INF-BIOX120: Analysis of RNA-Seq Data

Jan Christian Bryne[1,2,3], Susanne Lorenz[1], and Chilamakuri C.S. Reddy[2]

[1]Genomics Core Facility, Oslo University Hospital and Oslo University.
www.genomics.no/oslo
[2]Department of Tumor Biology, ICR, Oslo University Hospital
[3]Northern Genomics AS

18-19 September, 2013

Contents

Schedule

Time	Topic	Responsible
Wed 18th 09:00 - 10:00	General Introduction	Susanne
Wed 18th 10:00 - 10:45	Introduction Gene Expression Analysis	Jan Christian
Wed 18th 10:45 - 12:00	Practical: Genome Alignment	Jan Christian
Wed 18th 12:00 - 12:45	Lunch	
Wed 18th 12:45 - 14:00	Practical: Expression Quantification	Jan Christian
Wed 18th 14:00 - 16:00	Practical: Differential Expression Analysis	Jan Christian
Wed 18th 16:00 - 17:00	Questions and Answers	Jan Christian
Thu 19th 09:00 - 10:00	Introduction Transcriptome Assembly	Jan Christian
Thu 19th 10:00 - 12:00	Practical: Transcriptome Assembly	Jan Christian
Thu 19th 12:00 - 12:45	Lunch	
Thu 19th 12:45 - 14:45	Practical: Transcriptome Assembly	Jan Christian
Thu 19th 14:45 - 15:00	Summary	Jan Christian
Thu 19th 15:00 - 16:00	Other Applications of RNA-seq	Susanne
Thu 19th 15:45 - 17:00	Questions and Answers	Susanne and Jan Christian

1 Introduction

RNA-seq is a method to investigate the RNA content of a sample using high-throughput sequencing. This has proven to be a very successful method and has led to many discoveries related to RNA and transcription. RNA-seq can be performed using different sequencing technologies, and applied in different way to answer diverse biological questions. In this course we focus on RNA-seq from an Illumina HiSeq sequencer. To assist those that would like to perform similar analysis on their own, we have listed the tools used in this tutorial in chapter 7.

1.1 Data set

We are providing a data set that we will be using for all the analyses. The data are from an osteosarcoma cancer cell line (called U2OS), and from healthy tissue (called WT). There are in total four RNA-seq data sets, two biological replicates from each condition. The samples were sequenced with a paired end strategy, with 100 bp long reads. See the table below for a description of all data. We used the TruSeq Stranded Total RNA with Ribo-Zero Gold protocol from Illumina, which uses ribosomal RNA depletion and produces strand specific reads. As these four data sets are rather large and would take some time to process in full, we will only use a small subset of the data in this course. We have retained only those reads that aligning to chromosome 19, and we will focus only on this chromosome for all analyses.

Condition	Replicate	Pair	Filename
Healty	one	first	WT.rep1.R1.fastq
Healty	one	second	WT.rep1.R2.fastq
Healty	two	first	WT.rep2.R1.fastq
Healty	two	second	WT.rep2.R2.fastq
Tumor	one	first	U2OS.rep1.R1.fastq
Tumor	one	second	U2OS.rep1.R2.fastq
Tumor	two	first	U2OS.rep2.R1.fastq
Tumor	two	second	U2OS.rep2.R2.fastq

We have also downloaded some reference data sets that we will be using. The DNA sequence of chromosome 19 from the human reference genome was downloaded as a fasta file from the University of California Santa Cruz (UCSC) table browser, and the human genome annotation was downloaded as a GTF file from Ensembl.

1.2 Directory Structure

We are going to set up a new directory called RNASeq that will act as a separate workspace for this analysis. Within this directory we will create several new directories. To set up the workspace, please run the following commands:

```
mkdir RNASeq
cd RNASeq
cp -r /doc/RNASeq doc
mkdir data
cp -r /data/RNASeq/fastq data/
cp -r /data/RNASeq/reference data/
```

```
mkdir tracks
cp -r /data/RNASeq/RNASeq_Course_TrackHub/ tracks/
```

1.3 Commands file

Throughout this document we will describe many commands to use in the terminal window. To make it easier for you, we have collected all the commands you will use in a separate file called 'commands.txt', which is located in the doc directory together with this document. You may use this file to see how to type the commands, or to copy and paste the commands into the terminal window.

1.4 Tracking file

There are several types of information that needs to be recorded during the analysis, and we recommend that you create a text file that you use during the analysis to store this information. It does not matter where you place this file as long as you are able to find it again when you need it.

2 Initial Quality Control

We can now have an initial look at the RNA-seq data by navigating to the relevant directory and look at the files there. To start with we can use the following commands:

```
cd data/fastq
ls -lh
```

You should now see the following list of the fastq files containing the RNA-seq data:

```
-rw-rw-r-- 1 username group  98M Sep 15 15:14 U2OS.rep1.R1.fastq
-rw-rw-r-- 1 username group  98M Sep 15 15:14 U2OS.rep1.R2.fastq
-rw-rw-r-- 1 username group  96M Sep 15 15:14 U2OS.rep2.R1.fastq
-rw-rw-r-- 1 username group  96M Sep 15 15:14 U2OS.rep2.R2.fastq
-rw-rw-r-- 1 username group 107M Sep 15 15:14 WT.rep1.R1.fastq
-rw-rw-r-- 1 username group 107M Sep 15 15:14 WT.rep1.R2.fastq
-rw-rw-r-- 1 username group 104M Sep 15 15:14 WT.rep2.R1.fastq
-rw-rw-r-- 1 username group 104M Sep 15 15:14 WT.rep2.R2.fastq
```

As previously mentioned we are looking at paired end data from two different conditions and with two replicates. See sample table in the introduction for details. Note that each experiment have two sets of reads which represents the first and second read of the pairs (R1 and R2). Lets have a look at the beginning of one file:

```
head U2OS.rep1.R1.fastq
```

You can now see the first reads of this file, where each read is provided with an ID and a quality score. This was the 'first' or 'left' reads of the U2OS replicate 1 experiment. Let's have a look at the 'second' or 'right' reads from the same experiment:

```
head U2OS.rep1.R2.fastq
```

As you can see these reads look a bit different, but they share the same ID as the ones we saw for the first reads. This is how the pairing information is retained and can be used by an aligner. The protocol we used for these data provides strand information, but it's the cDNA that has been sequenced, so the strand actually being expressed is the opposite of what has been sequenced. An R1 file contains reads sequenced from the forward strand of the start of the cDNA, and the R2 file

contains reads sequenced from the reverse strand of the end of the cDNA. By keeping track of which read is R1 and which is R2, we are able to identify the strand that was transcribed.

The first part of the read ID refers to the name of the sequencing machine, and is therefore the same for all experiments. We can use this first part of the ID to count the number of reads in each data set with the grep command:

```
grep -c ^@HWI-ST1337 *
```

The grep command counts the number of times "@HWI-ST1337" occurs at the beginning of a line. Note that these characters could theoretically also occur at the beginning of a quality string in the fastq file, but the probability of that is extremely small so for all practical purposes we can safely assume we are getting accurate counts.

You should now have the total read count for each experiment. Please record these numbers as we will use them later on in the analysis. Note that you have the identical number of reads in the 'R1' and 'R2' files for each experiment. This is of course not surprising since they contain the first and second reads from each read pair. For simplicity we only count the number of pairs for each experiment. The number of reads should not be too different between experiments, since we will compare the experiments with each other later on. A higher read count gives an increase of power in detection of differentially expressed genes, but it requires a comparable number of reads in each experiment. We will apply statistical methods that can deal with some differences, but a read count difference of one or several orders of magnitude will likely be too large.

As always with sequencing data, we apply a tool for checking for artifacts at the beginning of the analysis. We can now create a directory to put the quality control data in, and run the FastQC program on each file with the following commands.

```
mkdir QC
fastqc --outdir QC *fastq
```

A bias towards high GC count has been reported several times for RNA-seq data. This may be caused by both a general high GC content in coding regions and a preference for GC sequences during fragmentation. Do you see any bias in these data? We finish this part of the analysis by navigating back to the RNASeq/data directory:

```
cd ..
pwd
```

This should give a path similar to the following:

```
/home/username/RNASeq/data
```

3 Genome Alignment

Quantification of expression from RNA-seq data can quickly and accurately be performed by alignment to the reference genome, when the experiment is performed on an organism with a well annotated genome like human or mouse. A fundamental problem with aligning to the genome is the difference between the genome and the transcriptome: In RNA-seq we are sequencing RNA molecules which often differs from the corresponding transcribed region in the genome due to splicing or other factors such as fusion transcripts. See figure 1.

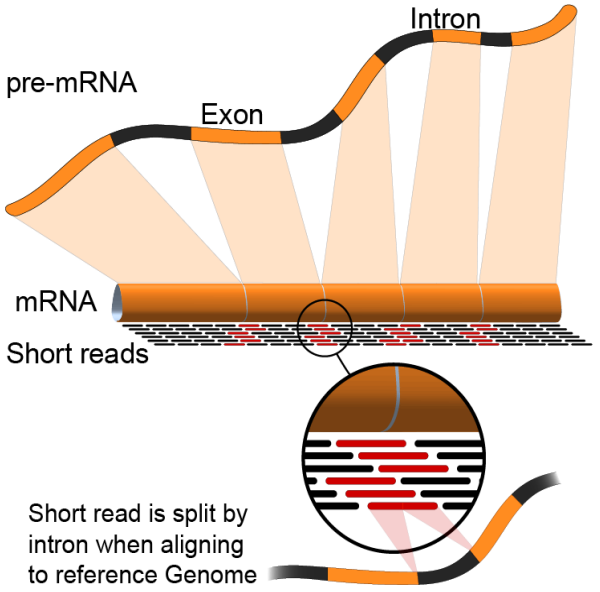

Figure 1: From en.wikipedia.org/wiki/File:RNA-Seq-alignment.png

3.1 Running the Tophat2 Aligner

We are going to use the Tophat2 aligner which is able to align reads across exon junctions and between fusion transcripts. It works by first aligning reads to known transcript sequences, as defined by existing genome annotation. Those reads that does not align to any known transcript sequence are then fragmented into segments and aligned to the genome. These segment alignments can be used to identify novel splice junctions that complement the existing genome annotation. This may enable alignment of those reads that did not align to the known transcript sequences. See figure 2 for an illustration.

Before we can run Tophat2, we will need to generate an alignment index. This index is a binary representation of the reference genome which makes the process of aligning much faster. In our case we are only using human chromosome 19 as reference. The genomic sequences of chromosome 19 is located in data/reference/genome/chr19.fa. The Tophat2 aligner actually use the Bowtie aligner under the hood, so we are going to use a Bowtie tool to generate the alignment index. The name of the tool is bowtie2-build, and it requires two parameters: The reference sequence in fasta format, and a prefix for the output which includes the directory to place the index files in and the first part of filenames. Tophat2 will execute faster if also has access to the fasta file, so we are placing the index in the same directory, and with the same prefix in order for Tophat2 to find it later. To generate the index, type:

```
bowtie2-build reference/genome/chr19.fa reference/genome/chr19
```

This takes a few minutes to run and will generate several files. It also produce a lot of output on the terminal that is not strictly necessary. When the command is finished you can inspect the results by typing:

```
ls -lh reference/genome
```

This will produce output similar to the following:

```
-rw-rw-r-- 1 username group 22M Sep 15 15:19 chr19.1.bt2
```

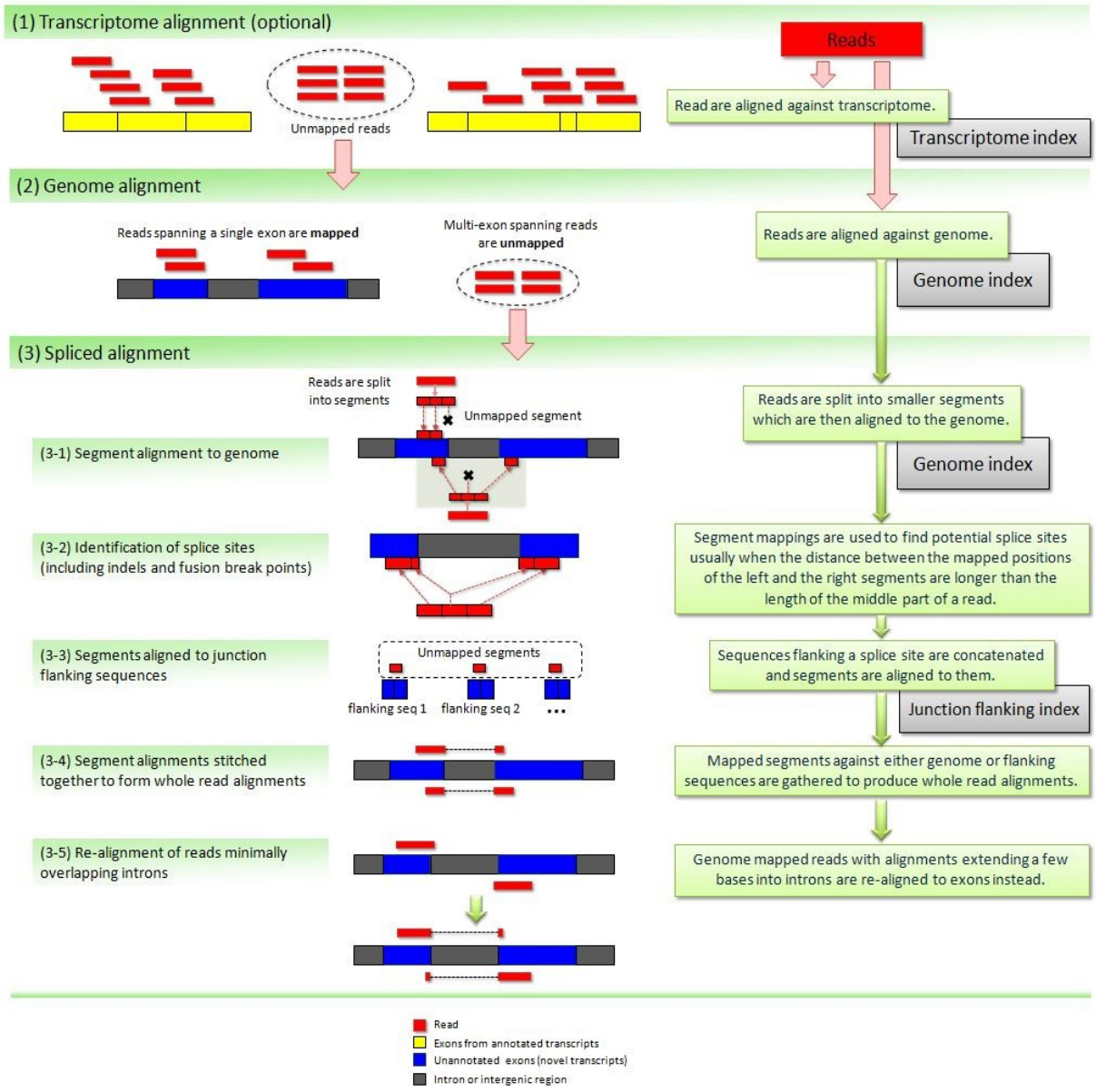

Figure 2: TopHat2 pipeline. Kim *et al.*, 2013

```
-rw-rw-r-- 1 username group 14M Sep 15 15:19 chr19.2.bt2
-rw-rw-r-- 1 username group  62 Sep 15 15:18 chr19.3.bt2
-rw-rw-r-- 1 username group 14M Sep 15 15:18 chr19.4.bt2
-rw-rw-r-- 1 username group 58M Sep 15 15:14 chr19.fa
-rw-rw-r-- 1 username group 22M Sep 15 15:20 chr19.rev.1.bt2
-rw-rw-r-- 1 username group 14M Sep 15 15:20 chr19.rev.2.bt2
```

We are now ready to align our RNA-seq data to the chromosome 19 reference index. As previously described, the Tophat2 aligner will initially align to known transcript sequences, but in order to do so

we need to supply this to Tophat2 with the GTF parameter. Secondly we need to provide the reference index we have created. Finally we also provide both fastq files for an RNA-seq experiment. Please note that we are providing R2 files first and R1 reads last, although the Tophat2 documentation says we should provide R1 first and R2 second. We do this because the protocol used to produce these data sequence the cDNA, and by switching R1 and R2 we will provide the correct strand orientation to the aligner. Although Tophat2 creates lots of logs it can also be useful to store the terminal output to a log file as well. We do this by redirecting standard out and standard err to a file.

We need to align data from four separate RNA-seq experiments, so we need to execute Tophat2 four times. The following command is for one experiment, but the command file RNASeq/doc/commands.txt contains all four commands. The command file also keeps the commands on a single line which is easier to copy and paste, while the command below spans several lines to be easier to read. We suggest that you copy the four command lines from the commands file and paste them in the terminal.

```
mkdir aligned/logs
tophat -o aligned/U2OS.rep1 --GTF reference/Homo_sapiens.GRCh37.72.gtf
reference/genome/chr19
fastq/U2OS.rep1.R2.fastq
fastq/U2OS.rep1.R1.fastq &> aligned/logs/U2OS.rep1 &
```

You can follow the execution of Tophat2 by monitoring the log files. To follow the execution of the example above, you can type (use control-C to exit):

```
tail -f aligned/logs/U2OS.rep1
```

3.2 Quality Control

Once the alignments are finished, we can inspect the results and perform further analysis within the R environment. We are only showing you how to analyze one data set in this tutorial, but you need to perform the same analysis for the others as well. The most practical way of doing this is to complete this section and the next (visualization) for each data set. To get started we first start up R by typing:

```
R
```

This will display some R startup information, and you are now at the R prompt. The first thing we should do is to load some libraries that we will need later on:

```
library(ShortRead)
library(rtracklayer)
```

We are now ready to start the analysis, so let's load one dataset first and see how it looks like in R:

```
aln <- readBamGappedAlignmentPairs('aligned/U2OS.rep1/accepted_hits.bam', use.names=T)
```

It takes a little time to read all the alignments into memory. For full genome data sets this takes considerably longer time and also requires a lot of memory available on the machine used for analysis. The benefit is that once all the data is in memory, it is very fast to perform calculations and manipulations on the data. Once the data has been read in, you will see two messages being displayed. These refer to alignment pairs which are not considered valid by the readBamGappedAlignmentPairs function. Aligners (like Bowtie) can have different sets of requirements in the determination of which alignments should be considered valid, so the readBamGappedAlignmentPairs applies its own set of requirements to ensure a certain level of quality in the alignments. Briefly, these requirements are: 1) The reads should be paired by id, 2) they should both be mapped, 3) the paired reads should align to opposite strands, and 4) the aligned read pairs should be considered valid by the aligner. The retained alignment pairs are those that have passed all the quality requirements, and thus are useful for further analysis. Let's see how many we have:

```
length(aln)
```

We can compare this number to the total numbers of read pairs we recorded for U2OS replicate 1 previously, and calculate an alignment rate:

```
length(unique(names(aln)))/389139
```

The alignment rate is a common metric for evaluating the quality of an RNA-seq experiment. We wish to see as high an alignment rate as possible, and a successful experiment should be expected to have an alignment ratio above 0.7-0.8.

Another common metric is a library complexity estimate, which reflects the ratio of unique DNA fragments that were sequenced. If a library contains mostly many copies of a few fragments, it has a low complexity, and similarly, a library which contains mostly unique fragments has a high complexity. We can estimate the library complexity by dividing the number of unique aligned pairs with the total number of aligned pairs. One should also expect the library complexity estimate to be at the higher end, close to 1.

```
length(unique(granges(aln)))/length(aln)
```

A third quality control metric is the balance between alignment to forward strand and reverse strand. One would expect a similar amount of alignments to both strands, and if that is not the case, it may indicate that something went wrong in the library prep. To calculate this we simply divide the number of alignments on the forward strand with the number on the reverse strand:

```
sum(strand(aln)=='+') / sum(strand(aln)=='-')
```

Finally, an estimate of the typical DNA fragment length can be calculated from the alignments. This estimate should be close to the fragment length reported the library prep before sequencing. The fragments are sequenced from both ends in paired end sequencing protocols. Unless the fragment span an exon-exon junction, the two reads in a pair will align with an outer distance between them equal to the fragment length. The paired alignment positions can be obtained by a function called granges. By using the width function to these positions, we get the genomic width of the outer alignment positions for an aligned pair. An approximation of the most typical values can then be made by measuring the median of these:

```
median(width(granges(aln)))
```

3.3 Visualization

A common way of visualizing HTS data is to summarize the read coverage for each position in the genome. The coverage is, in other words, a value that indicates how many reads were aligned to any part of the genome. For RNA-seq data we would expect to see high coverage in exons, and low signal in introns and intergenic regions. For stranded RNA-Seq data, like we have here, you would also expect that the reads aligning to a gene aligns to the same strand. See figure 3 for a description of these and other visual evaluation criteria for RNA-Seq data.

We will generate coverage for our data and save the results to files for visualization in a genome browser. Since we have strand specific data, we will save the coverage separately for each strand and visualize them a bit differently. To generate the track files, type:

```
export(coverage(aln[strand(aln)=='+']),  '../tracks/U2OS.rep1.fs.bw', format='bw')
export(coverage(aln[strand(aln)=='-'])*(-1), '../tracks/U2OS.rep1.rs.bw', format='bw')
```

This is a good time to repeat the QC analysis and generate tracks for the other three data sets as well. Make sure you record the QC results for each data set. For the remainder of this tutoria we are assuming that you have completed QC and generated tracks for all four data sets. After completing the analysis for all data sets you can log out of R by typing "quit()" and then "n" at the prompt:

```
quit()
Save workspace image? [y/n/c]: n
```

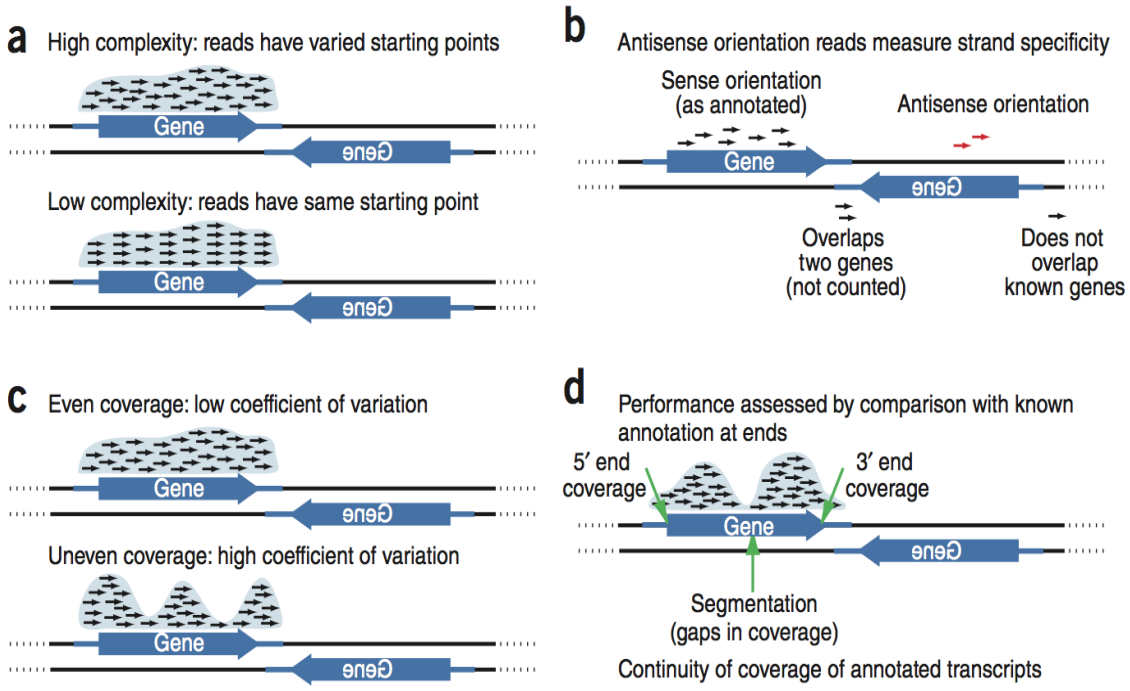

Figure 3: Key criteria for evaluation of strand-specific RNA-seq libraries. Categories of quality assessment were complexity (a), strand specificity (b), evenness of coverage (c) and comparison to known transcript structure (d). Double-stranded genome with gene ORF orientation (blue arrows) and UTRs (blue lines) are shown along with mapped reads (black and red arrows, reads mapped to sense and antisense strands, respectively). Figure and legend from Levin *et al.*, Nature Methods 2010

3.3.1 Using the UCSC Genome Browser

If you are new to the concept of a Genome Browser or have not seen the UCSC Genome Browser before, please go to genome.ucsc.edu, click on "Genome Browser" in the top left corner, and then click on the "submit" button. You are now looking at the default view of the genome browser. A basic concept is a track which represents a particular genome wide data set. The browser window is composed of several tracks in a stack. You can re-order the tracks by drag and drop with your mouse. You can navigate to a specific locus by typing in the search box at the top of the page. It's often a good idea to zoom out several times when looking at a particular region. The UCSC genome browser contains a lot of public data sets. You can see some of them are turned on by default. If you wish to hide a particular track you can right click on it and select "hide". If you wish to look at some more data sets, you can select from the categories below the main browser image. If you need further guidance you may investigate the different alternatives under the help menu at the top of the page.

There are several ways of uploading data to the genome browser, but for our data the best option is to to create a track hub which we will make available to the genome browser. A track hub is a

collection of track files and configuration files, stored within a directory which needs to be available on the the web. To visualize the data we provide the location of this directory to the UCSC genome browser. It will then read the data and generate the tracks.

First make sure you are located in the "RNASeq/tracks" directory. We have already prepared the track hub directory called "RNASeq_Course_TrackHub" with all necessary configuration. We are going to visualize both the raw alignments from the BAM files, and the coverage files we have just created. To complete the creation of the track hub, we need to copy the files into the track hub directory:

```
cd tracks
cp *bw RNASeq_Course_TrackHub/hg19/
cp ../data/aligned/U2OS.rep1/accepted_hits.bam RNASeq_Course_TrackHub/hg19/U2OS.rep1.bam
cp ../data/aligned/U2OS.rep2/accepted_hits.bam RNASeq_Course_TrackHub/hg19/U2OS.rep2.bam
cp ../data/aligned/WT.rep1/accepted_hits.bam RNASeq_Course_TrackHub/hg19/WT.rep1.bam
cp ../data/aligned/WT.rep2/accepted_hits.bam RNASeq_Course_TrackHub/hg19/WT.rep2.bam
```

A track hub works by not sending all the data to the genome browser, which can take a long time, but only sending the data needed. For this to work the genome browser needs to know exactly where the relevant data is located within the bam files. This is accomplished by creating index files that the genome browser can use. We use samtools to create these indices:

```
cd RNASeq_Course_TrackHub/hg19/
samtools index U2OS.rep1.bam
samtools index U2OS.rep2.bam
samtools index WT.rep1.bam
samtools index WT.rep2.bam
```

This completes the construction of the track hub, and we now have to make it available on the web by copying the track hub to a web browser:

```
cd ../..
cp -r RNASeq_Course_TrackHub  /uio/hts-www/hts/username/
```

Here you should replace "username" with the username you are using at the course. After the copy is completed, start the genome browser by navigating to genome.ucsc.edu, and click on "genome browser" at the top left. We add our data by clicking on "track hubs" at the center of the page. Click on "My Hubs" and in the URL field, type

```
http://hts-www.uio.no/hts/username/RNASeq_Course_TrackHub/hub.txt
```

Again, you will need to replace "username" with the username you are using at the course. Click "Add Hub", and then "Use Selected Hubs". We are now using the genome browser to look at a region on chromosome 21. Since we have limited the data set to chromosome 19, you will need to navigate to a region there. Our tracks are the four on top, and the default tracks are below. You can do that by typing

```
chr19:11,000,000-11,050,000
```

in the search filed where it says "enter position, gene symbol or search terms". You are now looking at a region with several expressed genes. Use the criteria mentioned in figure 3 to evaluate the quality of this data.

4 Expression Quantification

We are now ready to quantify gene expression by counting how many reads overlap a gene in each experiment. That means our measurement of expression is the number of reads that align within the genomic locations of the exons of a particular gene.

We are going to read all the alignment into R one more time to perform quantification. In an analysis with larger data sets this would not be very practical and you would rather perform QC, generate tracks and quantify expression for one data set at the time.

To get started with quantification we'll start a new R session again. Make sure you are in the RNASeq/data directory, and start R with:

```
cd ../../../data
R
```

First thing to do in R is to load some libraries we are going to need:

```
library(DESeq)
library(GenomicRanges)
library(rtracklayer)
```

We will need the genomic coordinates of gene models in order to count the overlapping reads. We have already provided this annotation in a GTF file. This file contain gene identifiers, coordinates, and type of gene. To read it into R and inspect the result, type:

```
genes <- import('reference/Homo_sapiens.GRCh37.72.gtf', asRangedData=FALSE)
genes
```

Since we need to quantify the expression from four different samples, we are going to create a function that contains all the steps we need to perform. This will reduce the typing we need to do, as we can use the same function for all four data sets. The function looks like this:

```
overlap.counts <- function(bamfile){

    aln <- readGappedAlignmentPairs(bamfile, use.names=TRUE)

    genehits <- summarizeOverlaps(split(genes, genes$gene_id), aln, mode="Union",
             singleEnd=FALSE, ignore.strand=FALSE)
    counts <- assays(genehits)$counts

    res <- genes[,c('source', 'gene_id','gene_name')]
    res <- res[order(width(ranges(res)), decreasing=T)]
    res <- res[!duplicated(res$gene_id)]
    res$counts <- counts[match(res$gene_id, rownames(counts))]
    res[order(res$counts, decreasing=T)]

    return(as.data.frame(res))
}
```

Essentially this defines a function that excepts a parameter called "bamfile" which is supposed to be the name of a bam file. The contents of this bam file is read into R with the "readGappedAlignmentPairs" function. Overlaps with genes are found by the "summarizeOverlaps function", which use the alignments we just read in, as well as the gene annotation we read in before. This is the function that performs the majority of the work. Note that we provide all features annotated to be a part of each gene, which means that for each gene we are counting the number of overlapping alignments with all isoforms of that gene. We could also perform this analysis on transcript level, or on the exon level. Also note the "mode" parameter. It refers to the mode in which to search for overlap with genes (see figure 4). The next lines extracts basic information from the gene annotation and the alignment count for each gene, and reports the results.

We are now ready to run this function on our data sets. For each data set we provide the alignment bam file, and the function reports a table with the number of alignments per gene:

```
U2OS1 <- overlap.counts('aligned/U2OS.rep1/accepted_hits.bam')
U2OS2 <- overlap.counts('aligned/U2OS.rep2/accepted_hits.bam')
WT1 <- overlap.counts('aligned/WT.rep1/accepted_hits.bam')
WT2 <- overlap.counts('aligned/WT.rep2/accepted_hits.bam')
```

This takes a few moments to run, but soon this should be finished, and you can inspect the results. As these tables will have many rows, it's best to use some R tools to look at the top of a table, and to get a summary of it's content:

```
head(U2OS1)
summary(U2OS1)
```

If you use the summary function on all four data sets, do you see any difference between them? We now have four tables with the expression measurements from each data set. We would like to have all the results into one table. Since the four tables all have the same order of genes, we can merge them into one and remove unexpressed genes by:

```
tab <- data.frame(gene_id=U2OS1$gene_id, gene_name=U2OS1$gene_name, U2OS1=U2OS1$counts,
                  U2OS2=U2OS2$counts, WT1=WT1$counts, WT2=WT2$counts)
 tab <- tab[rowSums(tab[,3:6])>0,]
 head(tab)
```

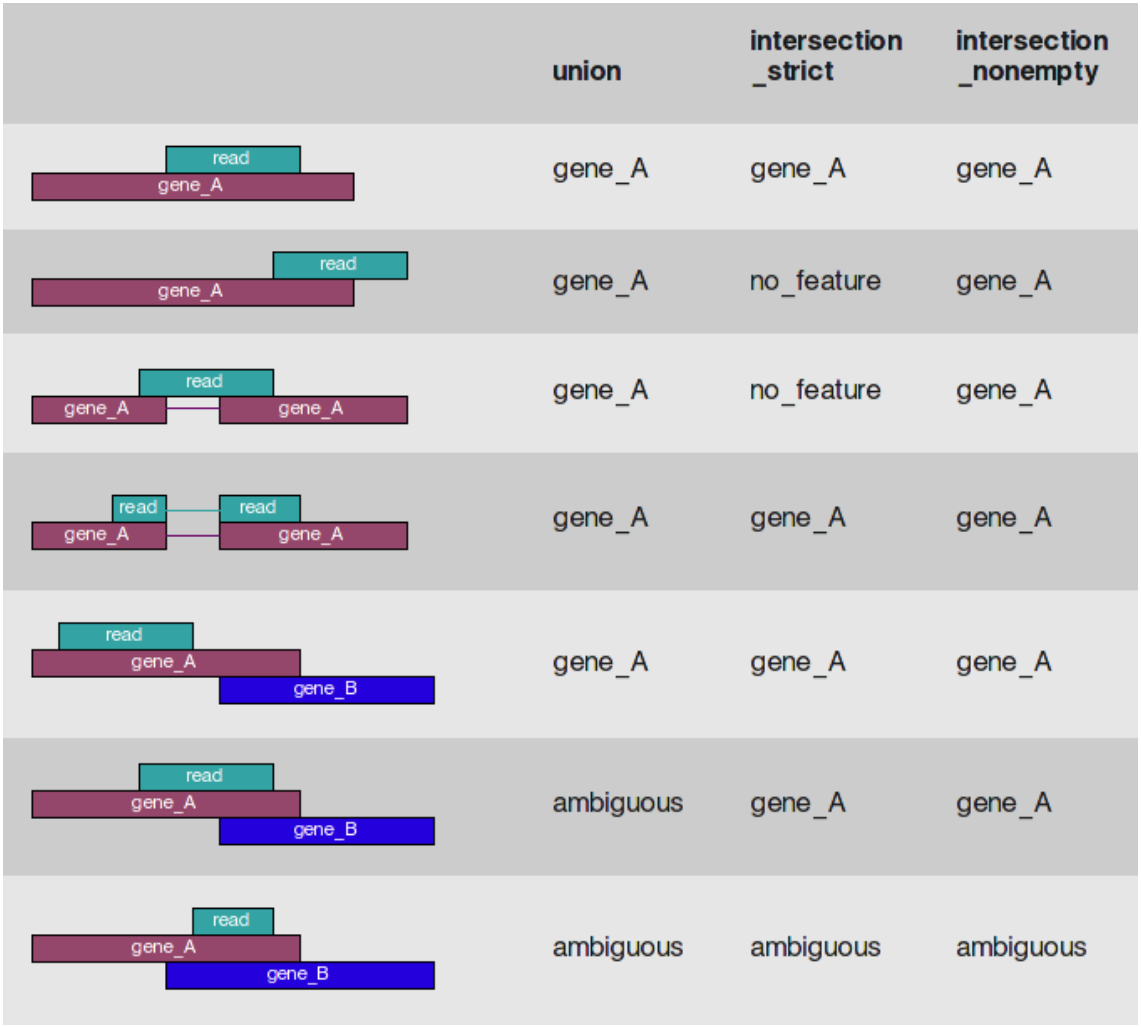

Figure 4: Different modes of counting overlap with gene models. An overlap is regarded as ambiguous when an alignment overlaps with several genes (partially or fully). Ambiguous overlaps are not reported as we can not identify the gene it originates from. Figure downloaded from www-huber.embl.de/users/anders/HTSeq/doc/_images/count_modes.png

As you can see we now have a single table with all expression measurements. Be aware that Bioconductor is developing at a rapid pace, and you should expect better solutions to expression quantification from paired end RNA-seq data to be available soon. The methods presented here will still continue to work even if other solutions are available. Finally, we write the results into a file in case we should need them in the future:

```
write.table(tab, file='expression/quantifications.txt', row.names=F, quote=F, sep="\t")
```

4.1 Quality Control

To evaluate the quality of gene expression measurements from a single experiment you can use any existing knowledge of which genes are expected to be expressed, and verify that we do find these genes being expressed. Unfortunately, we do not have any such knowledge for these experiments, but we can

perform some more general tests. The data we are using are produced by ribosomal depletion, and not polyA selection. This means we will have a combination of coding and non-coding RNA molecules in the sequencing library. We can get an overview of which types of genes are being expressed by running:

```
sort(table(U2OS1$source), decreasing=T)
```

This gives us the frequency of the different types of genes in decreasing order. We would expect to see mostly protein coding genes, but also various types of non-coding RNA. In particular we should not see a high frequency of ribosomal RNA, as this would indicate a failed ribosomal depletion in the library prep.

To get an overview of the relationship between the different experiments we can make a plot using the pairs function:

```
pdf('../plots/pairs.pdf')
pairs(log10(tab[,3:6]), lower.panel=NULL, pch=20, cex=0.5)
dev.off()
```

One would expect to see stronger correlation between replicates then between experiments from different conditions. One should also expect that all replicates appear similar in comparison with experiments from other conditions.

5 Differential Expression Analysis

Differential expression analysis is a very common application of RNA-Seq, and often the motivation for performing the experiments. In the experimental setup for differential expression analysis it's important to have replicates to limit the effect of biological variation. We have two replicates here, which is actually below the recommended number of at least three replicates. We should keep this in mind when evaluating the results from the differential expression analysis.

5.1 Normalization and Significance Estimation

An observation at a gene of a different number of aligned reads across conditions can be caused by several factors in addition to a potential differential expression level. A well known factor is sequencing depth, which naturally leads to a higher number of alignments in the experiment with a higher number of reads. Another factor is variation, or dispersion, across conditions. Variation can be caused by technical reasons or biological variation. Technical variation is usually a constant low level, while biological variation are often higher. The relative contributions of these two sources of variation are different at different numbers of aligned reads, where genes with a low number (i.e. are expressed at a low level) are more affected by technical variation, and genes with a higher number are more affected by biological variation.

To address the issues of different sequencing depth and expression dependent variation, we are going to use the DESeq package. It performs normalization and estimates the significance of differential expression.

```
rownames(tab) <- tab$gene_id
cds <- newCountDataSet(tab[,3:6], c('U2','U2','WT','WT'))
cds <- estimateSizeFactors( cds )
cds <- estimateDispersions( cds, fitType='local' )
res <- nbinomTest( cds, 'U2','WT')
res <- cbind(name=tab$gene_name,res)
```

For some genes there will be zero counts in one condition and not in the other. This may lead to a division by zero in the calculation of log2 fold change. To avoid this problem and get more meaningful results, we re-calculate log2 fold change while adding 1 to each value. This is also a good time to save our results for any future use.

```
res$log2FoldChange <- log2(res$baseMeanB+1)-log2(res$baseMeanA+1)
write.table(res, file='expression/expression.txt', row.names=F, quote=F)
```

5.2 Quality Control

As for evaluating a single experiment, it is also useful when evaluating differential expression to apply any existing knowledge of which genes should be expected to be misrelated. There are at least two types of plots commonly used for visual evaluation of differential expression. The first plot commonly used was the MA plot which is a type of scatter plot with log10 mean expression level across all experiments on the x-axis and log2 fold change on the y-axis. We can use a built-in function for generating this type of plot, and provide it with some additional parameters:

```
sig <- res$padj <= 0.01 & abs(res$log2FoldChange)>=1
pdf('../plots/MAplot.pdf')
plotMA(res, cex=0.7, col=ifelse(sig,"red3","gray32"))
dev.off()
```

We use two types of thresholds in defining differentially expressed gene: The adjusted p-values and log2 fold change. The exact values appropriate to use may vary, but we use p-value <=0.01 and absolute log2 fold change >=1 for this analysis. The data points in an MA plot should usually be centered around zero on the y-axis. If there is a large shift or tendency away from zero, it may indicate something went wrong in the normalization procedure. The data points should in general form the shape of a christmas tree laying horizontally.

The second type of plot is the Volcano plot, which is also a type of scatter plot. It has log2 fold change on the x-axis and -log10 p-values on the y-axis. It therefore complements the MA plot by also incorporating the significance estimate.

```
pdf('../plots/Volcano.plot.pdf')
plot(res$log2FoldChange, -log10(res$padj), pch=20, col=ifelse(sig, "red3","gray32"))
abline(v=0, lwd=4, col="#ff000080")
dev.off()
```

The data points in a volcano plot are typically more scattered then in an MA plot, and the overall impression of the plot shoulde usually something that may resemble a volcano eruption.

5.3 Functional Annotation Enrichment Analysis

There are large amounts of functional information registered in various databases for most genes, in particular for well annotated genomes like human and common model organisms. Examples of functional annotation information are Gene Ontology, KEGG pathways, decease association, literature, protein domains, and protein interactions. Functional annotation enrichment analysis takes advantage of these types of information and search for any statistical overrepresentation within a given set of genes. This type of analysis is also called gene set enrichment analysis.

It can be useful to separate up-regulated from down-regulated genes in this analysis, so we will generate two lists of gene identifiers. We define up-regulated genes to be those that are expressed above a given level (baseMean>10) and significantly higher expressed. We do the same for down-regulated genes:

```
upregulated <- res[res$baseMean>10 & res$padj <=0.05 & res$log2FoldChange>1,]
write.table(upregulated$id, file='expression/upregulated.ids.txt', row.names=F,
            col.names=F, quote=F)

downregulated <- res[res$baseMean>10 & res$padj <=0.05 & res$log2FoldChange<(-1),]
write.table(downregulated$id, file='expression/downregulated.ids.txt', row.names=F,
            col.names=F, quote=F)
```

We are going to use the Database for Annotation, Visualization and Integrated Discovery (DAVID) tool for this analysis. We start by navigating to david.abcc.ncifcrf.gov. Click on "Start Analysis" in the top menu, and then on "Upload" at the top of the page. Click on "Choose File" and select the file called upregulated.ids.txt. Then select "ENSEMBL_GENE_ID" from the drop-down box below, and "Gene List" below that. Then click on "Submit List", and "Functional Annotation Clustering" at the center of the page. This page shows the list of available annotation data and let's you choose which ones you would like to include in the analysis. We are going to use the default data sets and continue by clicking on "Functional Annotation Clustering". This results in a new browser window which contain the results of the analysis. The results are clustered by related terms, and each cluster has an enrichment score. This score does not indicate significance, but is used to rank the results. Each individual term has a significance value represented by the Benjamini column to the right.

6 Transcriptome Assembly

It may not be a suitable option to quantify expression using existing genome annotation when it is far from complete, which is commonly the case when not working with human or mouse. In these cases it may be a better option to initially perform transcriptome assembly to identify the expressed transcripts from the RNA-Seq data. The results can be used to increase the quality of genome annotation, and for some organisms also for improving the genome assembly. There are two basic approaches to transcriptome assembly; guided and unguided (de novo). Guided transcriptome assembly takes advantage of the genome sequence by performing an initial genomic alignment, and using that to reconstruct the expressed transcripts. Unguided transcriptome assembly is more similar to genomic assembly, and attempts to reconstruct the expressed transcripts only from the sequenced reads.

6.1 Running Cufflinks

We are going to use the Cufflinks tool for performing guided transcriptome assembly and expression quantification. Make sure you have quit the R session, if you have not already done so, by typing:

```
quit()
Save workspace image? [y/n/c]: n
```

While still in the data directory, type the following command to run cufflinks for guided transcriptome assembly:

```
cufflinks --frag-bias-correct reference/genome/chr19.fa --multi-read-correct
          --pre-mrna-fraction 0.3 --output-dir assembly/U2OS1
          aligned/U2OS.rep1/accepted_hits.bam
```

The fragment bias correction parameter is used to improve the quantification by compensating for sequence specific bias in random priming during the library prep. We provide the genome sequence to enable cufflinks to compensate for this bias. We also instruct cufflinks to use it's multi read correction to further improve quantification. It will then try to decide the best way to allocate reads that align to multiple locations in the genome. From previous experience we know that in this data set there are often some reads aligning to introns, indicating that they originate from unspliced pre-mRNA. To

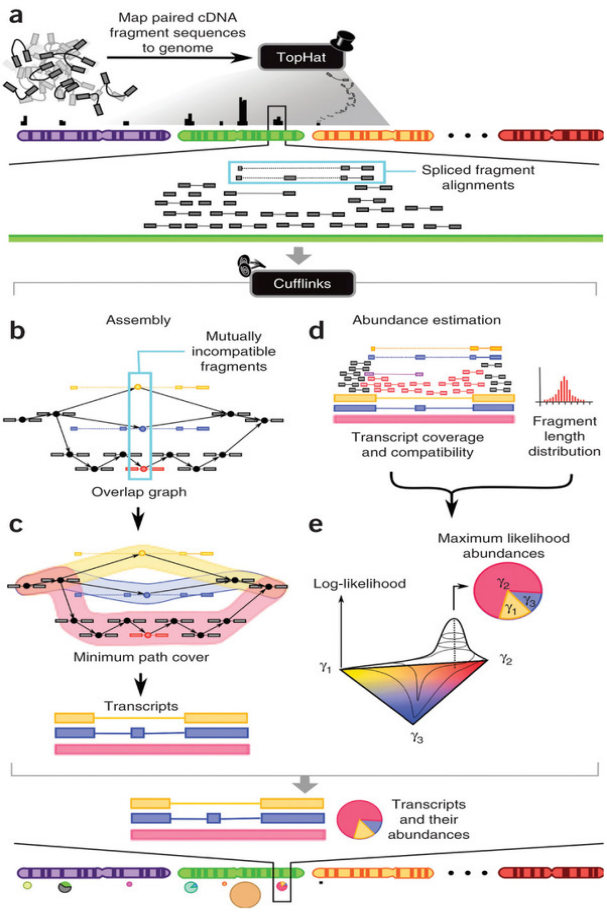

Figure 5: TopHat pipeline. Trapnell *et al.*, 2010

avoid using these reads in the transcriptome assembly, we increase the default setting to 0.3 to make cufflinks ignore more of such reads. Finally we tell cufflinks where to put the results and where to find the genome alignment file.

After completion, we navigate to the cufflinks output directory to inspect the results:

```
cd assembly/U2OS1/
ls -lh
-rw-rw-r-- 1 user user 115K Aug 28 12:14 genes.fpkm_tracking
-rw-rw-r-- 1 user user 161K Aug 28 12:14 isoforms.fpkm_tracking
-rw-rw-r-- 1 user user    0 Aug 28 12:11 skipped.gtf
-rw-rw-r-- 1 user user 2.5M Aug 28 12:14 transcripts.gtf
```

As you can see, there are four output files produced by cufflinks. The transcripts.gtf file contains the transcripts found by cufflinks, and the fpkm files contain expression quantification measurements on the gene and isoform levels.

6.2 Quality Control

To aid our evaluation of the Cufflinks transcripts, we can upload them to the browser as a track and have a look at them. To do that, go to the genome browser and click on the "add custom tracks"

button at the center of the page, just below the main image. Then click on "Choose File" and navigate to the data/assembly/U2OS1 directory and select the transcripts.gtf file. Then click on the "Submit" button, and when the file has loaded, click on "go to genome browser". You will now see the Cufflinks transcripts at the top of the page. It is often useful to zoom out to see the contexts of a transcript.

Since we already have a good quality genome annotation, it would be interesting to see cufflinks have identified any new transcripts. We can compare the cufflinks transcripts with the existing genome annotation with the cuffcompare tool. To run cuffcompare, we go back to the terminal and type:

```
cuffcompare -r ../../reference/Homo_sapiens.GRCh37.72.gtf -R transcripts.gtf
```

Cuffcompare produce six more files, but we're just going to use three of them. First we'll look at the "cuffcmp.stats" file which contains some summary statistics of the comparison:

```
cat cuffcmp.stats
```

In this file you'll see that Cufflinks reports sensitivity and specificity on several levels (from base to locus), and also includes fuzzy values for each level. It also reports missed and novel exons, introns and loci. To investigate potential new transcripts from Cufflinks, we should look in the file called "cuffcmp.transcripts.gtf.tmap". To see how this file looks like, type:

```
head cuffcmp.transcripts.gtf.tmap
```

This file contains a mix of annotated and Cufflinks transcripts, where Ensembl transcript identifiers starts with "ENST" and Cufflinks transcripts instead have the "-" character in the same column. In the third column called "class_code", there is always an "u" character for transcripts not similar to any annotated transcript. We can use this code to filter out new transcripts by running:

```
grep ^-.*u.*CUFF cuffcmp.transcripts.gtf.tmap | sort -nrk 7 >novel.transcripts.txt
```

This extracts the potentially novel Cufflinks transcripts and sorts them by expression level (the FPKM column). We can get an impression of this file by looking at the first lines, and count how many there are:

```
head novel.transcripts.txt
wc -l novel.transcripts.txt
```

The file called cuffcmp.loci contains all the genomic locations of these transcripts, so we can look up the Cufflinks ids in this file to get their locations. For example the first transcript:

```
grep CUFF.821 cuffcmp.loci
```

We can look at the specific loci in the genome browser to evaluate the Cufflinks transcript predictions. To inspect a loci, type it's genomic coordinates in the search box. In this case it would be:

```
chr19:36066507-36066636
```

6.3 Comparison of Expression

In addition to transcriptome assembly, Cufflinks also provide quantification of gene expression levels. We can compare these with the expression levels we produced in the previous section. To do that, we can plot the results against each other in a scatterplot. We first need to start R and load the data sets:

```
R
cuff <- read.table('cuffcmp.transcripts.gtf.tmap', header=T, sep="\t")
bioc <- read.table('../../expression/expression.txt', header=T)
head(cuff)
head(bioc)
```

To compare the expression values from these two data sets, we need to match the identifiers of each data set, and convert the FPKM values back into read counts:

```
cuff.matched <- cuff[match(bioc$name,cuff$ref_gene_id),]
cuff.matched$reads <- cuff.matched$FPKM*0.38*(cuff.matched$len/1000)
head(cuff.matched)
head(bioc)
```

We can now plot the two data sets against each other in a scatter plot:

```
pdf('../../../plots/bioc.cuff.comparison.pdf')
plot(log10(bioc$baseMeanA+1), log10(cuff.matched$reads+1), pch=20,
    xlab='BioConductor', ylab='Cufflinks')
dev.off()
```

What can you conclude from looking at the plot? You may have noticed that in addition to the general trend, there is a cluster of outliers which have zero values by Cufflinks but significant expression by BioConductor. We can investigate these outliers by selecting them from the data sets:

```
bioc[ cuff.matched$reads==0 & !is.na(cuff.matched$reads) & bioc$baseMeanA>0 ,]
cuff.matched[ cuff.matched$reads==0 & !is.na(cuff.matched$reads) & bioc$baseMeanA>0 ,]
```

We can now see the gene identifiers, and we can look up these regions by typing or copy/paste the identifiers into the search field of the genome browser. What is you impression of the expression of these genes?

7 Environment

We have set up an environment that contains everything you need, but to perform similar analyses on your own you will need to set this up for yourself. This is a description of the environment that is meant to point you in the right direction. This setup is meant to be installed on a computer with either a Linux distribution or OSX.

Task	Software	URL
Fastq file Quality Control	FastQC	www.bioinformatics.babraham.ac.uk/projects/fastqc
Genome Alignment	Tophat	tophat.cbcb.umd.edu
Data Analysis and Plotting	R	cran.r-project.org
Transcriptome Assembly	Cufflinks	cufflinks.cbcb.umd.edu

In addition, you will need several R packages that are not included in the standard R installation. To install these packages you will need to run some commands from inside R. After installing the latest version of R, open a terminal and run R by typing "R". You are now in the R shell. Type the following commands:

```
source("http://www.bioconductor.org/biocLite.R")
biocLite("DESeq")
biocLite("ShortRead")
biocLite("rtracklayer")
```