Read Mapping

Juan Galarza (juan.galarza@jyu.fi), February 2021.

Now that we have our assembled transcripts by Trinity, it is time to map the reads to them. But first some clarifying concepts.

Mapping vs Alignment

At this point you might have heard of mapping the reads and aligning the reads. These are often used interchangeably but differ fundamentally.

In the simplest of terms, mapping and alignment are way of arranging sequences of DNA, RNA, or protein to identify regions of similarity.

Mapping: Finds the **approximate origin** of a sequence. Where in the genome did the read originate.

Alignment: Finds the **exact difference** between two sequences.

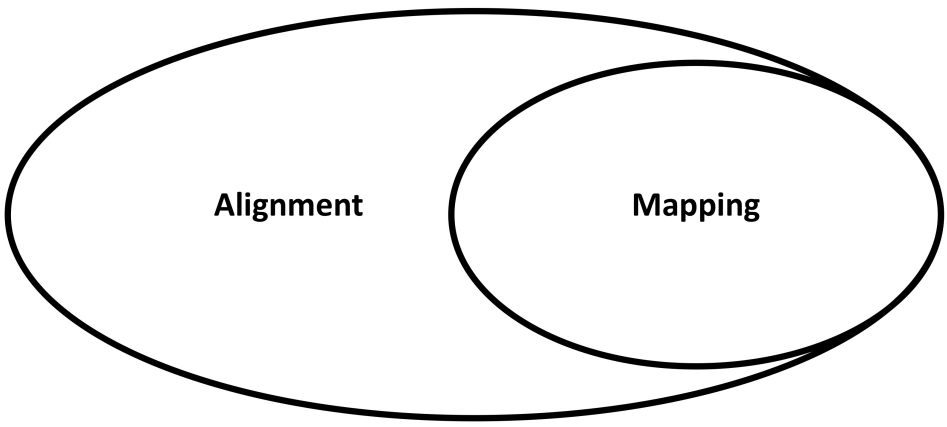
```
Alignment example:

GTGGTGCATCTGTTCTCCCCCGGCGGGAAGTACGACTCGCTGTATATG
|||||||||||||||_||_||___|||_||||||||||||||||||||
GTGGTGCATCTGTTTTCGCCAAACGGTAAGTACGACTCGCTGTATATG

Mapping example:

GTGGTGCATCTGTTCTCCCCCGGCGGGAAGTA   oqxB_EU370913
```

Mapping is part of alignment



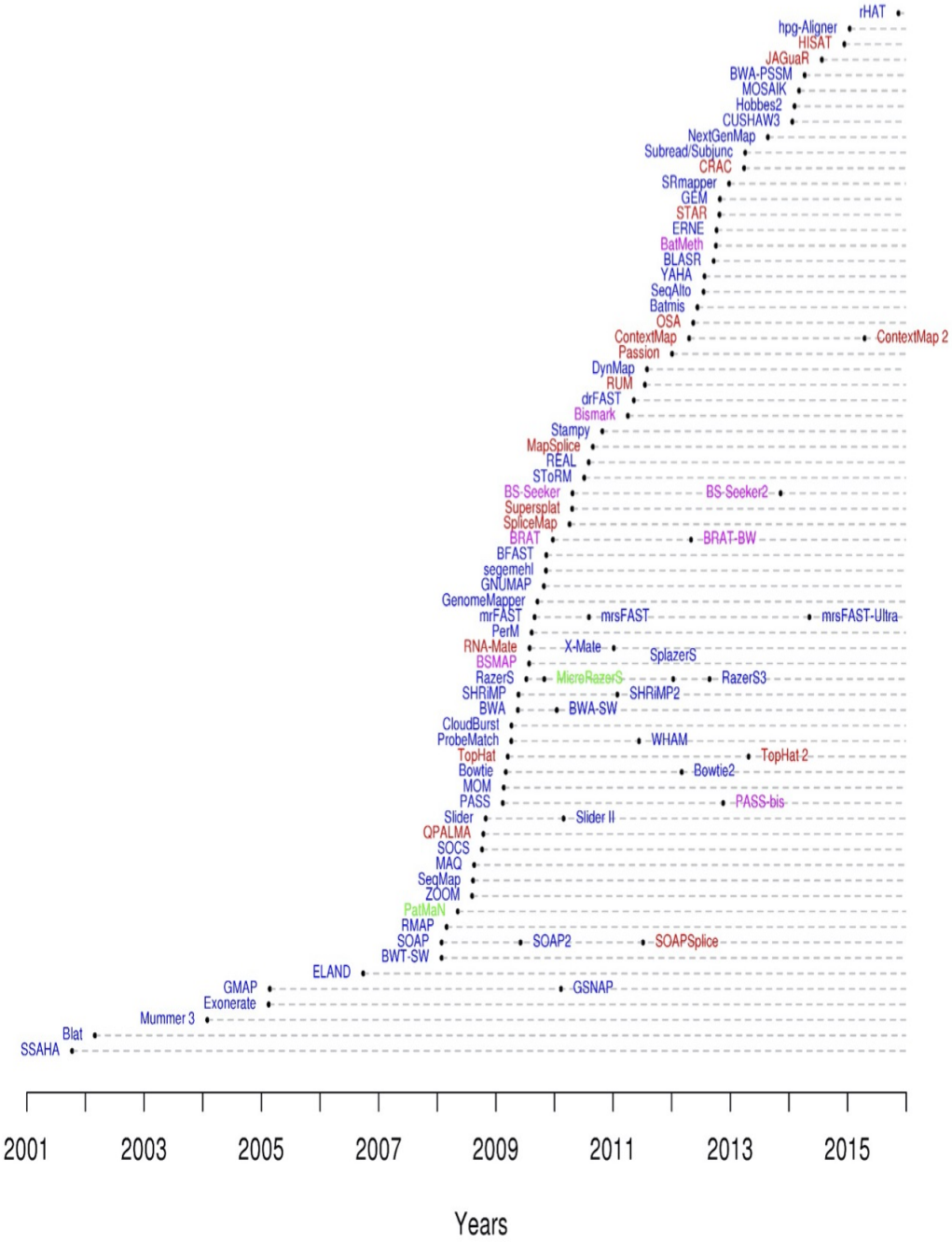
- A mapping is a region of the reference sequence where the read is placed
- A mapping is regarded to be correct if overlaps the true region

- An alignment is the detailed placement of each base of the read in the reference sequence
- An alignment is regarded to be correct if only if each base is placed correctly.

Currently there are over 90 NGS read aligners that differ in

- Speed
- Accuracy
- Approach
- Purpose

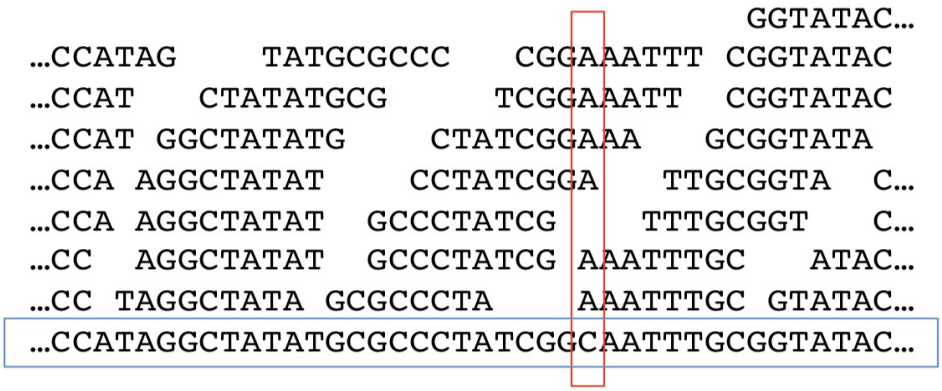
Choosing which to use not always straightforward.



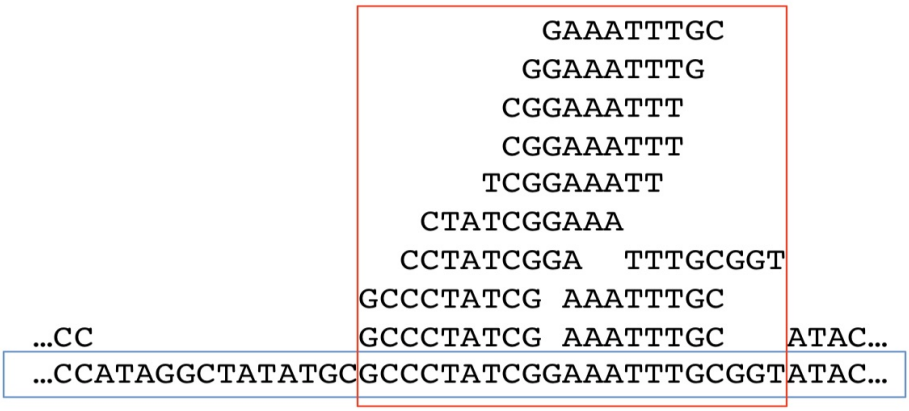
Why use mapping, if it is included in alignment?

1. Mapping is (much) faster, and computationally easier to solve, as we shall see below.
2. You might not be interested in the precise differences, but merely want to know the approximate origin.

Two of the main applications of mapping include genotyping, where the goal is to identify variations



and RNA-Seq, in which we want to identify and measure significant mapping peaks



Popular mapping methods:

1. KmerFinder
2. Kraken
3. Kallisto
4. Salmon
5. KMA-Sparse

Popular alignment methods:

1. BLAST
2. Bowtie2
3. BWA-MEM

4. GraphMap
5. MiniMap2
6. KMA

The mapping problem

Sequencing technologies nowadays are are capable of producing several millions of reads (Table 1), and with high throughput comes high error. The mapping process thus must be tolerant of mismatches, insertions, and deletions, in order to correctly distinguish between sequencing errors and genuine differences.

Platform	Reads/run	Accuracy
Roche 454	1 Million	99%
Illumina	~ 3 billion	99%
SOLiD	~1.4 billion	98%
Ion Torrent	< 5 million	98%
Pacific Biosciences	2 million	97%

Table 1. Basic specs of common Next-gen platforms.

Errors can originate from:

- Instrument weirdness
- Duplicate reads (PCR duplication)
- Indel errors (Skipping bases, inserting extra bases)
- Uncalled bases (Unreliable signal, replace with ”N”)
- Substitution errors (Reading wrong bases)

Why not just use BLAST)?

Assuming BLAST returns the result for a read in 1 sec, for 10 million reads: 10 million seconds = 116 days. More efficient algorithms are needed.

Mapping Strategies

Current mapping strategies can be broadly divided into two main categories: **Hash Table** (Lookup table, dynamic programming) FAST, but requires perfect matches.

Burrows-Wheeler Transform (BW Transform) FAST. Memory efficient. But for gaps/mismatches, it lacks sensitivity.

Both of these require that the reference genome (or transcriptome) to which the reads will be aligned (or mapped) be indexed. There are many ways to index a reference genome. In the simplest of terms, the index is a list of files that comprise the genome sequence, suffix arrays with corresponding positions, chromosome (or transcript) names and lengths, and splice junctions coordinates. You may think of an index as in a book index, where if you need to find a particular information, you don't look page by page for it. You look in the index first.

Index entries are generally in a form of `<key><value>`. Suppose that you wish to find the motif `TGC` in the sequence `GACTCGGATCTCGACATCG`. The corresponding index entry for this motif will be `345;789`. That is, the `key>TGC` has a value of `345;789`, which corresponds to positions `345` and `789` in the reference sequence. This index entry is stored in a table that can be quickly accesses, and thus, the aligner does not have to read the full sequence to find the motif, it access the table instead.

```
Pos: 01234567890123456789
Ref: GACTCGGATCTCGACATCG
Motif:  TCG          TCG
```

Let's start aligning!

Before you begin, make sure that you logged in to Puhti using -Y option to allow graphical output from the terminal.

```
ssh -Y USER@puhti.csc.fi
```

In the X directory we have a set of 6 biological samples

```
    LI101
    LI732
    LI74
    LI960
    LI961
    LI967
```

The samples are paired and contain 100,000 reads in each mate pair (i.e `SAMPLE_1.fq` and `SAMPLE_2.fq`). Chose one sample with both of its mates and copy it to your working directory.

```
cp LI101_100k_1.fq LI101_100k_2.fq /path/to/your/working/directory
```

From this point forward remember that text following '#' is a comment from me to you. You don't need to copy it to the terminal. Also remember that the back slash '\' tells the

terminal that the command continues in the next line. This helps to keep our code tidy. For instance:

```
# this is a comment
this is a command that continues \ # a comment can also be here
in the next line.
```

The alignment algorithm

Three of the main factors impacting alignment are the read quality, the read length, and of course, the alignment algorithm. Let's start with the alignment algorithm. Most aligners offer different choices of algorithms which must be set according to your question/expectations. One of the most popular aligners is Bowtie2

From Bowtie2 documentation we can read that it offers two types of alignment algorithms: End-to-end and Local. `bowtie2` takes a Bowtie2 index and a set of sequencing read files and outputs a set of alignments in SAM format.

The first step is to index the reference transcriptome that you built in the last session. This can take some time so I already indexed it using the `bowtie2-build` command.

End-to-end alignment versus local alignment

By default, Bowtie2 performs end-to-end read alignment. That is, it searches for alignments involving all of the read characters. This is also called an "untrimmed" or "unclipped" alignment.

When the `--local` option is specified, Bowtie2 performs local read alignment. In this mode, Bowtie2 might "trim" or "clip" some read characters from one or both ends of the alignment and in doing so maximises the alignment score.

End-to-end alignment example

The following is an "end-to-end" alignment because it involves all the characters in the read. Such an alignment can be produced by Bowtie2 in either end-to-end mode or in local mode.

```
Read:      GACTGGGCGATCTCGACTTCG
Reference: GACTGCGATCTCGACATCG

Alignment:
  Read:      GACTGGGCGATCTCGACTTCG
             |||||  |||||||||| |||
  Reference: GACTG--CGATCTCGACATCG
```

Where dash symbols represent gaps and vertical bars show where aligned characters match.

Local alignment example

The following is a "local" alignment because some of the characters at the ends of the read do not participate. In this case, 4 characters are omitted (or "soft trimmed" or "soft clipped") from the beginning and 3 characters are omitted from the end. This sort of alignment can be produced by Bowtie 2 only in local mode.

```
Read:      ACGGTTGCGTTAATCCGCCACG
Reference: TAACTTGCGTTAAATCCGCCTGG

Alignment:
  Read:      ACGGTTGCGTTAA-TCCGCCACG
                 ||||||||| ||||||
  Reference: TAACTTGCGTTAAATCCGCCTGG
```

Let's do an alignment with each mode and have a quick look at the results. See below the explanation for each of these parameters

```
bowtie2 -x Trinity_Index/Trin_index \
--end-to-end \
-t \
-p 2 \
-1 LI101_100k_1.fq \
-2 LI101_100k_2.fq \
-S LI101.end2end.sam \
2> LI101.end-to-end.metrics
```

Do another alignment using the local mode this time.

```
bowtie2 -x Trinity_Index/Trin_index \
--local \
-t \
-p 2 \
-1 LI101_100k_1.fq \
-2 LI101_100k_2.fq \
-S LI101.local.sam \
2> LI101.local.metrics
```

You can compare some basic alignment statistics in the `.metrics` files. Are there any differences in the number of aligned reads between the `local` and `end-to-end` alignments?. How are the `concordant` and `discordant` alignments?. Recall from previous lessons that a "paired-end" or "mate-pair" read consists of pair of mates, called mate 1 and mate 2. Pairs come with a prior expectation about (a) the relative orientation of the mates, and (b) the distance separating them on the original DNA molecule. In Bowtie2, a pair that aligns with the expected relative mate orientation and with the expected range of distances between mates is said to align `concordantly`. If both mates have unique alignments, but the alignments do not match paired-end expectations (i.e. the mates aren't in the expected relative orientation, or aren't within the expected distance range, or both), the pair is said to align `discordantly`.

Bowtie2 arguments

See full documentation here

-x	The basename of the index for the reference genome. The basename is the name of any of the index files up to but not including the final `.1.bt2` / `.rev.1.bt2` / etc. `bowtie2` looks for the specified index first in the current directory, then in the directory specified in the `BOWTIE2_INDEXES` environment variable.
-1	Comma-separated list of files containing mate 1s (filename usually includes `_1`), e.g. `-1 flyA_1.fq,flyB_1.fq`. Sequences specified with this option must correspond file-for-file and read-for-read with those specified in ``. Reads may be a mix of different lengths. If `-` is specified, `bowtie2` will read the mate 1s from the "standard in" or "stdin" filehandle.
-2	Comma-separated list of files containing mate 2s (filename usually includes `_2`), e.g. `-2 flyA_2.fq,flyB_2.fq`. Sequences specified with this option must correspond file-for-file and read-for-read with those specified in ``. Reads may be a mix of different lengths. If `-` is specified, `bowtie2` will read the mate 2s from the "standard in" or "stdin" filehandle.
-U	Comma-separated list of files containing unpaired reads to be aligned, e.g. `lane1.fq,lane2.fq,lane3.fq,lane4.fq`. Reads may be a mix of different lengths. If `-` is specified, `bowtie2` gets the reads from the "standard in" or "stdin" filehandle.
--end-to-end	In this mode, Bowtie 2 requires that the entire read align from one end to the other, without any trimming (or "soft clipping") of characters from either end. This is mutually exclusive with [`--local`]. `--end-to-end` is the default mode.
--local	In this mode, Bowtie 2 does not require that the entire read align from one end to the other. Rather, some characters may be omitted ("soft clipped") from the ends in order to achieve the greatest possible alignment score. The match bonus [`--ma`] is used in this mode, and the best possible alignment score is equal to the match bonus ([`--ma`]) times the length of the read. Specifying `--local` and one of the presets (e.g. `--local --very-fast`) is equivalent to specifying the local version of the preset (`--very-fast-local`). This is mutually exclusive with [`--end-to-end`]. `--end-to-end` is the default mode.
-p	Launch `NTHREADS` parallel search threads (default: 1). Threads will run on separate processors/cores and synchronise when parsing reads and outputting alignments. Searching for alignments is highly parallel, and speedup is close to linear. Increasing `-p` increases Bowtie 2's memory footprint. E.g. when aligning to a human genome index, increasing `-p` from 1 to 8 increases the memory footprint by a few hundred megabytes. This option is only available if `bowtie` is linked with the `pthreads` library (i.e. if `BOWTIE_PTHREADS=0` is not specified at build time).
--time	Print the wall-clock time required to load the index files and align the reads. This is printed to the "standard error" ("stderr") filehandle. Default: off.

--trim5	Trim N bases from 5' (left) end of each read before alignment (default: 0).
--trim3	Trim N bases from 3' (right) end of each read before alignment (default: 0).

We can further explore the alignment results using Samtools, which is a set of utilities that manipulate alignments in the SAM/BAM format. It imports from and exports to the SAM (Sequence Alignment/Map) format, does sorting, merging and indexing, and allows to retrieve reads in any regions swiftly.

Samtools has a variety of neat utilities, from which we will be using, `view` and `coverage` to explore our alignments. Start with the `view` command and relate the output to the sam format What are the fileds of your `.sam` file?. Identify them in the sam format descriptor.

```
# module load biokit (if you haven't load it yet)

samtools view LI101.local.sam | head
```

With the `coverage` command we can get more detailed information about our alignment. Let's have a peak of the output. What do the columns refer to, and which ones do you think are the most relevant to infer how good is our alignment?. You may see here for more detailed descriptions.

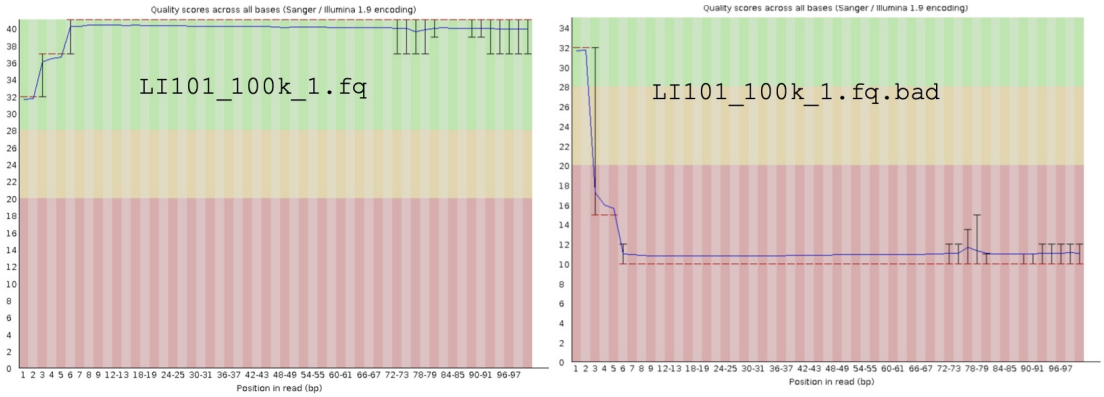
```
 samtools coverage LI101.end2end.bad.sam | head | column -t
```

Inspect the output

```
rname                     startpos  endpos  numreads  covbases  covera
TRINITY_DN74264_c0_g1_i2  1         655     4         199       30.3
TRINITY_DN84566_c2_g1_i19 1         5539    1         48        0.86
TRINITY_DN21378_c0_g1_i1  1         224     0         0         0
TRINITY_DN21326_c0_g1_i1  1         355     0         0         0
TRINITY_DN21358_c0_g1_i1  1         325     0         0         0
TRINITY_DN21300_c0_g1_i1  1         598     0         0         0
TRINITY_DN21328_c0_g1_i1  1         354     0         0         0
TRINITY_DN21388_c0_g1_i1  1         202     0         0         0
TRINITY_DN21388_c0_g1_i2  1         236     0         0         0
```

Read quality

Let's now look at the effect of read quality when performing alignments. Modern aligners make use of the quality of the reads being aligned to report some confidence in the alignment. In this particular read set, the sequencing company did the quality filtering and sent us reads of very high-quality. For the purpose of this practical, I artificially lowered the quality of the reads, keeping the read sequence intact.



In the **X directory** you will find the low-quality reads with a `.bad` suffix. Copy to your working directory the bad-quality sample file of your sample. For instance, if you choose `LI101_100k_1.fq` and `LI101_100k_2.fq`, copy the corresponding `LI101_100k_1.fq.bad` and `LI101_100k_2.fq.bad`. We will first check the quality of the reads.

Do the quality check with `fastqc` as you did in the previous practical.

```
# load the module that contains fastqc
module load biokit

# Create a directory to store the results
mkdir -p QC_reports

# run fastqc in all reads
for i in $(ls *.fq*); do \
fastqc --outdir QC_reports/ $i; done
```

Once `fastqc` finishes, we can combine all the results in a single report using MultiQC. Refer to the previous practical on quality control. Now run MultiQC

```
# we need to export these locales first
export LANG=C.UTF-8
export LC_ALL=C.UTF-8

# MultiQC can be launched with the following command

singularity exec --bind $PWD \ # Where $PWD is your current directory
/projappl/project_2000178/course-software.simg \ # where the software
multiqc . --fullnames --outdir QC_reports/ # calls the multiqc with c

# The result can be fetch from here and view in any browser.

scp -r USERNAME@puhti.csc.fi:/scratch/project_2000178/ECOS1179/Juan_Te
```

Have a look at how do the high- and bad-quality files compared.

Let's see if the bad-quality reads align as good as the high-quality ones.

```
bowtie2 -x Trinity_Index/Trin_index \
--end-to-end \
-t \
-p 2 \
-1 LI101_100k_1.fq.bad \
-2 LI101_100k_2.fq.bad \
-S LI101.end2end.bad.sam \
2> LI101.end-to-end.bad.metrics
```

As before, compare the `.metrics` file paying particular attention to the discordant and concordant alignments. What do you see as the main difference?.

Another important alignment metric is the mapping quality or MAPQC. Generally, the quality of a particular alignment is reported as –10 log10 Pr{mapping position is wrong}, rounded to the nearest integer. In plain english this means that, for instance, if the probability of a correctly mapped read is 0.99, then the corresponding MAPQ score should be 20 (i.e. log10 of 0.01 * -10). If the probability of a correct mapping is 0.999, the MAPQ score would increase to 30, and so on. This mapping quality is reported for each alignment in the fifth column of the `.sam` file that we generated above.

Let's extract this qualities to a file

```
samtools view LI101.end2end.sam | cut -f5 > LI101.end2end.mapqc
```

and make a quick histogram using R without leaving the terminal to inspect the distribution of quality values.

```
R --vanilla --slave -e 'x<-read.csv("LI101.end2end.mapqc");pdf("LI101_
```

If you logged in to puhti with ssh -Y as instructed above, you can view the plot directly from your terminal using

```
evince LI101_end2end_mapqc.pdf
```

Otherwise you can download the plot to your computer as

```
scp -r USERNAME@puhti.csc.fi:/path/to/LI101_end2end_mapqc.pdf .
```

Try inspecting the mapping quality indices of the different bad and high-quality samples to see what impact the read quality makes on the mapping, and thus inturn, in the alignment. Note that in this case, mapping is part of the alignment as we have discussed above.

Read length

Let's now mess with the read length. When planing a sequencing project several options and platforms exist for obtaining sequence reads of different length. The read length is

often chosen depending on the application (i.e. de novo assembly, gene expression, etc). To see the effect of read length in alignment we can use the options `--trim5` and `--trim3`. These tell Bowtie2 to ignore 50bp (or as many as you wish) from the 5' or 3' of the high-quality reads before the alignment.

```
bowtie2 -x Trinity_Index/Trin_index \
--end-to-end \
-t \
-p 2 \
--trim5 50 \
-1 LI101_100k_1.fq \
-2 LI101_100k_2.fq \
-S LI101.end2end.trim50.sam \
2> LI101.end-to-end.trim50.metrics
```

Continue playing with combinations of these parameters. Which ones do you think have the biggest impact?. What if you use only one mate of the pair (i.e. using the read*1 or read*2 as single reads with the `-U` option). Do you get comparable results as when using both mates?.

Expression tables

Ultimately, for downstream analyses of gene expression we will need count tables, also know as expression matrices. In essence, these show how many reads aligned to a particular transcript of our reference transcriptome. Trinity, the software you used before, conveniently provides perl scripts to generate such tables. The script `align_and_estimate_abundance.pl` can be called as below to produce an expression table.

```
singularity exec --bind $PWD \
/projappl/project_2000178/course-software.simg \
/usr/local/bin/trinityrnaseq/util/align_and_estimate_abundance.pl \
--transcripts Trinity_Index/TrinityB.fasta \
--seqType fq \
--left LI101_100k_1.fq \
--right LI101_100k_2.fq \
--est_method RSEM \
--output_dir LI101_expr_Bowtie2 \
--aln_method bowtie2 \
--trinity_mode
```

After running the above command we can inspect the results printed to the `RSEM.genes.results` file.

```
less LI101_expr_Bowtie2/RSEM.genes.results
```

gene_id	transcript_id(s)	length	effective_
TRINITY_DN100000_c0_g1	TRINITY_DN100000_c0_g1_i1	241.00	73.51
TRINITY_DN100001_c0_g1	TRINITY_DN100001_c0_g1_i1	325.00	154.58
TRINITY_DN100002_c0_g1	TRINITY_DN100002_c0_g1_i1	273.00	103.66
TRINITY_DN100003_c0_g1	TRINITY_DN100003_c0_g1_i1	203.00	40.60

The meaning of this output is as follows: **transcript_id** is the transcript name of this transcript. **length** is this transcript's sequence length. **effective_length** counts only the positions that generate a valid fragment. **expected_count** is the sum of the posterior probability that each read comes from this transcript over all reads. **TPM** stands for Transcripts Per Million, a relative measure of gene expression. It tells how many transcripts were found per every million bases sequenced. **FPKM** stands for Fragments Per Kilobase of transcript per Million mapped reads. It is another relative measure of transcript abundance. Which of these read count measures would you choose for gene expression analyses? why?.

Let's do some mapping now!

So far we have been looking at the alignment approach with Bowtie2. We can now have a look at the mapping approach using Kallisto and compare both approaches.

Kallisto is a program for quantifying abundances of transcripts from bulk and single-cell RNA-Seq data, or more generally of target sequences using high-throughput sequencing reads. It is based on the novel idea of pseudoalignment for rapidly determining the compatibility of reads with targets, without the need for alignment. You may find the algorithm details here.

From kallisto documentation we can read that the basic idea is to determine, for each read, not where in each transcript it aligns, but rather which transcripts it is compatible with. As such, it’s NOT necessary to do a full alignment of the reads to the genome which is often the slowest step in sequencing analysis. Instead, the raw sequence reads are directly compared to transcript sequences and then used to quantify transcript abundance. Put simply, it estimates the approximate location in the transcriptome from where the read originate. Sounds familiar?

As with Bowtie2, kallisto needs an index to look in. And as with Bowtie2, this can take a long time. So in the interest of time, I already generated the Kallisto index with (unsurprisingly) the `index` command.

Let's run Kallisto on the same samples that we ran with Bowtie2.

```
singularity exec --bind $PWD \
/projappl/project_2000178/course-software.simg \
kallisto quant --index Kallisto_Index/TrinityB.idx \
--output-dir Kallisto_LI101/ \
--threads 1 LI101_100k_1.fq LI101_100k_2.fq
```

Enter the Kallisto results directory (mine is Kallisto_LI101) and inspect the `abundance.tsv` file and compare it to the R `SEM.genes.results` from Bowtie2 above.

```
less LI101_expr_Bowtie2/RSEM.genes.results
less Kallisto_LI101/abundance.tsv
```

From the Kallisto output we can see that the table is not sorted as the Bowtie2 table. Let's fix that.

```
cat Kallisto_LI101/abundance.tsv | awk 'NR == 1; NR > 1 {print $0 | "s
```

You could also compare expression values between from the alignment and mapping approaches for individual transcripts. Suppose that the transcripts TRINITY*DN100000*c0*g1*i1 and TRINITY*DN100006*c0*g1*i1 have a very interesting biological function (we will see functional annotation in the next practical), or are involved in some process that you find particularly interesting for your experiment. You could quickly compare the expression values as below:

```
grep -E "TRINITY_DN100000_c0_g1_i1|TRINITY_DN100006_c0_g1_i1" LI101_e
```

What general conclusions can you draw about what to consider when performing read mapping for downstream gene expression?.

In the next practical you will perform differential gene expression analysis, for which you will need biological replicates. In the directory "split_800k/" you will find paired end read sets corresponding to six bank vole individuals (=biological replicates). Move all files that end in the same suffix (e.g. .aa, or .ab) in your directory using a wildcard.

```
mv *.aa /path/to/your/directory
```

You should now have a total of 12 files: 6 biological replicates times two, as it is paired-end data. Use the `align_and_estimate_abundance.pl` script to map to TrinityB assembly and count expression levels. Do this to all six of the replicates to produce six individual `RSEM.isoform.results`. Note that you can use a sample list so you don't have to input the command separately for each instance! Check the help file in the script on how to use the option.

```
align_and_estimate_abundance.pl -h
```

Once you have your counts, use the `abundance_estimates_to_matrix.pl` script to consolidate the counts into a single expression matrix. For this step and onward you will need to load the `R` environment at CSC.

```
module load r-env-singularity
```

Then run the script as below

```
$TRINITY_HOME/util/abundance_estimates_to_matrix.pl\
--est_method RSEM \
--gene_trans_map genetransmap \
--name_sample_by_basedir sample1.isoform.results \
sample2.isoform.results \
sample3.isoform.results \
sample4.isoform.results \
sample5.isoform.results \
sample6.isoform.results
```