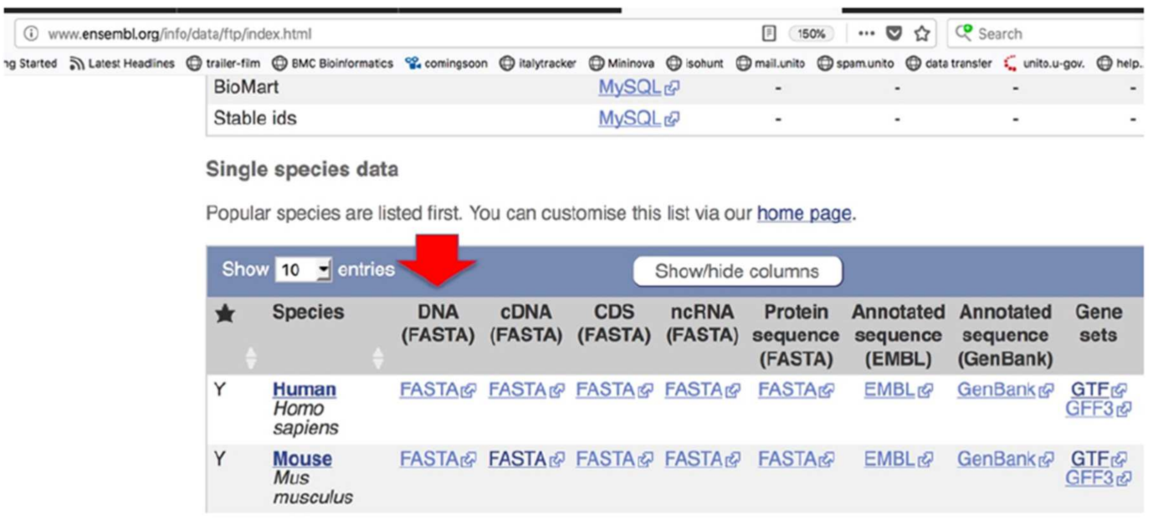
ALIGNMENT

Alignment requires: FASTQ files to (reads) to be aligned, a reference genome (to do genome mapping with the alignment algorithm), an alignment algorithm (the gold standard for transcriptomic analyses is **STAR**), and annotation info (using .gff or .gtf files → using **RSEM**, basically we do transcript identification and counting: different transcripts isoforms for each gene, and these are counts).

A unique reference genome, thus a single alignment, is no longer informative to retrieve information: info coming from all alternative versions (annotations?) of the human genome are necessary to produce interesting data.

The reference genome alone is a mere FASTA file, but to perform alignment and analysis the annotation files are necessary: these are stored mainly on **Ensembl** and **UCSC** repositories.

- **<u>Ensembl</u>** → European repository, <u>gene-centric</u> database focused on gene loci coordinates. Results are reproducible as Ensembl stores all different annotations and updates don't replace previous ones. Most used database, it's easier to have annotations referred to genes and then extrapolate isoforms than doing the opposite.



www.ensembl.org/info/data/ftp/index.html

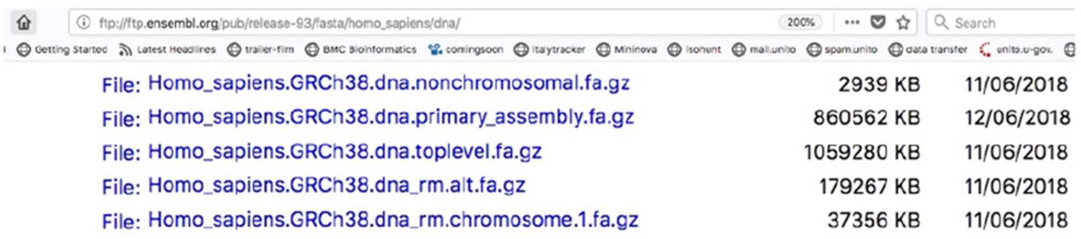
BioMart	MySQL	-	-	-	-
Stable ids	MySQL	-	-	-	-

Single species data

Popular species are listed first. You can customise this list via our home page.

Show 10 entries Show/hide columns

★	Species	DNA (FASTA)	cDNA (FASTA)	CDS (FASTA)	ncRNA (FASTA)	Protein sequence (FASTA)	Annotated sequence (EMBL)	Annotated sequence (GenBank)	Gene sets
Y	Human *Homo sapiens*	FASTA	FASTA	FASTA	FASTA	FASTA	EMBL	GenBank	GTF GFF3
Y	Mouse *Mus musculus*	FASTA	FASTA	FASTA	FASTA	FASTA	EMBL	GenBank	GTF GFF3



ftp://ftp.ensembl.org/pub/release-93/fasta/homo_sapiens/dna/

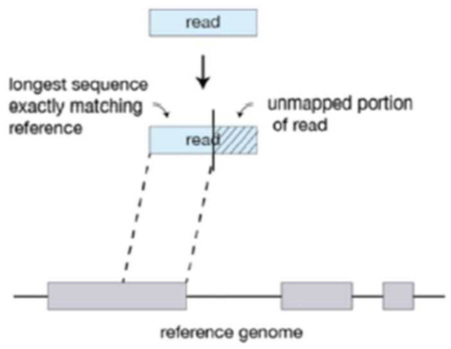
File: Homo_sapiens.GRCh38.dna.nonchromosomal.fa.gz	2939 KB	11/06/2018
File: Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz	860562 KB	12/06/2018
File: Homo_sapiens.GRCh38.dna.toplevel.fa.gz	1059280 KB	11/06/2018
File: Homo_sapiens.GRCh38.dna_rm.alt.fa.gz	179267 KB	11/06/2018
File: Homo_sapiens.GRCh38.dna_rm.chromosome.1.fa.gz	37356 KB	11/06/2018

<u>**Repository of Ensembl**</u>. The data used to perform mapping is the **DNA (FASTA)**. Click on DNA (FASTA) → a list of files appears (bottom window). In particular, the third file from the top (Homo_spaiens.GRCh38.dna.toplevel.fa.gz) contains all the standard genomes plus all the alternative variants that have been released over the years by sequencing different genomes. Conventional expression data analysis does not require this file, it is sufficient to use the second file from the top (**primary_assembly**), because the primary assembly is the one that contains all the coding information, and also because to generate the reference that will be used by the aligner software it is necessary to have memory on the RAM (and the second file is smaller than the third one).

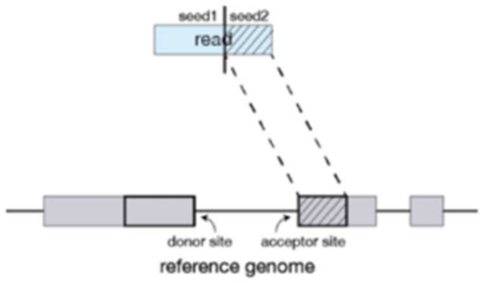
- **<u>UCSC</u>** → American database, <u>transcript-centric</u> as the gene loci are consequence of all transcripts associated to chromosomal positions. Data from UCSC are not very reproducible: you can get a snapshot of the annotation but every time the database is updated that snapshot cannot be reproduced anymore.

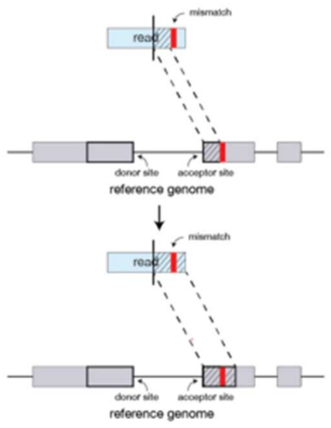
GENOME MAPPING: STAR

STAR is a RNA-seq gapped aligner (as eukaryotic genes have exons and introns) that allows to perform genome mapping. STAR is the gold standard owing to its speed in mapping, which is only limited by the disk (speed → hundred million reads per hour *vs* TopHat2 that aligns only 8-10 million reads). Despite the speed, STAR retains a remarkable precision. Obviously it doesn't work properly on sequences lacking introns. *How does STAR work?*



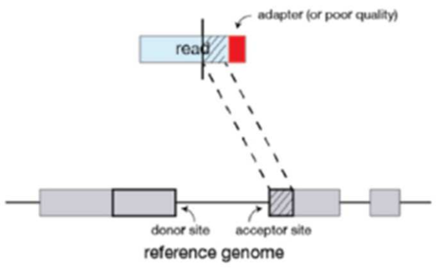
Maximal mappable prefixes (MMPs) are considered: you have a starting read and you consider the biggest part of it that perfectly matches on 1+ positions of the reference sequence (and since reads come from cDNA fragments they will match to exons only), and this part is called MMP indeed. MMPs are the 'seeds' (this first MMP will be *seed1*) and for this reason this step goes under the name of **seed searching**. The remaining portion of the read which is not matched anywhere is called the unmapped portion and will match with another exon far from the one where the neighbouring MMP was matched. This unmapped portion is cut from the *seed1* and matched somewhere else in the reference sequence, and it will take the name of *seed2*. This second match isn't searched everywhere but within hundreds kb from the *seed1* (as it will belong to the same gene), so the research is shrunk a lot.





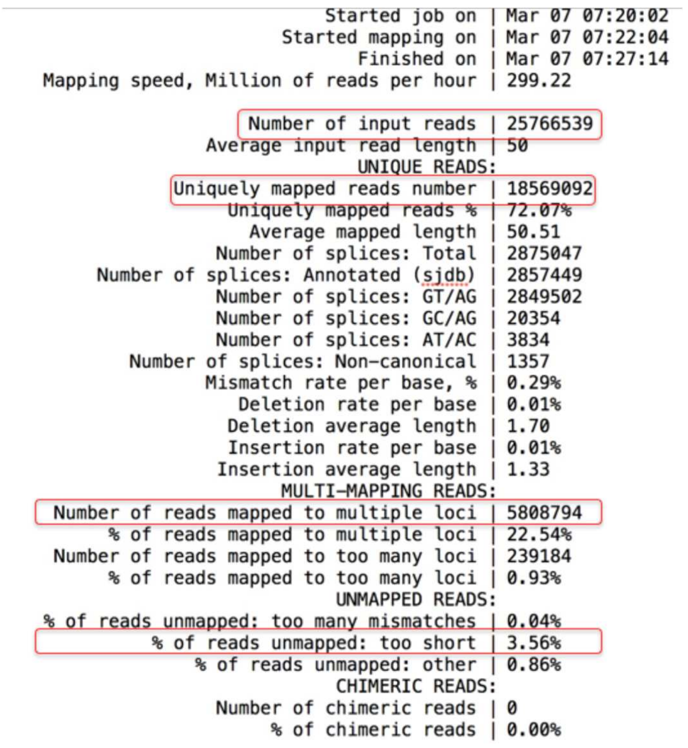
If STAR does not find an exact matching sequence for each part of the read, due to mismatches or indels, the previous MMPs will be extended.

What happens if there are some mismatches in the alignment and matches are not perfect? If a portion of the read is not matched perfectly but shows a mismatch, the piece of read is split in 2 parts: one part before the mismatch position, the second one after the mismatch. The algorithm tries matching the first part and if the match found is perfect (i.e. if a MMP is obtained), the mismatch is ignored and the second portion is placed immediately after by extending the previous MMP. If the extension doesn't give a good alignment, soft clipping of the poor quality sequence.



Poor quality sequences could also be represented by adapters, which do not map anywhere on the reference genome of course. These portions of the read would cause a dramatic drop in the quality evaluation of the mapping if considered, thus they are discarded during the alignment (e.g., a read of 150 nucleotides where 75 belong to the adapter: only 50% would match, so those 75 adapter nucleotides are ignored). By removing the adapters you also know how big the insert was.

STAR follows this approach **read by read**. The output of the mapping performed by STAR is the following (logfinal.out file in the dataset1 folder), which is among those stuff read every time a MultiQC analysis is launched:



```
                             Started job on | Mar 07 07:20:02
                         Started mapping on | Mar 07 07:22:04
                                Finished on | Mar 07 07:27:14
   Mapping speed, Million of reads per hour | 299.22

                      Number of input reads | 25766539
                  Average input read length | 50
                                UNIQUE READS:
               Uniquely mapped reads number | 18569092
                    Uniquely mapped reads % | 72.07%
                      Average mapped length | 50.51
                   Number of splices: Total | 2875047
        Number of splices: Annotated (sjdb) | 2857449
                   Number of splices: GT/AG | 2849502
                   Number of splices: GC/AG | 20354
                   Number of splices: AT/AC | 3834
           Number of splices: Non-canonical | 1357
                  Mismatch rate per base, % | 0.29%
                     Deletion rate per base | 0.01%
                    Deletion average length | 1.70
                    Insertion rate per base | 0.01%
                   Insertion average length | 1.33
                         MULTI-MAPPING READS:
    Number of reads mapped to multiple loci | 5808794
         % of reads mapped to multiple loci | 22.54%
    Number of reads mapped to too many loci | 239184
         % of reads mapped to too many loci | 0.93%
                              UNMAPPED READS:
   % of reads unmapped: too many mismatches | 0.04%
             % of reads unmapped: too short | 3.56%
                 % of reads unmapped: other | 0.86%
                              CHIMERIC READS:
                   Number of chimeric reads | 0
                        % of chimeric reads | 0.00%
```

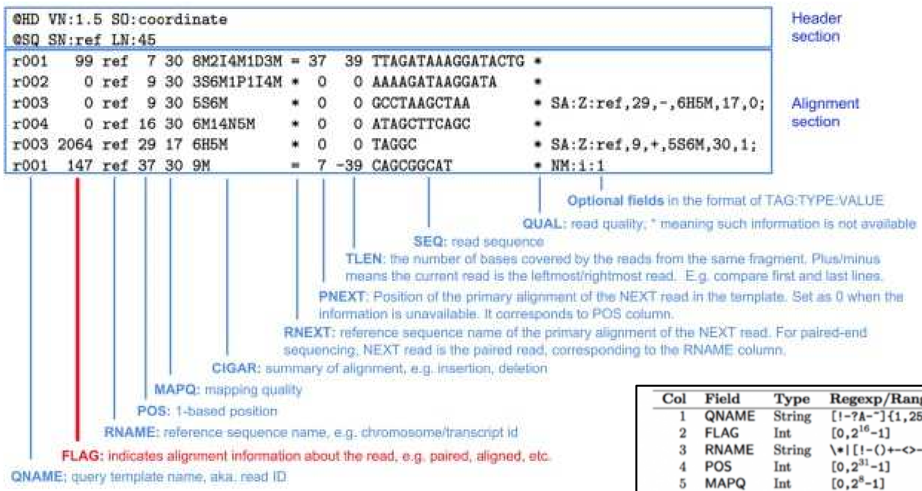
Number of input reads: there are ~25 million reads. Their average length is 50 nucleotides. **Uniquely mapped reads %** = 72.07%, quite low considering that reads come from coding genes (exonic regions → transcriptomic experiment), ~90% would be expected. Why then?

- % of reads mapped to multiple loci (22.54%) → deriving from noncoding genes as coding ones would map univocally on the reference. Likely due to DNA contamination, so to a poor cleaning of the sample.

- % of reads too short to be mapped (3.56%) → too short reads map to multiple positions in the genome, but these reads are likely derived from rRNAs that weren't cleaned from the sample (rRNA contamination this time), as rRNA are shorter than mRNA. Since on the reference there are no ribosomal genes and only nonrepetitive sequences are present, the majority of these short reads don't map anywhere.

Also a mycoplasma contamination, or a cross-contamination working with different cell lines, may cause a drop in the uniquely mapped reads %.

Results of a genome mapping are stored in **BAM** files (binary version of the tab-delimited SAM file), which contain all mapping information of the peaks of reads in specific areas of the genome (that should be associated to isoforms and genes → **RSEM** is used for this).



```
@HD VN:1.5 SO:coordinate
@SQ SN:ref LN:45
r001   99 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
r003 2064 ref 29 17 6H5M       *  0   0 TAGGC             * SA:Z:ref,9,+,5S6M,30,1;
r001  147 ref 37 30 9M         =  7 -39 CAGCGGCAT         * NM:i:1
```

Optional fields in the format of TAG:TYPE:VALUE

QUAL: read quality; * meaning such information is not available

SEQ: read sequence

TLEN: the number of bases covered by the reads from the same fragment. Plus/minus means the current read is the leftmost/rightmost read. E.g. compare first and last lines.

PNEXT: Position of the primary alignment of the NEXT read in the template. Set as 0 when the information is unavailable. It corresponds to POS column.

RNEXT: reference sequence name of the primary alignment of the NEXT read. For paired-end sequencing, NEXT read is the paired read, corresponding to the RNAME column.

CIGAR: summary of alignment, e.g. insertion, deletion

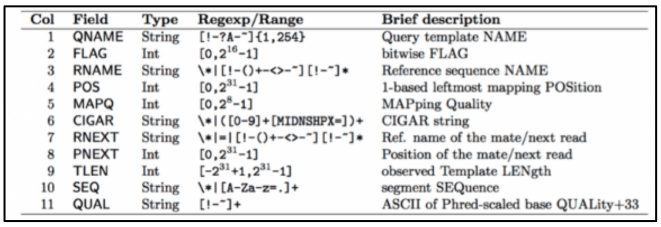
MAPQ: mapping quality

POS: 1-based position

RNAME: reference sequence name, e.g. chromosome/transcript id

FLAG: indicates alignment information about the read, e.g. paired, aligned, etc.

QNAME: query template name, aka. read ID

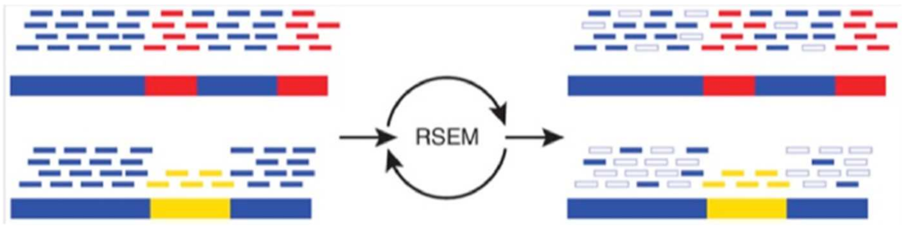


Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	[!-?A-~]{1,254}	Query template NAME
2	FLAG	Int	$[0,2^{16}-1]$	bitwise FLAG
3	RNAME	String	*|[!-()+-<>-~][!-~]*	Reference sequence NAME
4	POS	Int	$[0,2^{31}-1]$	1-based leftmost mapping POSition
5	MAPQ	Int	$[0,2^{8}-1]$	MAPping Quality
6	CIGAR	String	*|([0-9]+[MIDNSHPX=])+	CIGAR string
7	RNEXT	String	*|=|[!-()+-<>-~][!-~]*	Ref. name of the mate/next read
8	PNEXT	Int	$[0,2^{31}-1]$	Position of the mate/next read
9	TLEN	Int	$[-2^{31}+1,2^{31}-1]$	observed Template LENgth
10	SEQ	String	*|[A-Za-z=.]+	segment SEQuence
11	QUAL	String	[!-~]+	ASCII of Phred-scaled base QUALity+33

SAM file (.sam) structure. SAM files are the first products of the alignment process, while the BAM are derived (same content but in a more accessible format to programs used to call variants or to graphically display the mapping of reads on the reference). We can say the SAM file is readable by humans while the BAM file is readable by the computer.

RSEM

RSEM performs transcript quantification from RNA-seq data → it quantifies gene and isoform abundances (gene quantification is obtained collapsing the counts of transcripts belonging to the same gene).



When we have to quantify isoforms, the problem is represented by shared regions (in blue). Thus, RSEM estimates the relative expression of the different isoforms based on those areas that are isoform-specific (red and yellow areas) → the red area has nearly twice reads than the yellow area (~2:1 ratio). RSEM considers the common (blue) reads by assigning to the yellow isoform a number X of these reads while to the red one a number 2X (?)

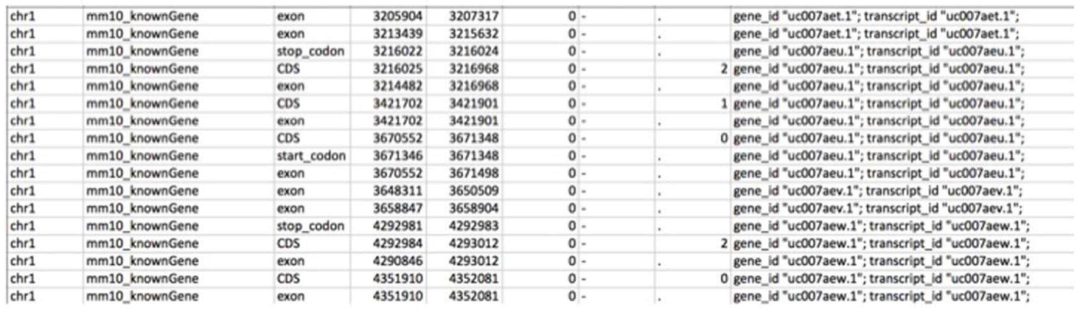
It follows that RSEM estimates the ratio between the amounts of different isoforms on the basis of the specific sequences that are isoform-specific (red, yellow), and it assigns the number of reads (*counts*).

Limitation → inhomogeneity in the level of coverage: it may be that the overall number of reads is not enough to detect a specific isoform. Higher coverage levels enable to appreciate all various isoforms. Second generation sequencing isn't suitable to have high coverage.

RSEM quantifies only **known transcripts**. Indeed, it needs a .gtf file in order to work (annotation file). Each annotation comes associated to a specific release of the assembly: you can't take an annotation and use it on a different assembly than the one it was associated to.

UCSC and ENSEMBL databases also differ in the way they organize the annotation files. In **UCSC** everything goes under the name of 'gene_id', even though this is the identifier of a specific isoform.

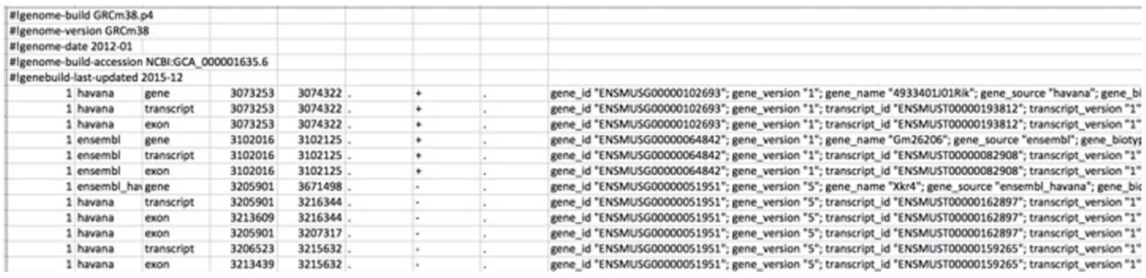
UCSC



chr1	mm10_knownGene	exon	3205904	3207317	0	-	.	gene_id "uc007aet.1"; transcript_id "uc007aet.1";
chr1	mm10_knownGene	exon	3213439	3215632	0	-	.	gene_id "uc007aet.1"; transcript_id "uc007aet.1";
chr1	mm10_knownGene	stop_codon	3216022	3216024	0	-	.	gene_id "uc007aeu.1"; transcript_id "uc007aeu.1";
chr1	mm10_knownGene	CDS	3216025	3216968	0	-	2	gene_id "uc007aeu.1"; transcript_id "uc007aeu.1";
chr1	mm10_knownGene	exon	3214482	3216968	0	-	.	gene_id "uc007aeu.1"; transcript_id "uc007aeu.1";
chr1	mm10_knownGene	CDS	3421702	3421901	0	-	1	gene_id "uc007aeu.1"; transcript_id "uc007aeu.1";
chr1	mm10_knownGene	exon	3421702	3421901	0	-	.	gene_id "uc007aeu.1"; transcript_id "uc007aeu.1";
chr1	mm10_knownGene	CDS	3670552	3671348	0	-	0	gene_id "uc007aeu.1"; transcript_id "uc007aeu.1";
chr1	mm10_knownGene	start_codon	3671346	3671348	0	-	.	gene_id "uc007aeu.1"; transcript_id "uc007aeu.1";
chr1	mm10_knownGene	exon	3670552	3671498	0	-	.	gene_id "uc007aeu.1"; transcript_id "uc007aeu.1";
chr1	mm10_knownGene	exon	3648311	3650509	0	-	.	gene_id "uc007aev.1"; transcript_id "uc007aev.1";
chr1	mm10_knownGene	exon	3658847	3658904	0	-	.	gene_id "uc007aev.1"; transcript_id "uc007aev.1";
chr1	mm10_knownGene	stop_codon	4292981	4292983	0	-	.	gene_id "uc007aew.1"; transcript_id "uc007aew.1";
chr1	mm10_knownGene	CDS	4292984	4293012	0	-	2	gene_id "uc007aew.1"; transcript_id "uc007aew.1";
chr1	mm10_knownGene	exon	4290846	4293012	0	-	.	gene_id "uc007aew.1"; transcript_id "uc007aew.1";
chr1	mm10_knownGene	CDS	4351910	4352081	0	-	0	gene_id "uc007aew.1"; transcript_id "uc007aew.1";
chr1	mm10_knownGene	exon	4351910	4352081	0	-	.	gene_id "uc007aew.1"; transcript_id "uc007aew.1";

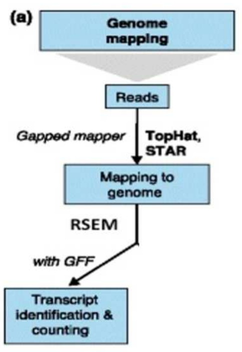
On **ENSEMBL** instead, all info are easily available and it's easy to find what you need. It also includes *biotypes* (which allow to identify specific subpopulations of RNA annotated all over the genome).

ENSEMBL



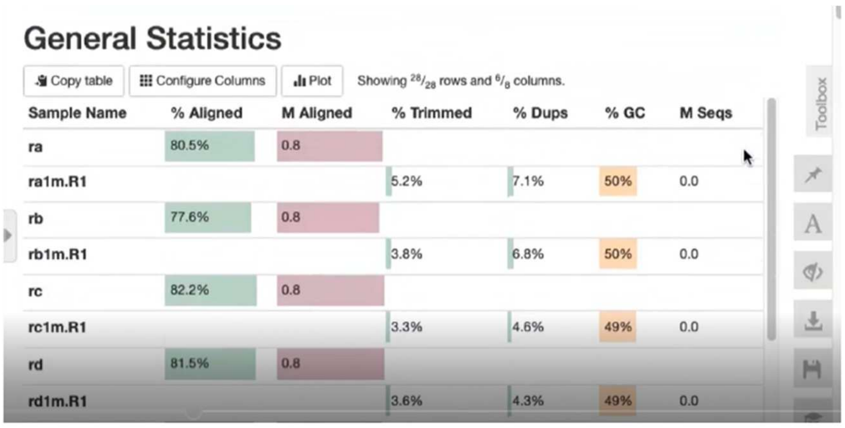
#!genome-build GRCm38.p4								
#!genome-version GRCm38								
#!genome-date 2012-01								
#!genome-build-accession NCBI:GCA_000001635.6								
#!genebuild-last-updated 2015-12								
1	havana	gene	3073253	3074322	.	+	.	gene_id "ENSMUSG00000102693"; gene_version "1"; gene_name "4933401J01Rik"; gene_source "havana"; gene_bi
1	havana	transcript	3073253	3074322	.	+	.	gene_id "ENSMUSG00000102693"; gene_version "1"; transcript_id "ENSMUST00000193812"; transcript_version "1"
1	havana	exon	3073253	3074322	.	+	.	gene_id "ENSMUSG00000102693"; gene_version "1"; transcript_id "ENSMUST00000193812"; transcript_version "1"
1	ensembl	gene	3102016	3102125	.	+	.	gene_id "ENSMUSG00000064842"; gene_version "1"; gene_name "Gm26206"; gene_source "ensembl"; gene_biotyp
1	ensembl	transcript	3102016	3102125	.	+	.	gene_id "ENSMUSG00000064842"; gene_version "1"; transcript_id "ENSMUST00000082908"; transcript_version "1"
1	ensembl	exon	3102016	3102125	.	+	.	gene_id "ENSMUSG00000064842"; gene_version "1"; transcript_id "ENSMUST00000082908"; transcript_version "1"
1	ensembl_hav	gene	3205901	3671498	.	-	.	gene_id "ENSMUSG00000051951"; gene_version "5"; gene_name "Xkr4"; gene_source "ensembl_havana"; gene_bio
1	havana	transcript	3205901	3216344	.	-	.	gene_id "ENSMUSG00000051951"; gene_version "5"; transcript_id "ENSMUST00000162897"; transcript_version "1"
1	havana	exon	3213609	3216344	.	-	.	gene_id "ENSMUSG00000051951"; gene_version "5"; transcript_id "ENSMUST00000162897"; transcript_version "1"
1	havana	exon	3205901	3207317	.	-	.	gene_id "ENSMUSG00000051951"; gene_version "5"; transcript_id "ENSMUST00000162897"; transcript_version "1"
1	havana	transcript	3206523	3215632	.	-	.	gene_id "ENSMUSG00000051951"; gene_version "5"; transcript_id "ENSMUST00000159265"; transcript_version "1"
1	havana	exon	3213439	3215632	.	-	.	gene_id "ENSMUSG00000051951"; gene_version "5"; transcript_id "ENSMUST00000159265"; transcript_version "1"

To summarize, everything we saw so far starts with reads in form of FASTQ files. These are the input for the FastQC analysis that will give info about the quality of the reads. Next to the FastQC analysis, **trimming** is performed to remove adapter sequences and to also know the average size of the insert. Then, alignment (STAR) and annotation/counting (RSEM + annotation files). **These steps are done sample-by-sample**. At this point, the MultiQC analysis can be run to evaluate trimming and alignment quality of all the sample together (you get an overview).



From a MultiQC analysis you obtain a **report**. This report includes:

- **General Statistics** → overview of the report results, where 'M Aligned' stands for million reads that have been aligned.

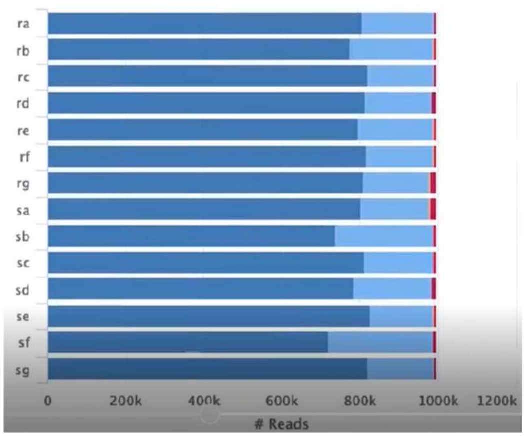


General Statistics

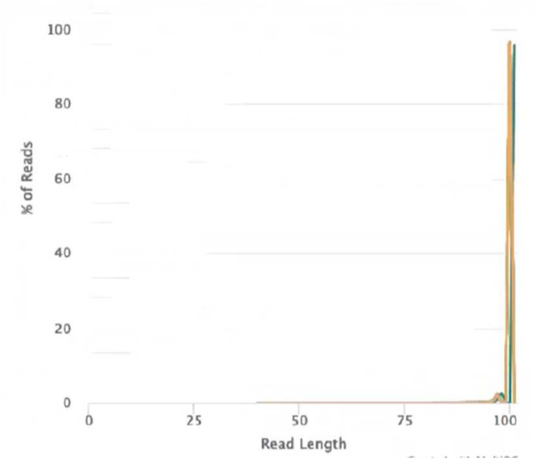
Showing $^{28}/_{28}$ rows and $^{6}/_{8}$ columns.

Sample Name	% Aligned	M Aligned	% Trimmed	% Dups	% GC	M Seqs
ra	80.5%	0.8				
ra1m.R1			5.2%	7.1%	50%	0.0
rb	77.6%	0.8				
rb1m.R1			3.8%	6.8%	50%	0.0
rc	82.2%	0.8				
rc1m.R1			3.3%	4.6%	49%	0.0
rd	81.5%	0.8				
rd1m.R1			3.6%	4.3%	49%	0.0

0.8 million reads aligned (1M circa). The sample is called 'ra1m' because it's a subsample of the initial dataset and just 1M reads were taken to do a quicker mapping. '.R1' refers to the name of the FASTQ file used for the mapping.

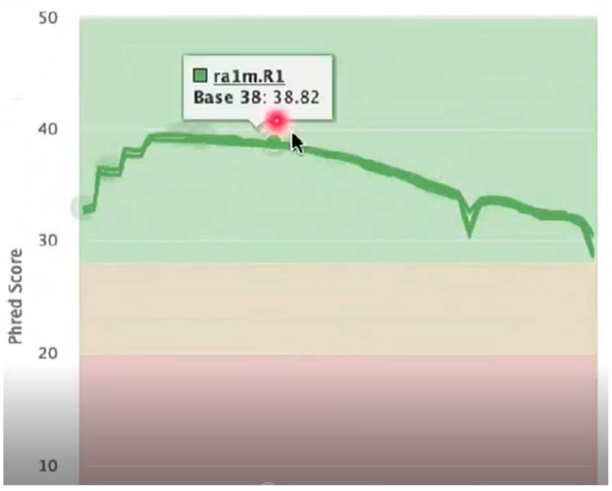


- **STAR: Alignment Scores** → info about uniquely mapped reads (blue), reads mapped to multiple loci (light blue), unmapped reads (red). All samples in this example behave similarly so the quality is quite good.



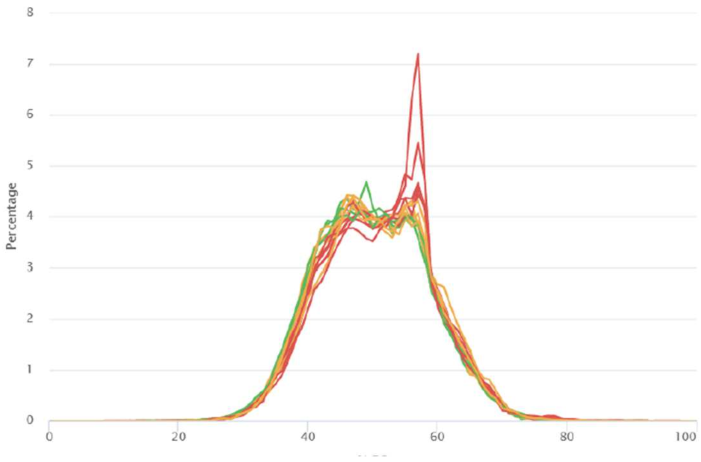
- **Skewer: read length distribution after trimming** → it tells the entity of trimming. In this case, only few nucleotides were trimmed.

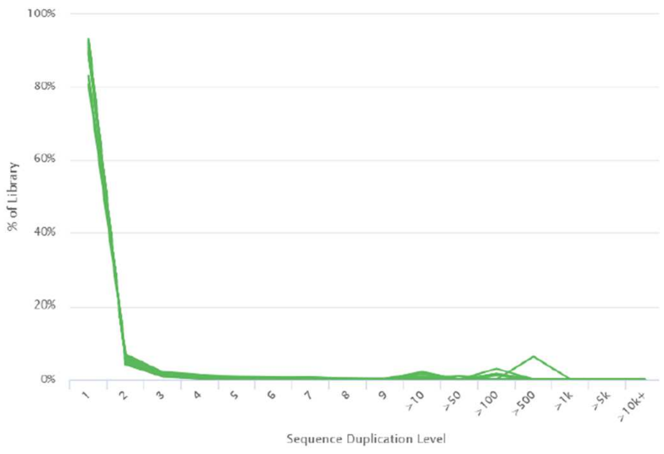
-



- **FastQC: Mean Quality Scores** → reads from all samples behave in the same way and stay in the green portion of the Phred Score, which is good.

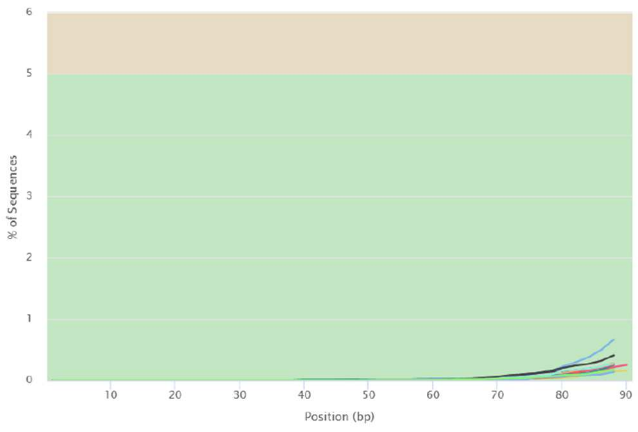
- **FastQC: Per Sequence GC Content** → if there is homogeneity between samples, it's ok no matter the shape of the curves.





- **FastQC: Sequence Duplication Levels** → chemical fragmentation was done, so a certain level of duplications is expected.

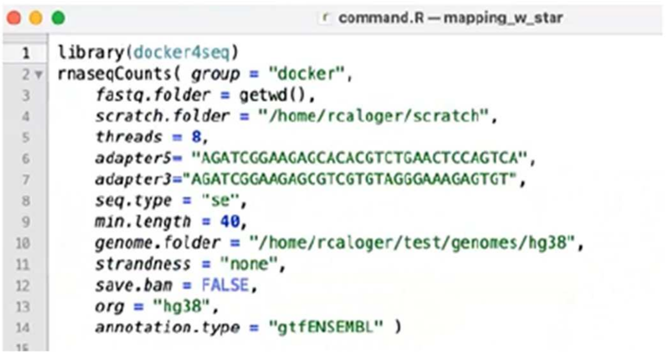
- **FastQC: Adapter Content** → adapters represent less than 1% of the sequences and thus less than 1% of the reads should be trimmed to remove adapters: very negligible amount.



Overall, MultiQC gives an overview of the quality of all of the samples at the same time.

The RNA-seq analysis (mapping and counting, etc) can be carried out using a specific package that wraps all these stuffs together, called **docker4seq**. The analysis is run on each sample using this tool which is handled by R. All the different functions (mapping, counting) are wrapped in Docker containers as they would require a lot of space (STAR and RSEM require a lot of RAM, very computing demanding). The command is the following:

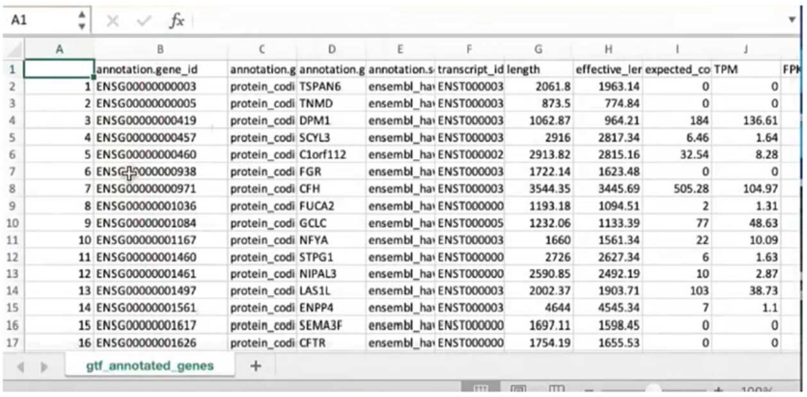
1. **`library(docker4seq)`** allows to load the docker4seq library.
2. **`rnaseqCounts`** is a function that wraps other functions inside. These are:
   a. a function retrieving a docker with FastQC to run it on each sample;
   b. another function downloads the docker with Skewer to perform the trimming
   c. a third function downloads STAR to perform alignment and counting
   d. then there are some scripts to put together all the above information in the wished final format.


command.R — mapping_w_star

```r
library(docker4seq)
rnaseqCounts( group = "docker",
     fastq.folder = getwd(),
     scratch.folder = "/home/rcaloger/scratch",
     threads = 8,
     adapter5= "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA",
     adapter3="AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT",
     seq.type = "se",
     min.length = 40,
     genome.folder = "/home/rcaloger/test/genomes/hg38",
     strandness = "none",
     save.bam = FALSE,
     org = "hg38",
     annotation.type = "gtfENSEMBL" )
```

The **`getwd()`** stands for the current working directory (same as **`pwd`** for Linux).The **`scratch.folder`** is needed because data are big and generally located on a low-performing disk (the hard disk), but doing the alignment on the hard disk would require a lot of time: therefore, the analysis is moved to the scratch folder (fast disk) and then the results are put back on the hard disk. Then, **`threads`** indicate the number of used computing units. The **`adapter5`** and **`adapter3`** are needed to perform the trimming and their sequences are available online (depending on the kit and library). Then you have to set the **`seq.type`**: either paired-end (pe) or single-end (se). Then there is **`min.length`** that allows to select the minimum length of the sequences after trimming: in this case, if after trimming the sequences shorter than 40 nucleotides, these are thrown away. The reference sequence that will be used for the alignment is stored in the indicated **`genome.folder`** (not in a FASTA format though). As regards **`strandness`**, these could be either "none", "forward", or "reverse" → "none" when we don't care about it (as with poly-A genes, which are coding ones and do not overlap); almost all kits are stranded. Knowing the strandness is important to assign reads to the two strands of the reference sequence, even when two reads come from the same position. Then, **`save.bam`** can be set to TRUE or FALSE (if we want the BAM file we set it TRUE, otherwise if we just want to count all sequences on specific positions of the genome, but we don't want to extract info related to variants, we set it FALSE). Then we specify the assembly and annotation we want to work with: **`org`** and **`annotation.type`** respectively.

From this analysis we obtain an <u>output for the FastQC analysis</u>, an <u>output for the STAR genome mapping</u>, and <u>an output for the trimming procedure</u> and <u>one for</u>


A1

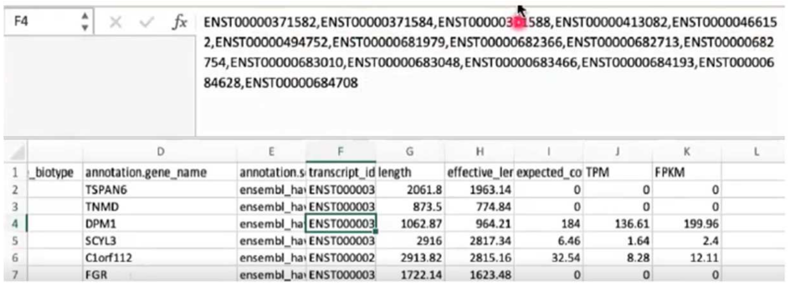
	A	B	C	D	E	F	G	H	I	J	
1		annotation.gene_id	annotation.g	annotation.g	annotation.s	transcript_id	length	effective_ler	expected_co	TPM	FPK
2	1	ENSG00000000003	protein_codi	TSPAN6	ensembl_ha	ENST000003	2061.8	1963.14	0	0	
3	2	ENSG00000000005	protein_codi	TNMD	ensembl_ha	ENST000003	873.5	774.84	0	0	
4	3	ENSG00000000419	protein_codi	DPM1	ensembl_ha	ENST000003	1062.87	964.21	184	136.61	
5	4	ENSG00000000457	protein_codi	SCYL3	ensembl_ha	ENST000003	2916	2817.34	6.46	1.64	
6	5	ENSG00000000460	protein_codi	C1orf112	ensembl_ha	ENST000002	2913.82	2815.16	32.54	8.28	
7	6	ENSG00000000938	protein_codi	FGR	ensembl_ha	ENST000003	1722.14	1623.48	0	0	
8	7	ENSG00000000971	protein_codi	CFH	ensembl_ha	ENST000003	3544.35	3445.69	505.28	104.97	
9	8	ENSG00000001036	protein_codi	FUCA2	ensembl_ha	ENST000000	1193.18	1094.51	2	1.31	
10	9	ENSG00000001084	protein_codi	GCLC	ensembl_ha	ENST000005	1232.06	1133.39	77	48.63	
11	10	ENSG00000001167	protein_codi	NFYA	ensembl_ha	ENST000003	1660	1561.34	22	10.09	
12	11	ENSG00000001460	protein_codi	STPG1	ensembl_ha	ENST000000	2726	2627.34	6	1.63	
13	12	ENSG00000001461	protein_codi	NIPAL3	ensembl_ha	ENST000000	2590.85	2492.19	10	2.87	
14	13	ENSG00000001497	protein_codi	LAS1L	ensembl_ha	ENST000003	2002.37	1903.71	103	38.73	
15	14	ENSG00000001561	protein_codi	ENPP4	ensembl_ha	ENST000003	4644	4545.34	7	1.1	
16	15	ENSG00000001617	protein_codi	SEMA3F	ensembl_ha	ENST000000	1697.11	1598.45	0	0	
17	16	ENSG00000001626	protein_codi	CFTR	ensembl_ha	ENST000000	1754.19	1655.53	0	0	

gtf_annotated_genes

The second column is the **gene identifier** in ENSEMBL, and tells which is the code associated to that specific gene on ENSEMBL. Then there is the **biotype** (protein_coding in this case). Then there is column with gene symbols (**annotation_gene_name**). **TPM** and **FPKM** are sort of normalization that will be discussed later on in the course. The **expected_counts** is the output of RSEM (which gets for each isoform the number of counts, sums them together and gives the expected counts for the gene given all the annotated transcripts) → looking at **transcript_id**, in this column all transcripts associated to that specific genomic locus (indicated by the gene name column) are annotated: by putting together all those transcripts we get the counts associated to that specfiic gene:

<u>the annotation</u>. Everything is stored in an Excel file (*gtf_annotated_genes.results*):

The *gtf_annotated_genes.results* file contains counts of genes and we'll work on these.



F4 fx ENST00000371582,ENST00000371584,ENST00000371588,ENST00000413082,ENST00000466152,ENST00000494752,ENST00000681979,ENST00000682366,ENST00000682713,ENST00000682754,ENST00000683010,ENST00000683048,ENST00000683466,ENST00000684193,ENST0000068 4628,ENST00000684708

		D	E	F	G	H	I	J	K	L
1	_biotype	annotation.gene_name	annotation.s	transcript_id	length	effective_ler	expected_co	TPM	FPKM	
2		TSPAN6	ensembl_ha	ENST000003	2061.8	1963.14	0	0	0	
3		TNMD	ensembl_ha	ENST000003	873.5	774.84	0	0	0	
4		DPM1	ensembl_ha	ENST000003	1062.87	964.21	184	136.61	199.96	
5		SCYL3	ensembl_ha	ENST000003	2916	2817.34	6.46	1.64	2.4	
6		C1orf112	ensembl_ha	ENST000002	2913.82	2815.16	32.54	8.28	12.11	
7		FGR	ensembl_ha	ENST000003	1722.14	1623.48	0	0	0	

<u>DOCKER CONTAINERS</u>

You can build a docker container from scratch, but you can also download them from online repositories (Docker Hub). With docker you can run the analysis you want just by downloading and then running a specific image/container (which contains the software/scripts you need for the analysis).

To download a docker image, type the following command: **`docker pull <name_of_the_container>`**:

`docker pull ubuntu:21.04` → to download a specific version of ubuntu inside a docker image/container.

The idea is to build a container with Ubuntu (same as we using a virtual machine to create a Linux environment in our Windows system), because the things we want to do require a Linux-based system. At this point we can run this container we just downloaded:

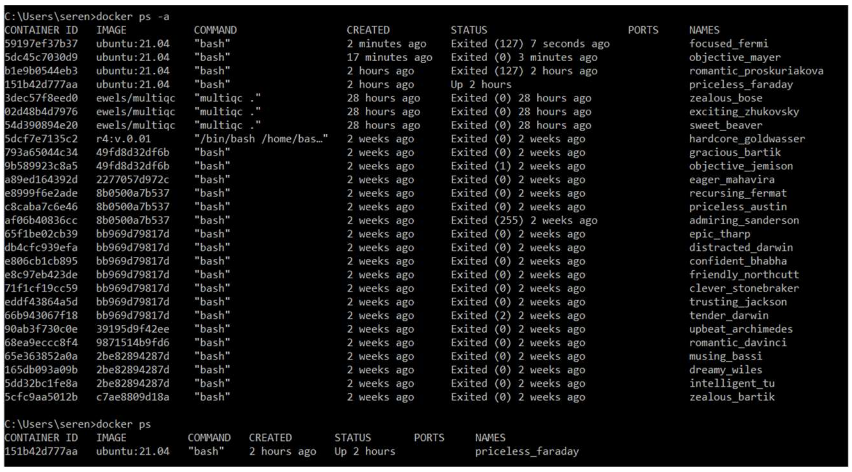
`docker run -i -t ubuntu:21.04` → **`-i`** to run it interactively, **`-t`** to have a terminal allowing to interact with the docker container. **If you directly type this command without pulling it before, it works but it tells you that no such image was found locally (on your PC), and thus it pulls it from online and then immediately runs it** (both things done together).

After this command is launched, you will find yourself inside the docker container (symbol **#** and **root@** → **you are in a Linux environment**, and you can type **`ls`** to see what's inside the container). Move through the directories using **`cd`** (e.g., **`cd /home`**).

Once you run a docker container, this docker is not an image anymore but it's something running on the computer (on the RAM). So when the docker is not run, it is an image, but then it becomes a container running on your RAM (i.e. you're using it).

`docker ps` → to know which docker containers are running at the present time.

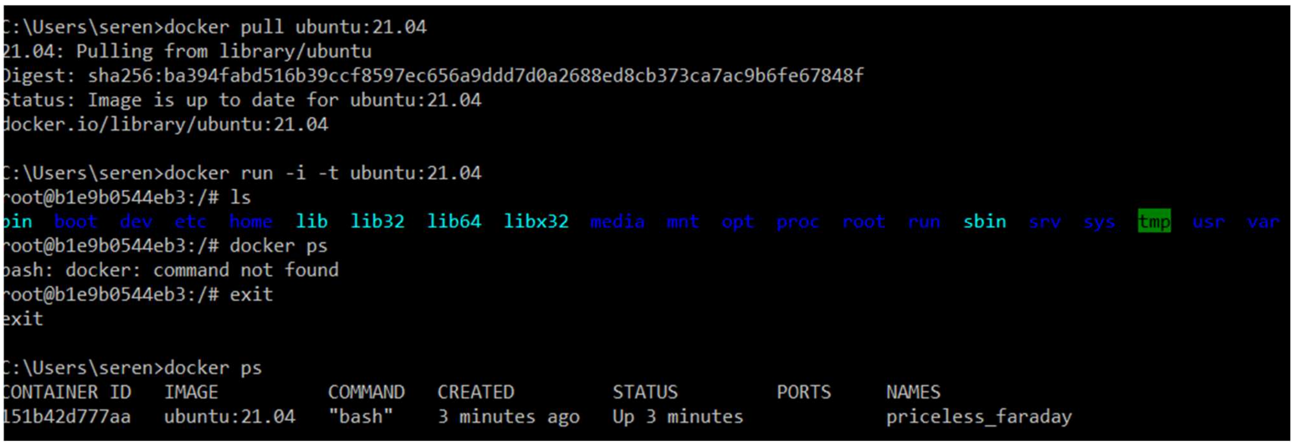
`docker ps -a` → it shows all the docker containers that were previously closed.



```
C:\Users\seren>docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                      PORTS     NAMES
59197ef37b37   ubuntu:21.04   "bash"                   2 minutes ago    Exited (127) 7 seconds ago            focused_fermi
5dc45c7030d9   ubuntu:21.04   "bash"                   17 minutes ago   Exited (0) 3 minutes ago              objective_mayer
b1e9b0544eb3   ubuntu:21.04   "bash"                   2 hours ago      Exited (127) 2 hours ago              romantic_proskuriakova
151b42d777aa   ubuntu:21.04   "bash"                   2 hours ago      Up 2 hours                            priceless_faraday
3dec57f8eed0   ewels/multiqc  "multiqc ."              28 hours ago     Exited (0) 28 hours ago               zealous_bose
02d48b4d7976   ewels/multiqc  "multiqc ."              28 hours ago     Exited (0) 28 hours ago               exciting_zhukovsky
54d390894e20   ewels/multiqc  "multiqc ."              28 hours ago     Exited (0) 28 hours ago               sweet_beaver
5dcf7e7135c2   r4:v.0.01      "/bin/bash /home/bas…"   2 weeks ago      Exited (0) 2 weeks ago                hardcore_goldwasser
793a65044c34   49fd8d32df6b   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                gracious_bartik
9b589923c8a5   49fd8d32df6b   "bash"                   2 weeks ago      Exited (1) 2 weeks ago                objective_jemison
a89ed164392d   2277057d972c   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                eager_mahavira
e8999f6e2ade   8b0500a7b537   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                recursing_fermat
c8caba7c6e46   8b0500a7b537   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                priceless_austin
af06b40836cc   8b0500a7b537   "bash"                   2 weeks ago      Exited (255) 2 weeks ago              admiring_sanderson
65f1be02cb39   bb969d79817d   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                epic_tharp
db4cfc939efa   bb969d79817d   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                distracted_darwin
e806cb1cb895   bb969d79817d   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                confident_bhabha
e8c97eb423de   bb969d79817d   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                friendly_northcutt
71f1cf19cc59   bb969d79817d   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                clever_stonebraker
eddf43864a5d   bb969d79817d   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                trusting_jackson
66b943067f18   bb969d79817d   "bash"                   2 weeks ago      Exited (2) 2 weeks ago                tender_darwin
90ab3f730c0e   39195d9f42ee   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                upbeat_archimedes
68ea9eccc8f4   9871514b9fd6   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                romantic_davinci
65e363852a0a   2be82894287d   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                musing_bassi
165db093a09b   2be82894287d   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                dreamy_wiles
5dd32bc1fe8a   2be82894287d   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                intelligent_tu
5cfc9aa5012b   c7ae8809d18a   "bash"                   2 weeks ago      Exited (0) 2 weeks ago                zealous_bartik

C:\Users\seren>docker ps
CONTAINER ID   IMAGE          COMMAND   CREATED       STATUS       PORTS     NAMES
151b42d777aa   ubuntu:21.04   "bash"    2 hours ago   Up 2 hours             priceless_faraday
```

`exit` → to exit the container. Now if you type `docker ps` you won't see any image running anymore (??).



```
:\Users\seren>docker pull ubuntu:21.04
21.04: Pulling from library/ubuntu
igest: sha256:ba394fabd516b39ccf8597ec656a9ddd7d0a2688ed8cb373ca7ac9b6fe67848f
tatus: Image is up to date for ubuntu:21.04
ocker.io/library/ubuntu:21.04

:\Users\seren>docker run -i -t ubuntu:21.04
oot@b1e9b0544eb3:/# ls
in  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
oot@b1e9b0544eb3:/# docker ps
ash: docker: command not found
oot@b1e9b0544eb3:/# exit
xit

:\Users\seren>docker ps
ONTAINER ID   IMAGE          COMMAND   CREATED         STATUS         PORTS     NAMES
51b42d777aa   ubuntu:21.04   "bash"    3 minutes ago   Up 3 minutes             priceless_faraday
```

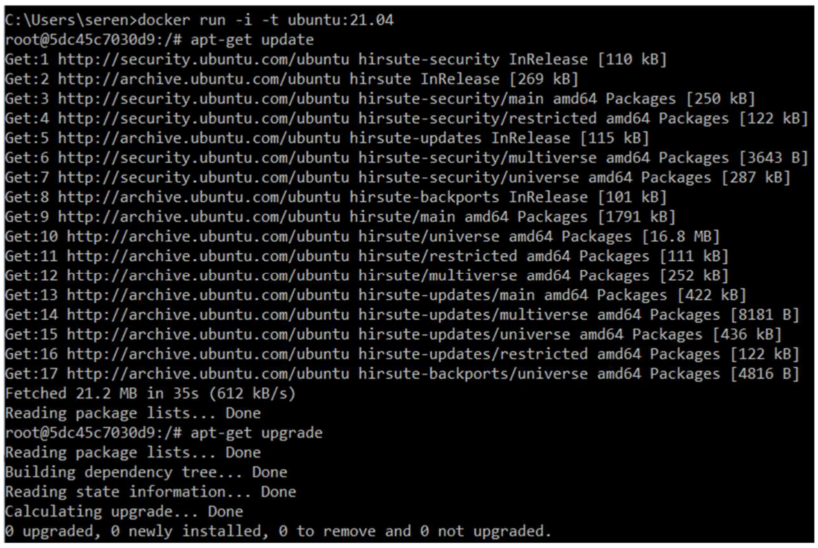
What else can you do while you're inside the docker container?

`apt-get update` → to update the Ubuntu version you just downloaded.

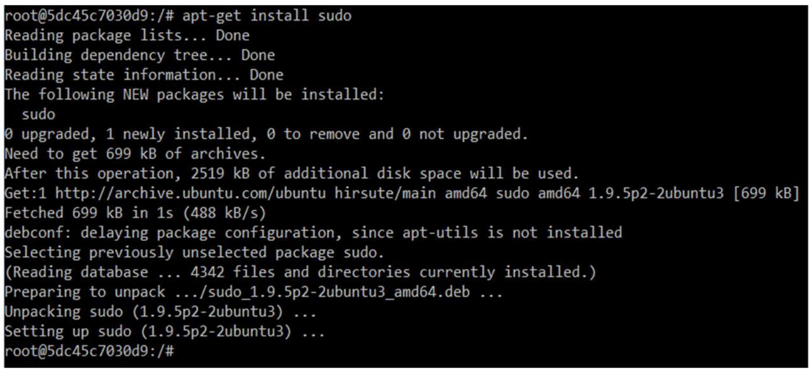
`apt-get upgrade` → no upgrade if your version was already updated.

Both commands have to be run.

The code after `root@` is the **instance** and it is associated to the docker container. Every time you exit the docker, and then you run again the same container, the code changes.



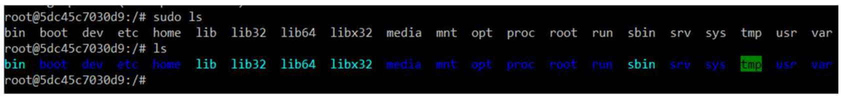
```
C:\Users\seren>docker run -i -t ubuntu:21.04
root@5dc45c7030d9:/# apt-get update
Get:1 http://security.ubuntu.com/ubuntu hirsute-security InRelease [110 kB]
Get:2 http://archive.ubuntu.com/ubuntu hirsute InRelease [269 kB]
Get:3 http://security.ubuntu.com/ubuntu hirsute-security/main amd64 Packages [250 kB]
Get:4 http://security.ubuntu.com/ubuntu hirsute-security/restricted amd64 Packages [122 kB]
Get:5 http://archive.ubuntu.com/ubuntu hirsute-updates InRelease [115 kB]
Get:6 http://security.ubuntu.com/ubuntu hirsute-security/multiverse amd64 Packages [3643 B]
Get:7 http://security.ubuntu.com/ubuntu hirsute-security/universe amd64 Packages [287 kB]
Get:8 http://archive.ubuntu.com/ubuntu hirsute-backports InRelease [101 kB]
Get:9 http://archive.ubuntu.com/ubuntu hirsute/main amd64 Packages [1791 kB]
Get:10 http://archive.ubuntu.com/ubuntu hirsute/universe amd64 Packages [16.8 MB]
Get:11 http://archive.ubuntu.com/ubuntu hirsute/restricted amd64 Packages [111 kB]
Get:12 http://archive.ubuntu.com/ubuntu hirsute/multiverse amd64 Packages [252 kB]
Get:13 http://archive.ubuntu.com/ubuntu hirsute-updates/main amd64 Packages [422 kB]
Get:14 http://archive.ubuntu.com/ubuntu hirsute-updates/multiverse amd64 Packages [8181 B]
Get:15 http://archive.ubuntu.com/ubuntu hirsute-updates/universe amd64 Packages [436 kB]
Get:16 http://archive.ubuntu.com/ubuntu hirsute-updates/restricted amd64 Packages [122 kB]
Get:17 http://archive.ubuntu.com/ubuntu hirsute-backports/universe amd64 Packages [4816 B]
Fetched 21.2 MB in 35s (612 kB/s)
Reading package lists... Done
root@5dc45c7030d9:/# apt-get upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```



```
root@5dc45c7030d9:/# apt-get install sudo
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  sudo
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 699 kB of archives.
After this operation, 2519 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu hirsute/main amd64 sudo amd64 1.9.5p2-2ubuntu3 [699 kB]
Fetched 699 kB in 1s (488 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package sudo.
(Reading database ... 4342 files and directories currently installed.)
Preparing to unpack .../sudo_1.9.5p2-2ubuntu3_amd64.deb ...
Unpacking sudo (1.9.5p2-2ubuntu3) ...
Setting up sudo (1.9.5p2-2ubuntu3) ...
root@5dc45c7030d9:/#
```

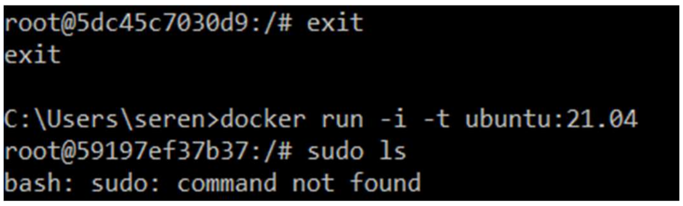
`apt-get install sudo` → `sudo` means the possibility to run a command as a superuser. As such, you have the highest priority but you could also delete everything. Luckily if you delete something and you don't commit the changes, you just need to exit the docker container and run it again to recover everything. Thus, you have to type this command once you've run your container. **You have to perform <u>every time</u> `apt-get update` and `apt-get upgrade` before typing this command otherwise it won't work** (I guess because this command works only with latest Ubuntu version and not this one).

`sudo ls` → basically this is the classical ls command but run with sudo. If sudo has been successfully installed, you should see this output:



```
root@5dc45c7030d9:/# sudo ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@5dc45c7030d9:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@5dc45c7030d9:/#
```

If you don't commit this action, and you exit the docker, once you go back inside sudo won't be present if you type **`sudo ls`**. A new instance has been associated to the container, and in this instance doesn't have anything of what you did before.

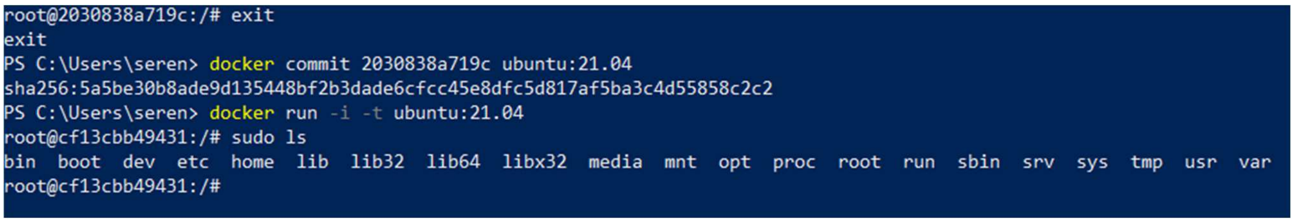


```
root@5dc45c7030d9:/# exit
exit

C:\Users\seren>docker run -i -t ubuntu:21.04
root@59197ef37b37:/# sudo ls
bash: sudo: command not found
```

The command **`docker commit <instance_code> <name_of_the_container>`** allows you to save the changes you made:

`docker commit 59197ef37b37 ubuntu:21.04` → this should be done after you've exit the docker to have your changes saved (when you run again the container you will be able to type **`sudo ls`**).

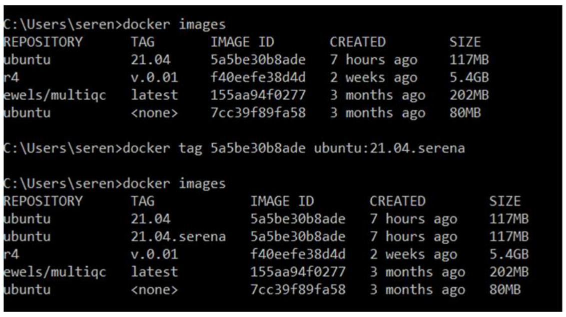


```
root@2030838a719c:/# exit
exit
PS C:\Users\seren> docker commit 2030838a719c ubuntu:21.04
sha256:5a5be30b8ade9d135448bf2b3dade6cfcc45e8dfc5d817af5ba3c4d55858c2c2
PS C:\Users\seren> docker run -i -t ubuntu:21.04
root@cf13cbb49431:/# sudo ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@cf13cbb49431:/#
```

You also have to install **`nano`** with the following command: **`sudo apt-get install nano`** (see later why: basically, **`nano`** is a editor without graphical interface to use from the terminal).

The docker instance is also called image ID as it is associated to a docker image. The problem when you run again the same container is that the image ID next time will be different from before → the **`docker tag <image_ID> <tag>`** command enables to save the modified version of the container with the tag you prefer, so that you can redownload the same version again:

`docker tag 5a5be30b8ade ubuntu:21.04.serena`



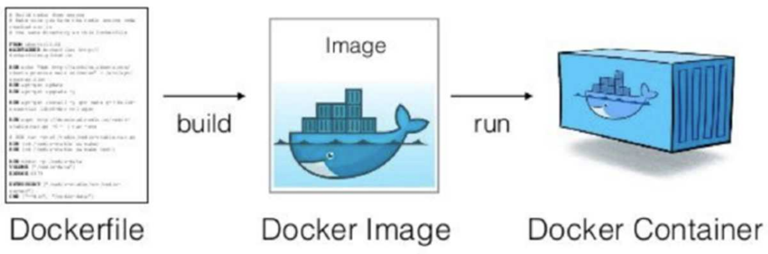
```
C:\Users\seren>docker images
REPOSITORY      TAG       IMAGE ID       CREATED         SIZE
ubuntu          21.04     5a5be30b8ade   7 hours ago     117MB
r4              v.0.01    f40eefe38d4d   2 weeks ago     5.4GB
ewels/multiqc   latest    155aa94f0277   3 months ago    202MB
ubuntu          <none>    7cc39f89fa58   3 months ago    80MB

C:\Users\seren>docker tag 5a5be30b8ade ubuntu:21.04.serena

C:\Users\seren>docker images
REPOSITORY      TAG            IMAGE ID       CREATED         SIZE
ubuntu          21.04          5a5be30b8ade   7 hours ago     117MB
ubuntu          21.04.serena   5a5be30b8ade   7 hours ago     117MB
r4              v.0.01         f40eefe38d4d   2 weeks ago     5.4GB
ewels/multiqc   latest         155aa94f0277   3 months ago    202MB
ubuntu          <none>         7cc39f89fa58   3 months ago    80MB
```

Having your own image allows you to have a version where you do all the modifications you want.

There is still a problem: every time we close the terminal, we lose all the information and stuff we did with the previous steps. How to keep memory of them? You create a file where all these steps are stored, such a script let's say: you generate a **Dockerfile**. In a Dockerfile all the info to build a Docker image from scratch (and to run a container) are stored.



How to build a Dockerfile?

1. Create a new folder on your PC (e.g., your Desktop, so that you'll have to call a short path);
2. Inside this folder, generate a text file which will be your Dockerfile and write these strings of info:

   ```
   FROM ubuntu:20.04
   RUN apt-get update
   RUN apt-get upgrade -y
   RUN apt-get install -y sudo
   RUN sudo apt-get install -y nano
   COPY ./command.sh /home/  #the file command.sh is copied to the folder 'home'.
   RUN chmod +x /home/command.sh  #this makes the file executable (chmod in general is used to modify the properties of the file).
   ```

3. Launch the command to create the docker image from scratch (move to the folder where the Dockerfile is first if you don't want to type the whole path to the folder as done below):

```
docker build <path_of_the_dockerfile_folder> -t <name_of_docker_image_you_want>
```

-y in the script stand for the fact that installation processes won't ask for permission and the command will be entirely executed after you've launched the above command. Everything after the **RUN** is done inside the image (in this case, **ubuntu:21.04**). **-t** is needed to tag (give a name) to the docker you're creating.

REMOVE THE .txt EXTENSION TO THE DOCKERFILE IN ORDER TO AVOID ERRORS.