells.sorted.bam \
../insert_size/2cells.sorted.sam
```

The parameters are:
`-b`: require output in BAM format
`-o`: the output file name
the input SAM file

In your folder you should see a new BAM file. This file needs to be converted in the pileup format, reporting, for each genome nucleotide, the reads alignment.:

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

The `-f` parameter specifies the path to the genomic sequence, and the output is redirected to a new file. 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. The first column is the identifier of the reference genome sequence (the chromosome 12 in our case), the second is the coordinate, the third is the nucleotide at that position in the reference genome, the fourth reports how many reads align to that position, and the following columns report how each read aligns there (if there is a match, a mismatch, a deletion, and so on), using a particular encoding. You can look at `http://samtools.sourceforge.net/pileup.shtml` to have more information on the format.

You are now ready for the identification of variants. `Varscan` offers several tools, each having several parameters for setting the reliability of the variant identification. All variant calling algorithms face the problem of discriminating between true variations and sequencing errors. Run *VarScan* using the following command (all in one line):

```
$ java -jar ../../Scrivania/Elixir-RNA-Seq-
Tools/VarScan.v2.3.9.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 variant
- `--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

The input file is the pileup that you just obtained, and the output is redirected to a file. 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 variant, 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) variant, go to the UCSC Genome Browser and point it to the variant coordinate (remember to set `Jul 2010 (Zv9/danRer7)` as genome assembly). In which gene the SNP falls into?
Once you have the variants 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.

8. Duplicate removal

Most sequencing platforms are affected, to various degrees, to PCR artifacts. PCR amplification is required by most platforms for obtaining a stringer signal and to facilitate reading the sequences. Yet, this PCR amplification is supposed to amplify each nucleic acid fragment of the same amount. If a given fragment is amplified improperly, more than the others or if the amplified fragments do not remain local, this fragment will be counted more than once. In quantitative applications these PCR artifacts could affect the analysis outcome. For this reason, it is sometimes necessary to remove duplicated reads, i.e. reads that occur more than once in the dataset, with the assumption that it is very unlikely that in a large genome you will get the same identical read more than once. In the case of RNA-Seq this point is still debated, since reads are derived from the transcriptome instead of the genome, moreover not all transcripts are present in the RNA extract in the same quantity, due to different expression levels. For example, imagine a gene encoding for short transcripts expressed at high levels. The random shearing of these transcripts would produce identical fragments and identical reads with some not negligible frequency. Therefore, duplicates removals would artificially decrease the expression estimate for this gene.

Nevertheless, let's see how to remove duplicated reads from a sample. One approach would be to compare each pair of reads, cluster the identical or similar ones and then select one member from each cluster, but this would take too much time. The most common approach is to first map the reads to the genome, then consider duplicates all those reads mapping to the same position in the genome. Let's do this, using a tool, `rmdup`, in the `samtools` suite. To see its usage, run the following:

```
$ samtools rmdup
```

You can see that the input file is a bam file, from which `rmdup` will get the alignment position for each read. Similarly, also the output file is in bam format. Now, create a folder, calling it `duplicates`, and move into it. In this folder, run the following command:

```
$ samtools rmdup ../tophat_out/2cells/accepted_hits.bam
2cells.nodup.bam
```

The arguments are the input and output bam files, respectively. Ignore the program warnings. When completed, let's check how many reads were removed. Use the `samtools flagstat` tool to count the aligned read pairs in the output bam file:

```
$ samtools flagstat 2cells.nodup.bam
```

Now run `samtools flagstat` on the original *BAM* file:

```
$ samtools flagstat ../tophat_out/2cells/accepted_hits.bam
```

Compare in particular the *mapped* reads, and the reads *paired in sequencing* values, with the `flagstat` output for the original *BAM* file. How many were discarded?

9 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.

9.1 Genome indexing for STAR

First, you need to go back to the `tutorial` folder and to create a folder for the entire analysis, and one subfolder for storing the *STAR* indexed genome:

```
$ mkdir star
$ mkdir star/starIndex
```

Run the file the *STAR* indexing command for the zebrafish chromosome 12:

```
$ STAR --runThreadN 2 --runMode genomeGenerate \
--genomeFastaFiles \
../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.dna.fa \
--genomeDir star/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

9.2 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 star/starMapping
```

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 star/starIndex --
sjdbGTFfile ../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.gtf -
-outSAMtype BAM Unsorted --outFileNamePrefix
star/starMapping/2C.star --readFilesIn
trimmed_fastq/2cells_1.trim.paired.fq
trimmed_fastq/2cells_2.trim.paired.fq
```

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

Then, run the same for the 6-hours embryo data. Remember to select the right input files and to change the output file prefix (i.e. the `--outFileNamePrefix` parameter, for example into `starMapping/6H.star`). For example:

```
$ STAR --outSAMstrandField intronMotif --outFilterIntronMotifs
RemoveNoncanonical --runThreadN 2 --genomeDir star/starIndex --
sjdbGTFfile ../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.gtf -
-outSAMtype BAM Unsorted --outFileNamePrefix
star/starMapping/6h.star --readFilesIn
trimmed_fastq/6h_1.trim.paired.fq
trimmed_fastq/6h_2.trim.paired.fq
```

After both runs are completed, in each 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`, since it requires the *BAM* files to be sorted by genomic coordinates. Let's use `samtools` to get this. To sort the *BAM* file for the 2-cells embryo data, you can type:

```
$        samtools        sort        -O        bam        -o
star/starMapping/2C.starAligned.out.sorted.bam     -T     tmp
star/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
star/starMapping/6h.starAligned.out.sorted.bam     -T     tmp
star/starMapping/6h.starAligned.out.bam
```

The two *BAM* files that you created will be used as input for *Cuffdiff*. You can also take a look at the files ending in `starLog.final.out`, which contain information on the runs.

9.3 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 star/cuffdiff_star
```

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

```
$ cuffdiff -o star/cuffdiff_star -L ZV9_2cells,ZV9_6h -b
../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.dna.fa -u --
library-type fr-unstranded
../Scrivania/Dataset_Corso/Danio_rerio.Zv9.66.gtf
star/starMapping/2C.starAligned.out.sorted.bam
star/starMapping/6h.starAligned.out.sorted.bam
```

As we did 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 star/cuffdiff_star/gene_exp.diff  >
star/cuffdiff_star/gene_exp_sorted.diff
```

Look at the first few lines of the sorted file. If everything went as it should have, the first 19 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. How 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 star/cuffdiff_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?

10. Concluding remarks

In this tutorial you learned how to analyze RNA-Seq data using which can be considered the most commonly used pipeline. New tools and methods are available, and others might be developed in the near future, rendering some steps of this pipeline obsolete. Nevertheless, these pipelines are highly modular, since input and output file formats are standardized in a way that allows individual tools to be switched. For example, *STAR* is a relatively recent spliced aligner that some consider a better alternative to *TopHat*. You can easily include *STAR* in your pipeline instead of *TopHat* (*STAR* is not included in this Galaxy, but it is in many others), since its input are *FASTQ* files and its output is a *bam* file, just like *TopHat*. After using *STAR* for read mapping and splicing reconstruction, you can continue with the procedure using *Cufflinks* and *Cuffidiff* like we did today without any problems.

Several alternatives are available. The major issue that you might encounter is the lack of sufficient computational power for processing large datasets. A standard RNA-seq dataset would require a multiprocessor machine, and especially a large amount of RAM (at least 8Gb but likely more than that, 16Gb or 32Gb). If you (or your lab) do not have sufficient computational power, you can ask for CPU time and storage space to a specialized center. For example CINECA (`https://www.cineca.it`) provides such services to Italian universities and institutions through monthly calls.

Alternatively, you can use remote services, in which you can upload your data, set up the pipeline and run all needed steps using their computational power. For example, CINECA provides an RNA-Seq remote analysis service, *RAP*, through a web-based interface (`https://epigen.hpc.cineca.it/rap`). Galaxy (`https://usegalaxy.org`) is another popular remote service for the analysis of RNA-Seq data and many other NGS data types. Several Galaxy servers are freely available. Some are simple mirrors of the main Galaxy, other are specialized for particular applications. Most are intended for basic and applied research, others are mainly intended for teaching purposes. Some offers anonymous access, others requires a registration. No matter which Galaxy you are using, the basic structure and logic is always the same. For a large yet incomplete list of public Galaxy mirrors, see `https://galaxyproject.org/public-galaxy-servers/`

Clearly, using a remote service is less flexible than running the analysis on your own machines, and is more difficult to automatize for analyzing several datasets.

11. Some useful references

Collins JE, White S, Searle S & Stemple DL (2012) Incorporating RNA-seq data into the zebrafish Ensembl genebuild. Genome Res. 22(10):2067-78

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

Grabherr MG, Haas BJ, Yassour M, Levin JZ et al. (2011) Trinity: reconstructing a full-length transcriptome without a genome from RNA-Seq data. Nat. Biotechnol. 29(7):644-652

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

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

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

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R and 1000 Genome Project Data Processing Subgroup (2009) The Sequence alignment/map (SAM) format and SAMtools. Bioinformatics, 25, 2078-9

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

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

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

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