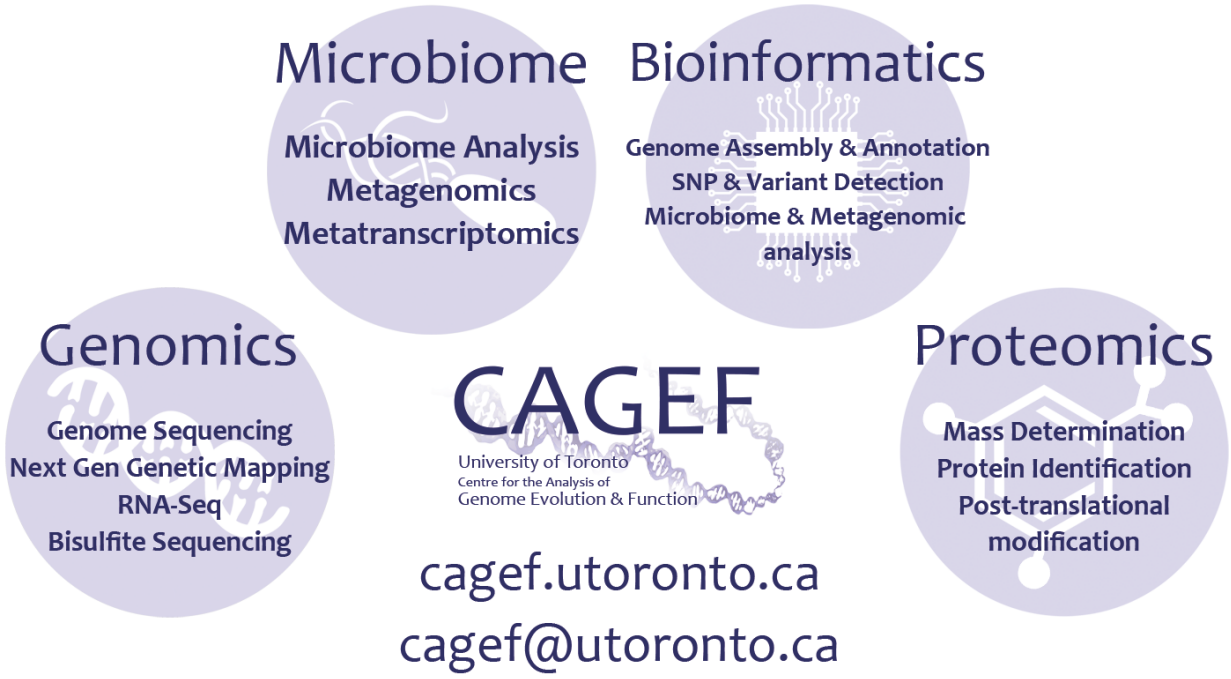


CSB1021HF LEC0131

FUNDAMENTALS OF GENOMIC DATA SCIENCE

0.0.0 Module 7: Variant calling and RNA-Seq in the command line

0.1.0 About Fundamentals of Genomic Data Science

Fundamentals of Genomic Data Science is brought to you by the **Centre for the Analysis of Genome Evolution & Function (CAGEF)** bioinformatics training initiative. This course was developed based on feedback on the needs and interests of the Department of Cell & Systems Biology and the Department of Ecology and Evolutionary Biology.

The structure of this course is a “code-along”, hands-on style! A few hours prior to each lecture, materials will be made available for download at QUERCUS (https://q.utoronto.ca/). The teaching materials will consist of a weekly PDF that you can use to follow along with the instructor along with any datasets that you’ll need to complete the module. This learning approach will allow you to spend the time coding and not taking notes!

As we go along, there will be some in-class challenge questions for you to solve. Post lecture assessments will also be available for each module, building upon the concepts learned in class (see syllabus for grading scheme and percentages of the final mark).

0.1.1 Where is this course going?

We’ll take a blank slate approach here to learning genomic data science and assume you know nothing about programming or working directly with next generation sequencing data. From the beginning of this course to the end we want to guide you from potential scenarios like:

- You don’t know what to do with a set of raw sequencing files fresh from a facility like.

- You've been handed a legacy pipeline to analyse your data or maintain for the lab but you don't know what it runs or how.
- You plan on generating high-throughput data but there are no bioinformaticians around to help you out.

and get you to the point where you can:

- Recognize the basic tools in sequence analysis.
- Plan and write your own data analysis pipelines.
- Explain your data analysis methods to labmates, supervisors, and other colleagues.

0.1.2 How do we get there?

In the first half of this course, we'll focus on how to generate analysis pipelines using the Galaxy platform – a user-friendly graphical interface that provides access to common sequence analysis tools. After we are comfortable with these tools, we'll look at life through the lens of a command-line interface. It is here that we will learn the basics of file manipulation and how to program scripts that can carry out multiple tasks for us. From there we'll revisit tools from the first half and learn skills to make your data analysis life easier.

0.2.0 Goals of the module

1. Learn how to perform and analyze reference alignments at the command line with bowtie2 and HISAT2.
2. Learn how to perform variant calling at the command line with samtools and bcftools.
3. Learn how to analyze RNA-Seq datasets at the command line with featureCounts and in R with DESeq2.

0.3.0 Pre-class modules with Coursera

Each week we strongly encourage you to complete the assigned Coursera modules and/or readings ***before*** class. These are meant to provide you with sufficient background material on each week's module so that we can focus on the act of "doing" something with that data rather than spend a lot of time on the origins of it. You'll find a section outlining the next set of Coursera modules and readings at the end of each module.

0.3.1 Go to www.coursera.org and sign up for an account with your e-mail.

0.3.2 Search the following courses and enroll to audit each course (audit):

- Genomic Data Science with Galaxy, Johns Hopkins University.
- Command Line Tools for Genomic Data Science, Johns Hopkins University.

0.4.0 Setting up your working directory

We suggest that you create a new directory (folder) for this course directly off your root directory called "**FGDS**". Working from your root directory is not necessary, but it will make some of the aspects of the course a little easier to manage. For Mac OS users, we suggest you create this as a subfolder in your **user** directory.

0.4.1 Within this directory, create another directory called "**Module7**". This is where we will store the data used in this week's module.

0.4.2 Create a subdirectory called "**Data**" to store the initial files as we download them before decompressing and working with them in later steps.

1.0.0 Generating reference alignments with Bowtie2

This week we revisit our data from Modules 4 and 6 to generate a reference alignment with **bowtie2** much like we did in Galaxy. Unlike with Galaxy, we will be working with the trimmed and filtered paired-end data generated during Module 6.

1.1.0 Directory and data setup

Before you begin, open your command line and make sure you have a path for **~/FGDS/Module7/Data**. This is where we will store some data used and generated in today's lecture.

1.1.1 Check for the FGDS/Module7/Data path. If necessary, create it before setting it as your current directory.

```
cd ~/FGDS
ls ./Module7/Data       # This will return an error if the directory doesn't exist
mkdir -p Module7/Data         # -p will make parent folders for you if necessary
cd Module7
```

1.1.2 You have already downloaded the files that you will need for this section during Module 4 and Module 6. Copy the complete *H. influenzae* str. KW20 reference genome and the input fastq files into your **Module7** directory. As was the case in Module 6, we will be using paired-end data in this module. This can be particularly useful for identifying longer structural variants between the reference genome and the sequencing data, but we won't be examining those today.

```
ls ../Module4
cp ../Module4/HinfKW20_genomic.fna ./Data/HinfKW20_ref.fa
ls ../Module6/Data
cp ../Module6/Data/HinfKW20_trimmed_*_paired.fastq ./Data
```

1.2.0 Run bowtie2 from Anaconda

Recall that we installed the **bowtie2** aligner into an Anaconda environment **alignersENV**. We don't need to prepare anything additional with our fastq data since it has already been trimmed and filtered.

1.2.1 Activate your anaconda manager if it has not already been activated when you log into your terminal. Activate your **alignersENV**.using the **--stack** option. While Anaconda will deactivate environments before activating new ones, the **--stack** option will allow you to nest environments, keeping them within your **PATH** instead of deactivating.

```
conda activate                          # Activate your base environment
                                        # Add in your aligner environment
```

1.2.2 Review the command-line options for **bowtie2**. Defaults can be used for most of the options that are available here, but we want to make sure we are performing a paired-end, end-to-end alignment. Local options for alignment are available but, particularly with short reads, this can result in a lot of misalignments.

```
bowtie2 -h | less
```

1.2.3 Review the command-line options for **bowtie-build**. This command indexes your reference genome much like a book can be indexed. Indexing is used to compress and order the information in the reference genome so it can be efficiently searched. This step was performed automatically on Galaxy.

```
bowtie2-build -h
```

Note that an abbreviated help menu for most bioinformatics software can also be accessed by simply typing the command without any options or input, but recall that for **fastqc** this will launch the GUI version of the software instead.

1.2.4 Index your reference genome using **bowtie2-build**, then check the output files that are produced.

Command	Meaning
`HinfKW20_ref.fa`	Reference fasta file
`HinfKW20_ref`	User defined filename prefix for index files

```
ls -la
```

```
Total time for backward call to driver() for mirror index: 00:00:01
(alignersENV) mokca@LAPTOP-7LF6OG94:~/FGDS/Module7$ ll
total 11188
drwxr-xr-x 1 mokca mokca    4096 Dec 10 15:24 ./
drwxrwxrwx 1 mokca mokca    4096 Dec 10 14:37 ../
drwxr-xr-x 1 mokca mokca    4096 Dec 10 14:41 Data/
-rw-r--r-- 1 mokca mokca 4805877 Dec 10 15:24 HinfKW20_ref.1.bt2
-rw-r--r-- 1 mokca mokca  457512 Dec 10 15:24 HinfKW20_ref.2.bt2
-rw-r--r-- 1 mokca mokca     989 Dec 10 15:24 HinfKW20_ref.3.bt2
-rw-r--r-- 1 mokca mokca  457506 Dec 10 15:24 HinfKW20_ref.4.bt2
-rw-r--r-- 1 mokca mokca 4805877 Dec 10 15:24 HinfKW20_ref.rev.1.bt2
-rw-r--r-- 1 mokca mokca  457512 Dec 10 15:24 HinfKW20_ref.rev.2.bt2
(alignersENV) mokca@LAPTOP-7LF6OG94:~/FGDS/Module7$
```

1.2.5 Build a **bowtie2** command for a paired-end, end-to-end alignment. This command will produce a number of warnings as a result of a subset of aggressively trimmed reads in your fastq files. These reads won't be aligned. You can suppress these warnings with the **--quiet** option, but we don't want to do this because it will also suppress the standard output that summarizes the alignment.

```
bowtie2 --end-to-end --sensitive --threads 4 -x HinfKW20_ref -1
     Data/HinfKW20_trimmed_for_paired.fastq -2
     Data/HinfKW20_trimmed_rev_paired.fastq
```

Command	Meaning
`--end-to-end`	Require that each matched read aligns end-to-end. No clipping in our matches.
`--sensitive`	Set the type of end-to-end alignment. This is the default end-to-end mode.
`--threads 4`	Using additional threads (CPUs) will speed up the alignment.
`-x HinfKW20_ref`	The index filename prefix generated from `bowtie2-build`.

`-1 Data/HinfKW20_trimmed_for_paired.fastq`	File with mates, paired to files in `-2` file.
`-2 Data/HinfKW20_trimmed_rev_paired.fastq`	File with mates, paired to files in `-1` file.
`-S HinfKW20_pe_align.sam`	Sets output for SAM format using file name provided.

```
Warning: skipping mate #2 of read 'SRR065202.999254 HWI-EAS367:1:10:954:1459 length=42'
Warning: skipping mate #2 of read 'SRR065202.999254 HWI-EAS367:1:10:954:1459 length=42'
979232 reads; of these:
  979232 (100.00%) were paired; of these:
    11430 (1.17%) aligned concordantly 0 times
    874345 (89.29%) aligned concordantly exactly 1 time
    93457 (9.54%) aligned concordantly >1 times
    ----
    11430 pairs aligned concordantly 0 times; of these:
      2904 (25.41%) aligned discordantly 1 time
    ----
    8526 pairs aligned 0 times concordantly or discordantly; of these:
      17052 mates make up the pairs; of these:
        8185 (48.00%) aligned 0 times
        5201 (30.50%) aligned exactly 1 time
        3666 (21.50%) aligned >1 times
99.58% overall alignment rate
(alignersENV) mokca@LAPTOP-7LF6OG94:~/FGDS/Module7$
```

1.2.6 Review the output information about the success of the alignment on the screen, then take a look in your output SAM file with the **`less`** command (remember the **`-S`** option will keep **`less`** from wrapping the text). Note that unlike the alignment that we performed in Module 3, we now also have the paired-end information for each read where the alignment of both pairs was successful.

```
less -S HinfKW20_pe_align.sam
```

Remember that the columns in the SAM file represent:

- QNAME: Read name
- FLAG: Bitwise flag that categorizes the read (eg. Unmapped or mapped)
- RNAME: Reference sequence
- POS: Leftmost mapping position
- MAPQ: Mapping quality
- CIGAR: Cigar string providing information on gaps in the alignment
- MRNM/RNEXT: Reference sequence of the mate (paired-end only)
- MPOS/PNEXT: Reference position of the mate (paired-end only)
- ISIZE/TLEN: Template length based on position of the mate (paired-end only)
- SEQ: Read sequence string
- QUAL: Read quality string (Phred scores)
- OPT: Optional fields

Note the first 3 lines of our file begin with @. These are part of the header of the file, containing reference genome information like its length. Notice also that the files are listed in the same order as they appear within the fastq files we provided.

2.0.0 Variant calling with SAMtools and BCFtools

Previously in Galaxy we generated BAM files but could view them because the Galaxy interface could convert it automatically from the binary format to the human-readable SAM version. You'll note that the SAM file we created is nearly 400MB of data since it also includes all the sequence information from our fastq files.

2.1.0 Convert your SAM to BAM format with `samtools view`

To convert our SAM file to a BAM file we'll have to dig into a few other **`samtools`** commands. The first step of converting is rather simple and we can use the **`samtools view`** command.

2.1.1 Remember we are currently in our **`alignersENV`** and **`samtools`** is located in **`sambcftoolsENV`**. Activate the correct environment by nesting with the current environment.

```
conda activate --stack sambcftoolsENV
samtools view
```

2.1.2 Review the help menu for **`samtools view`**.

```
samtools view
```

2.1.3 Use the **`samtools view`** command to convert your SAM file to a BAM file. While we provide a reference sequence, we strictly do not need it since our SAM file already has a proper header with a genome size.

```
```

Command	Meaning
`-b`	Output in bam format
`-T`	Provide additional reference data. This is most useful when your SAM file is missing header information which the BAM file needs to be generated.
`HinfKW20_ref.fa`	Reference fasta file
`HinfKW20_pe_align.sam`	SAM alignment file to convert
`> HinfKW20_pe_align.bam`	Send standard output to this file path

```
ls -la
```

Notice that the size of our BAM file is nearly ¼ that of the SAM file now that it has been converted to binary format.

2.2.0 `sort` and `index` your BAM file for variant analysis

While we now have a BAM file, we should consider what we plan to do with the BAM file. In our case, we want to perform some variant calling but that requires our BAM files to be sorted - ie all of the sequencing reads must be ordered by their location with the reference genome. Furthermore, if we plan on importing the BAM into a program like IGV, it will need to be indexed as well. You might recall working with our **.bam/.bam.bai** files in IGV.

2.2.1 Review the help menu for the **`samtools sort`** command and generate a sorted version of our BAM file. Remember that our original SAM file was sorted by order of appearance in our fastq data files.

```
samtools sort
```

2.2.2 Review the help menu for the **`samtools index`** command and generate an indexed version of our sorted BAM file. By default, the **`samtools index`** command will create a **.bai** file.

```
samtools index
```

```
ls -la
```

```
(sambcftoolsENV) mokca@LAPTOP-7LF6OG94:~/FGDS/Module7$ ls -la
total 571844
drwxr-xr-x 1 mokca mokca      4096 Dec 11 00:53 .
drwxrwxrwx 1 mokca mokca      4096 Dec 10 14:37 ..
drwxr-xr-x 1 mokca mokca      4096 Dec 10 17:03 Data
-rw-r--r-- 1 mokca mokca 102253764 Dec 10 17:04 HinfKW20_pe_align.bam
-rw-r--r-- 1 mokca mokca 392791282 Dec 10 15:33 HinfKW20_pe_align.sam
-rw-r--r-- 1 mokca mokca  78842475 Dec 11 00:52 HinfKW20_pe_align.sort.bam
-rw-r--r-- 1 mokca mokca      5624 Dec 11 00:53 HinfKW20_pe_align.sort.bam.bai
-rw-r--r-- 1 mokca mokca   4805877 Dec 10 15:24 HinfKW20_ref.1.bt2
-rw-r--r-- 1 mokca mokca    457512 Dec 10 15:24 HinfKW20_ref.2.bt2
-rw-r--r-- 1 mokca mokca       989 Dec 10 15:24 HinfKW20_ref.3.bt2
-rw-r--r-- 1 mokca mokca    457506 Dec 10 15:24 HinfKW20_ref.4.bt2
```

2.3.0 Generate an mpileup file of your alignment with `samtools mpileup`

Just as we did in Galaxy, we can create an mpileup file to view the sorted results of our BAM file. It can give us a general idea of how the raw fastq reads line up against the reference genome.

2.3.1 Begin by reviewing the **`samtools mpileup`** command.

```
samtools mpileup
```

2.3.2 Using **`samtools mpileup`** requires that the BAM alignment file has been sorted, but it will index the file for you if you haven't done so already. These are the files that are going to be used as inputs for the majority of SNP calling pipelines. In practice, multiple SNP identification and alignment pipelines should be used to verify the calls that are made.

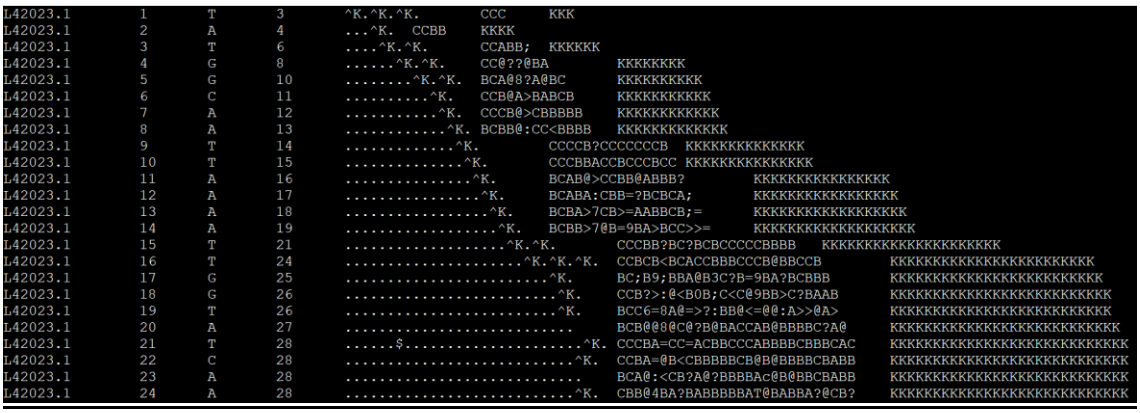
Command	Meaning
`-s`	Output the **mapping** quality scores
`-a`	Output all positions, even if they have no coverage
`-f Data/HinfKW20_ref.fa`	Indexed reference sequence file
`HinfKW20_pe_align.sort.bam`	Input sorted bam alignment file
`-o HinfKW20_pe_align.mpileup`	Output file name

2.3.3 Review the mpileup output file and note the column structure.

```
less HinfKW20_pe_align.mpileup
```

The mpileup columns are:

1. Reference chromosome/contig
2. Reference position
3. Reference base
4. Reads covering the site
5. String of all aligned bases
6. String of all base qualities
7. Mapping quality scores



```
L42023.1        1       T       3       ^K.^K.^K.       CCC     KKK
L42023.1        2       A       4       ...^K.  CCBB    KKKK
L42023.1        3       T       6       ....^K.^K.      CCABB;  KKKKKK
L42023.1        4       G       8       ......^K.^K.    CC@??@BA        KKKKKKKK
L42023.1        5       G       10      ........^K.^K.  BCA@8?A@BC      KKKKKKKKKK
L42023.1        6       C       11      ..........^K.   CCB@A>BABCB     KKKKKKKKKKK
L42023.1        7       A       12      ...........^K.  CCCB@>CBBBBB    KKKKKKKKKKKK
L42023.1        8       A       13      ............^K. BCBB@:CC<BBBB   KKKKKKKKKKKKK
L42023.1        9       T       14      ............^K.         CCCCB?CCCCCCCB  KKKKKKKKKKKKKK
L42023.1        10      T       15      .............^K.        CCCBBACCBCCCBCC KKKKKKKKKKKKKKK
L42023.1        11      A       16      ..............^K.       BCAB@>CCBB@ABBB?        KKKKKKKKKKKKKKKK
L42023.1        12      A       17      ...............^K.      BCABA:CBB=?BCBCA;       KKKKKKKKKKKKKKKKK
L42023.1        13      A       18      ................^K.     BCBA>7CB>=AABBCB;=      KKKKKKKKKKKKKKKKKK
L42023.1        14      A       19      .................^K.    BCBB>7@B=9BA>BCC>>=     KKKKKKKKKKKKKKKKKKK
L42023.1        15      T       21      ...................^K.^K.       CCCBB?BC?BCBCCCCCBBBB   KKKKKKKKKKKKKKKKKKKK
L42023.1        16      T       24      ....................^K.^K.^K.   CCBCB<BCACCBBBCCCB@BBCCB        KKKKKKKKKKKKKKKKKKKKK
L42023.1        17      G       25      ......................^K.      BC;B9;BBA@B3C?B=9BA?BCBBB       KKKKKKKKKKKKKKKKKKKKK
L42023.1        18      G       26      .......................^K.     CCB?>:@<B0B;C<C@9BB>C?BAAB      KKKKKKKKKKKKKKKKKKKKKK
L42023.1        19      T       26      .......................^K.     BCC6=8A@=>?:BB@<=@@:A>>@A>      KKKKKKKKKKKKKKKKKKKKKK
L42023.1        20      A       27      ........................        BCB@@@@C@?B@BACCAB@BBBBC?A@     KKKKKKKKKKKKKKKKKKKKKKK
L42023.1        21      T       28      ......§.....................^K. CCCBA=CC=ACBBCCCABBBBCBBBCAC    KKKKKKKKKKKKKKKKKKKKKKKKK
L42023.1        22      C       28      ........................^K.    CCBA=@B<CBBBBBCB@B@BBBBCBABB    KKKKKKKKKKKKKKKKKKKKKKKKK
L42023.1        23      A       28      ........................        BCA@:<CB?A@?BBBBAc@B@BBCBABB    KKKKKKKKKKKKKKKKKKKKKKKKK
L42023.1        24      A       28      .........................^K.    CBB@4BA?BABBBBBAT@BABBA?@CB?    KKKKKKKKKKKKKKKKKKKKKKKKK
```

2.4.0 Generate a VCF file from BAM files with `bcftools mpileup`

Recall from Galaxy that we mentioned that older versions of **`samtools mpileup`** could produce VCF output. As with Module 3, since we are working with newer versions of **`samtools`** and **`bcftools`**, we turn to **`bcftools mpileup`** to generate a variant call format file.

2.4.1 Begin by reviewing the **`bcftools mpileup`** command.

```
bcftools mpileup
```

2.4.2 **`bcftools mpileup`** also requires that the bam alignment file has been sorted but it will also index the file for you if you haven't done so already.

```
bcftools mpileup -f Data/HinfKW20_ref.fa HinfKW20_pe_align.sort.bam -o
      HinfKW20_pe_align.vcf
```

Command	Meaning
`-f Data/HinfKW20_ref.fa`	Indexed reference sequence file
`HinfKW20_pe_align.sort.bam`	Input sorted bam alignment file
`-o HinfKW20_pe_align.vcf`	Output file name

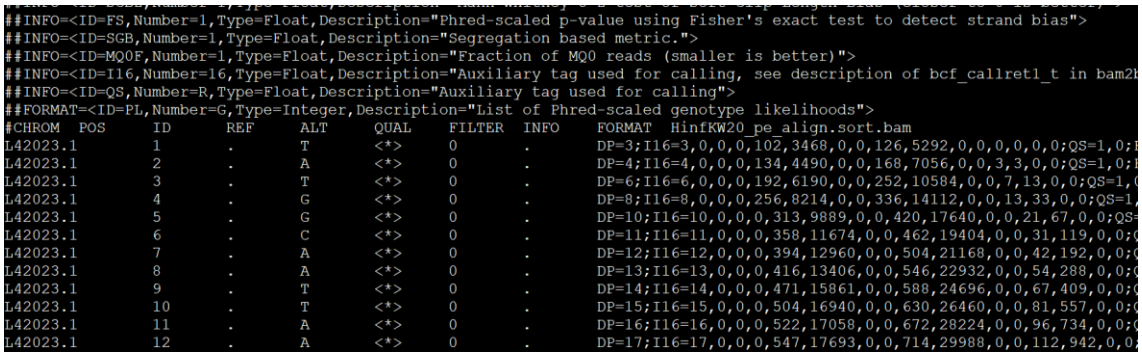
2.4.3 Review the VCF output file and note the column structure looks very similar to that of an mpileup file. The major noticeable differences include additional header information denoted by lines beginning with "# #";

```
less HinfKW20_pe_align.vcf
```

The VCF columns are:

1. **CHROM**: Reference chromosome/contig
2. **POS**: Reference position. Note that there may be multiple entries with the same position.
3. **ID**: Identifier of the variant. A semicolon-separated list of unique identifiers where available.
4. **REF**: Reference base(s)
5. **ALT**: Alternative base(s) or <*> (homozygous reference site)
6. **QUAL**: A Phred-scaled quality score for the assertion made in "Alt".
7. **FILTER**: Filter status of the call. Usually PASS if the position has passed all filters.
8. **INFO**: A semicolon-separated list of additional information with codes listed in the metadata.
9. **FORMAT**: An optional genotype information format descriptor consisting of many colon-separated fields. The codes describe the type of data and its order in the remaining columns.
10. **<sample_name>**: Genotype data field(s) if **FORMAT** is specified.

* Missing values are specified with a dot (.). See VCF 4.2 specifications section 1.4.2 for more genotype information.



2.5.0 Filter VCF for variants only with `bcftools call`

Notice again in your output that you are getting base calls at every position. While good for confirmation, again there is an excess of information that we do not need. Let us proceed with filtering the variants based on some criteria like their likelihood, and only retain non-reference variants in the case of output from *bcftools mpileup*.

2.5.1 Begin by reviewing the bcftools call command.

```
bcftools call
bcftools call --ploidy ?
```

2.5.2 Use **`bcftools call`** to filter the variants in your VCF file based on one of two calling algorithms: **`-m`** is the default multi-allelic caller and **`-c`** uses the original consensus caller (which has some known limitations). The **`bcftools call`** command replaces the former **`bcftools view`** command.

Command	Meaning
`--ploidy 1`	Set the analysis to treat all samples as haploid (2 = diploid)
`-m`	Use the multi-allelic variant caller [default] (`-c` is the alternative)
`-v`	Output variant sites only (can also use `--variants-only`)
`HinfKW20_pe_align.vcf`	Input VCF file
`-Ov`	Set output type (`O`) to uncompressed VCF (`v`) file
`-o HinfKW20_pe_align_filtered.vcf`	Output file name

2.5.3 Review the filtered VCF output file and note how many variants are now present in the file using grep.

```
less HinfKW20_pe_align_filtered.vcf
grep -icv '^#' HinfKW20_pe_align_filtered.vcf
```

2.6.0 Use `samtools tview` to quickly view alignments

Now you are free to return to your favourite genome viewer like IGV to look more carefully at these filtered VCF files. Alternatively, you can visualize alignments in the regions where SNPs and INDELs were identified at the command line using **`samtools tview`**. This tool is similar to the Integrated Genome Viewer. Many of the

shortcuts within the viewer are similar to those used to move around a document with the **`less`** command. Not that **`samtools tview`** requires that your BAM file is sorted an indexed.

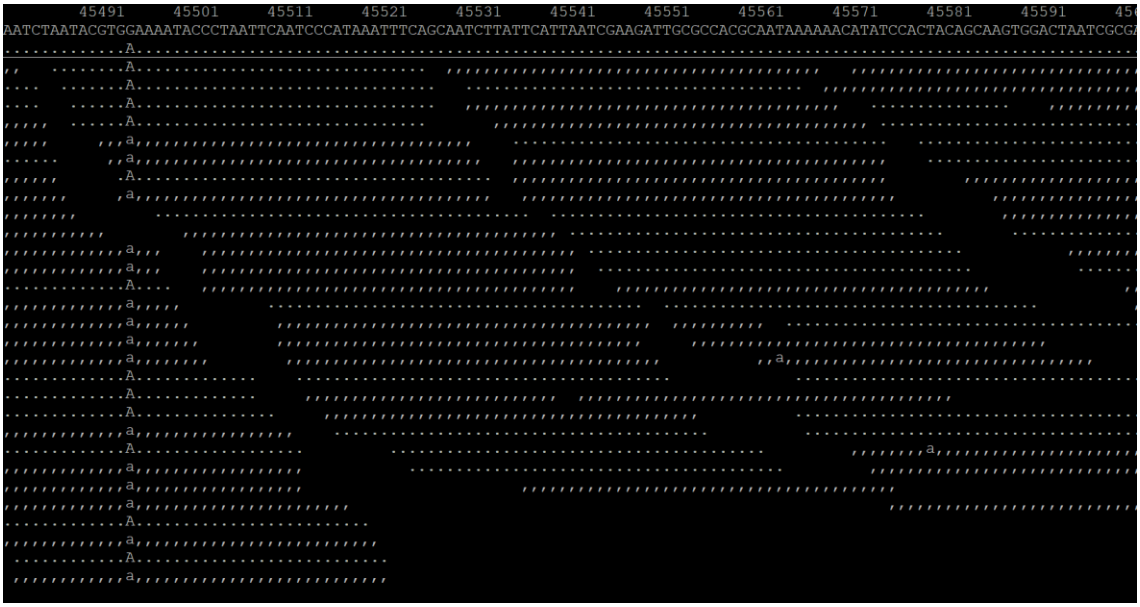
2.6.1 Review the **`samtools tview`** command call.

```
samtools tview
```

2.6.2 Use the **`samtools tview`** command on your sorted BAM file to examine the reference alignment and go to site L42023.1:45496 for an example of variant calls.

```
samtools tview HinfKW20_pe_align.sort.bam Data/HinfKW20_ref.fa
```

Command	Meaning
↑↓	move up and down the file to see all aligned reads
←→	move across the alignment one base at a time
Space Bar	move across the alignment in pages
b	colour bases by quality score.
/L42023.1:45496	go to site 45,496 on chromosome L42023.1 in the alignment.
q	quit
?	View a quick help window explaining additional keys



2.7.0 Move your files to a new folder

Now that we have completed our variant calling, you can move the files to another location for analysis at a later time.

2.7.1 Create a folder inside **`Module7`** called **`variantDetection`** and move all of the files generated so far in this module into that directory. This will isolate our variant detection output files from our upcoming RNA-Seq analysis output.

```
mkdir variantDetection
ls -la
mv HinfKW20_* variantDetection/
ls -la
ls /variantDetection
```

3.0.0 RNA-Seq reference alignment with HISAT2

We are now ready to revisit our RNA-Seq analysis from Module3. As was the case when we performed an RNA-Seq analysis on Galaxy, we're going to be comparing RNA-Seq datasets that were collected for *H. influenzae* under two different antibiotic treatments, **beclomethasone** and **prednisolone**. However, this time, we are going to work with a ***single replicate dataset*** throughout the analysis until we build a pipeline.

I downloaded all of these data from NCBI, but like last time, I've had to subset the fastq files so that the alignments can be run within a reasonable time frame. Unfortunately, even with the subset of reads, if we run all six analyses (2 treatments, 3 replicates), it could take a while. Therefore, while we learn the commands we will handle just one replicate sample from the **beclomethasone** treatment. We'll handle the remaining samples in a few subsections.

3.0.1 Begin by creating a secondary subdirectory in **`Module7`** called **`RNASeq`**, then copy the reference genome (Module7) and annotation .gtf (Module3) into the directory.

```
pwd
ls
mkdir RNASeq
cp Data/HinfKW20_ref.fa RNASeq/
cp ../Module3/RNA-SeqDownload/HinfKW20_genomic.gtf RNASeq/
```

3.0.2 If you haven't done so already, download the RNA-Seq files from the class server to the **~FGDS/Module7/Data** folder. There is an additional copy of **HinfKW20_genomic.gtf** as well.

```
cd ~/FGDS/Module7/Data
sftp fgds@142.150.215.32
      PW: fgds2021

ls Data/Module7
get Data/Module7/HinfKW20_[b,p]*.fastq

lcd ../RNASeq                              # Change the directory on your machine
get Data/Module7/RNASeqAnalysis.sh
get Data/Module7/sampleInfo.txt
get Data/Module7/DESeq.analysis.R
exit
ls -la
```

3.1.0 Check the quality of your paired-end RNAseq data

Similar to the beginning of class, we should take the time to check and see if splitting our RNA-Seq fastq files reveals any underlying issues with the sequencing data.

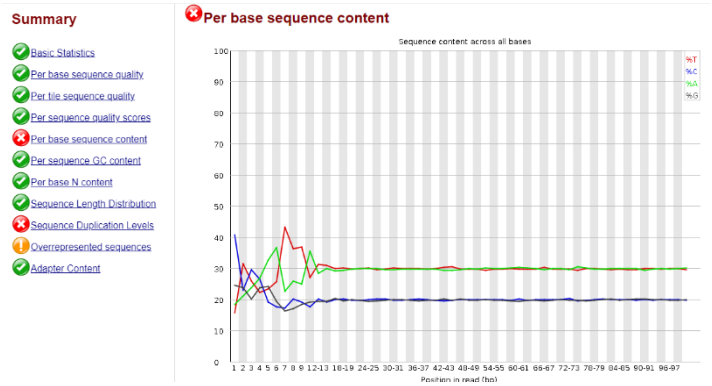
3.1.1 Generate a **`fastqc`** command from the command line to analyse all of our R1 RNA-Seq data. This should encompass the 2 files (forward and reverse) each of our R1 conditions for beclomethasone and prednisolone.

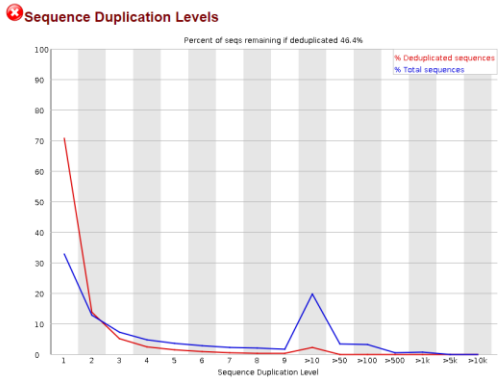
```
cd RNASeq
mkdir fastqc_RNAseq
fastqc --outdir fastqc_RNAseq ../Data/HinfKW20_[b,p]*_R1*
```

3.1.2 Review the HTML output from your **`fastqc`** output. Recall you can use the **`open`** command in macOS. For Windows users you can either copy to a temporary Windows folder or find the

`wsl$\Ubuntu-20.04\` shortcut in Windows Explorer and navigate to **home/<user_name>/FGDS/Module7/RNASeq/fastqc_RNAseq/** and open the file directly.

The output HTML files all show a similar series of problems: per base sequence content issues mainly in the first ~10 bases of data, sequence duplication failure/warnings, and overrepresented sequences. We already know from Module 4 that the latter two warnings are likely due to the nature of using expression data versus simple whole genome reads.





Overrepresented sequences

Sequence	Count	Percentage	Possible Source
CTTCGATCCTCAAACGGTGGCTTCCACACGATCTCGTTTTGGTTTGACTC	2071	0.2071	No Hit
CTGGATTCGACGGGATTAGCGAAGCCCAAGGTGCACGTCGAGGTGCGGTA	1806	0.1806	No Hit
CTCGTCTTTAATTTCACTTAAGCATGCGGACAGACACGCTAAACTTAAGC	1724	0.1724	No Hit
CGGGATTAGCGAAGCCCAAGGTGCACGTCGAGGTGCGGTAGGCCTCGTAA	1334	0.1334	No Hit
GTCCGAAATTACTCTACCTTCAGTACTACACGTTTAGTCTCGTCTTTAAT	1263	0.1263	No Hit
CGCGGAGTCAAACCAAAACGAGATCGTGTGGAAGCCACCGTTTGAGGATC	1066	0.10660000000000001	No Hit

The Sequence Content errors stem from the same underlying issue – biased content in the first 10 bases of our sequences. This is a known issue that results from a slight bias in the cut sites of the tagmentation enzyme that is used during library preparation. Removing these bases will get rid of the fastqc error but won't solve the underlying problem that bases near these regions are slightly more likely to be sequenced than other regions. Therefore, we are going to proceed ***without*** performing any head trimming.

3.2.0 Align your RNA-Seq reads with `hisat2`

Since we are skipping the trimming step, we will move straight to generating our **`HISAT2`** alignment. Recall from Module 3 that we are using **`HISAT2`** because it can handle both DNA and RNA-Seq alignment vs **`bowtie2`** which was designed with DNA sequencing in mind. It also produces fast alignment results with a moderate memory footprint – something we must think about when running from our own machines.

In order to complete our alignment we must work in 2 steps.

1. Generate an index of our genome. Similar to how we create an indexed reference genome when working with **`samtools`**, we must do the same but use **`HISAT2`**'s specific algorithms to index our *H. influenzae* genome.

2. Generate our RNA-Seq alignment with **`HISAT2`**.

3.2.1 Review the **`hisat2-build`** options before proceeding to index your reference genome.

```
hisat2-build -h
```

3.2.2 Generate an indexed reference genome using **`hisat2-build`**.

```
hisat2-build -p 4 HinfKW20_ref.fa HinfKW20_ht2
```

Command	Meaning
`-p 4`	Set the number of threads (4) to run the index process on. Larger genomes will take more time to index
`HinfKW20_ref.fa`	Reference genome to index by HISAT2
`HinfKW20_ht2`	The file name prefix used when generating the ht2 index files

```
ls -la                          # take a look at the files you've generated
```

```
(sambcftoolsENV) mokca@LAPTOP-7LF6OG94:~/FGDS/Module7/RNASeq$ ll
total 13128
drwxr-xr-x 1 mokca mokca    4096 Dec 15 13:31 ./
drwxr-xr-x 1 mokca mokca    4096 Dec 13 15:37 ../
-rw-r--r-- 1 mokca mokca 2219091 Dec 13 15:37 HinfKW20_genomic.gtf
-rw-r--r-- 1 mokca mokca 4805897 Dec 13 17:03 HinfKW20_ht2.1.ht2
-rw-r--r-- 1 mokca mokca  457512 Dec 13 17:03 HinfKW20_ht2.2.ht2
-rw-r--r-- 1 mokca mokca     989 Dec 13 17:03 HinfKW20_ht2.3.ht2
-rw-r--r-- 1 mokca mokca  457506 Dec 13 17:03 HinfKW20_ht2.4.ht2
-rw-r--r-- 1 mokca mokca  806381 Dec 13 17:03 HinfKW20_ht2.5.ht2
-rw-r--r-- 1 mokca mokca  465730 Dec 13 17:03 HinfKW20_ht2.6.ht2
-rw-r--r-- 1 mokca mokca      12 Dec 13 17:03 HinfKW20_ht2.7.ht2
-rw-r--r-- 1 mokca mokca       8 Dec 13 17:03 HinfKW20_ht2.8.ht2
-rw-r--r-- 1 mokca mokca 1853084 Dec 13 15:38 HinfKW20_ref.fa
```

3.2.3 Generate the **`hisat2`** alignment by providing the indexed reference genome and the paired-end sequencing files. Recall that in our Galaxy module, we were able to run **`HISAT2`** using multiple files. Each file we selected generated a ***separate*** alignment run/call. Here we must run each file in a separate call. If we combine multiple files into a single command, it will produce a single alignment containing all of the sequencing reads.

Command	Meaning
`-x HinfKW20_ht2`	the path and prefix for the indexed reference genome you've created.
`-1 HinfKW20_beclo_R1_for.fastq`	a comma-separated list of the mate1 (forward) paired end sequencing files
`-2 HinfKW20_beclo_R1_rev.fastq`	a comma-separated list of the mate2 (forward) paired end sequencing files
`-S HinfKW20_beclo_R1_align.sam`	Send the output to a .sam file with the provided name

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module7/RNASeq$ hisat2 -x HinfKW20_ht2 -1 HinfKW20_beclo_R1_for.fastq
-2 HinfKW20_beclo_R1_rev.fastq -S HinfKW20_beclo_R1_align.sam
1000000 reads; of these:
  1000000 (100.00%) were paired; of these:
    29762 (2.98%) aligned concordantly 0 times
    954717 (95.47%) aligned concordantly exactly 1 time
    15521 (1.55%) aligned concordantly >1 times
    ----
    29762 pairs aligned concordantly 0 times; of these:
      5099 (17.13%) aligned discordantly 1 time
    ----
    24663 pairs aligned 0 times concordantly or discordantly; of these:
      49326 mates make up the pairs; of these:
        31697 (64.26%) aligned 0 times
        17031 (34.53%) aligned exactly 1 time
        598 (1.21%) aligned >1 times
98.42% overall alignment rate
(sambcftoolsENV) mokca@LAPTOP-7LF6OG94:~/FGDS/Module7/RNASeq$
```

```
ls -la                                          # Review the SAM file you made
less HinfKW20_beclo_R1_align.sam
```

3.2.4 Recall that unlike our Galaxy pipeline, which generated a series of BAM files for us, our command-line calls have created SAM files instead. We will have to convert and sort the resulting SAM file again just as in section **2.1.0** and **2.2.0**. This time we'll use the command-line pipe (|) to accomplish our goals by converting and then sorting in a single command.

```
# sort and convert beclomethasone replicate 1
samtools view -b -T HinfKW20_ref.fa HinfKW20_beclo_R1_align.sam | samtools sort >
      HinfKW20_beclo_R1_align.sort.bam

ls -la                                          # Take a look at your files
```

3.2.5 Note how large your SAM files are in comparison to the BAM file? Nearly 700 Mb vs < 150 Mb each! Remove the original SAM file to save on disk space.

```
rm *.sam
```

4.0.0 Summarize RNA-Seq alignments with featureCounts

To review, we've completed an alignment of our RNA-Seq reads and now have a BAM file. We can turn to **featureCounts** to compare with a **GTF** file to catalog the number of reads matched to specific gene features. Afterwards, we can import the **featureCounts** file into **R** to use with the **DESeq2** package.

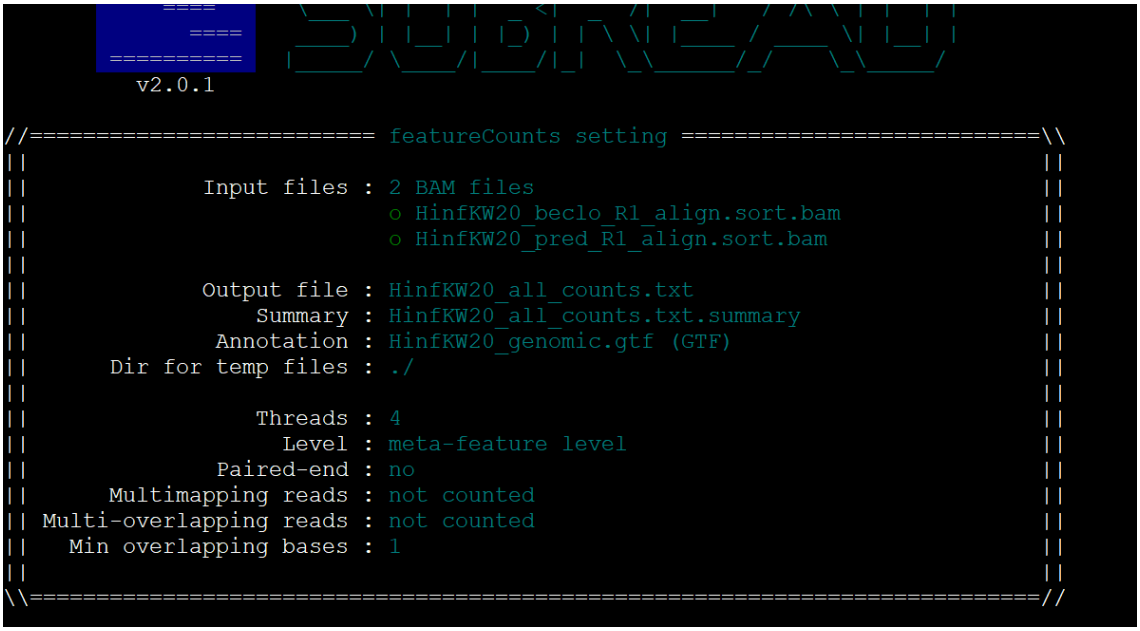
4.1.0 Generate read analysis with featureCounts

For our purposes, it makes sense to provide all of our BAM files in a single command that will produce one large table for comparison downstream. For now, however, we only have a single BAM file to work with.

4.1.1 Use the **featuresCounts** command with the sorted BAM file generated from your **HISAT2** alignment. Use the "**gene**" feature type and specify "**gene_id**" as the attribute within our GTF annotation file.

Command	Meaning
-B	Only count read pairs that have both ends aligned
-C	Do not count read pairs that have their two ends mapping to different chromosomes or mapping to opposite strands
-T	Number of threads to process with (default is 1)
-t gene	Comma-separated list of feature types in gtf annotation (default is exon)
-g gene_id	Specify attribute type in GTF annotation (default is gene_id)
-a HinfKW20_genomic.gtf	Path to the .gtf annotation file
-o HinfKW20_all_counts.txt	Path to the output file
HinfKW20_beclo_R1_align.sort.bam	A space-separated list of SAM or BAM files used in the analysis. File names will become the column names for each count column

The **featureCounts** command should produce standard output similar to the following images. Note that this information identifies the output and summary file names.



```
//============================== Running ===============================\\
||                                                                      ||
|| Load annotation file HinfKW20_genomic.gtf ...                        ||
||    Features : 1775                                                   ||
||    Meta-features : 1775                                              ||
||    Chromosomes/contigs : 1                                           ||
||                                                                      ||
|| Process BAM file HinfKW20_beclo_R1_align.sort.bam...                 ||
||    WARNING: Paired-end reads were found.                             ||
||    Total alignments : 2039738                                        ||
||    Successfully assigned alignments : 1677122 (82.2%)                ||
||    Running time : 0.02 minutes                                       ||
||                                                                      ||
|| Process BAM file HinfKW20_pred_R1_align.sort.bam...                  ||
||    WARNING: Paired-end reads were found.                             ||
||    Total alignments : 2043003                                        ||
||    Successfully assigned alignments : 1606341 (78.6%)                ||
||    Running time : 0.04 minutes                                       ||
||                                                                      ||
|| Write the final count table.                                         ||
|| Write the read assignment summary.                                   ||
||                                                                      ||
```

4.1.2 Review the resulting output file from featureCounts.

```
less HinfKW20_all_counts.txt
less HinfKW20_all_counts.txt.summary
```

Note in the counts.txt file which column holds count data vs the other metadata that has been included.

```
# Program:featureCounts v2.0.1; Command:"featureCounts" "-BC" "-T" "4" "-t" "gene" "-g" "gene_id"
 "-a" "HinfKW20_genomic.gtf" "-o" "HinfKW20_all_counts.txt" "HinfKW20_beclo_R1_align.sort.bam"
Geneid  Chr     Start   End     Strand  Length  HinfKW20_beclo_R1_align.sort.bam
HI_0001 L42023.1        2       1021    +       1020    4472
HI_0002 L42023.1        1190    3013    +       1824    460
HI_0003 L42023.1        3050    3838    -       789     311
HI_0004 L42023.1        3854    4318    -       465     198
HI_0005 L42023.1        4579    5391    -       813     177
HI_0006 L42023.1        5656    8748    +       3093    1741
HI_0007 L42023.1        8750    9688    +       939     373
HI_0008 L42023.1        9681    10397   +       717     313
HI_0009 L42023.1        10467   11375   +       909     152
```

4.2.0 Simplify count data for analysis by `DESeq2`

As we prepare to move on to analysis with **`DESeq2`** we only need to retain the basic count information mapping **gene_id** to **count** information. We also need to remove the first line of our counts file as it is a comment header unsuitable for import into **`DESeq2`**.

4.2.1 Use **`awk`** to remove the first line and **`cut`** to produce a tabular data file with only **Geneid** and **count** information. Review your final output.

```
awk 'NR>1' HinfKW20_all_counts.txt |
less HinfKW20_simple_counts.txt
```

```
Geneid  HinfKW20_beclo_R1_align.sort.bam
HI_0001 4472
HI_0002 460
HI_0003 311
HI_0004 198
HI_0005 177
HI_0006 1741
HI_0007 373
HI_0008 313
```

5.0.0 Bash to the future (part 2): build a script for RNASeq analysis

Before proceeding to **DESeq2**, we must consider that it cannot generate a proper analysis without using replicate data. Thus far, we've only been working with a single set of RNA-Seq data for each condition. In total we have ***2 conditions each with 3 replicates and each replicate has 2 files*** (forward and reverse fastq). In total there are 12 fastq files to work with. Now that we've worked out the basic conditions for our RNA-Seq analysis we should ***automate*** the remaining files in some way. Let's break down the pattern of calls we make:

1. A single call to index our reference genome for **HISAT2**

```
hisat2-build -p 4 HinfKW20_ref.fa HinfKW20_ht2
```

2. A call to **HISAT2** to align our sequence data. We must do this 6 times!

```
hisat2 -x HinfKW20_ht2 -1 file_for.fastq -2 file_rev.fastq -S file_align.sam
```

3. A call to **samtools** to make BAM files. We must do this 6 times!

```
samtools view -b -T HinfKW20_ref.fa file_align.sam | samtools sort >
      file_align.sort.bam

rm *.sam                # Remove all of the SAM files
```

4. A single call to **featureCounts** where we can combine all of our bam files

```
featureCounts -BC -T 4 -t gene -g gene_id -a HinfKW20_genomic.gtf -o
      HinfKW20_all_counts.txt file1_align.sort.bam file2_align.sort.bam ...
      file6_align.sort.bam
```

We can replace some of these steps with our good friend **awk** if we have the right information to build and make system command calls.

5.0.1 Look at the sample information we downloaded today in **sampleInfo.txt**. This file will also be used later as metadata for our **DESeq2** analysis.

```
less sampleInfo.txt
```

```
sample_ID       pe_file_prefix  condition
HinfKW20_beclo_R1_align.sam     HinfKW20_beclo_R1       beclomethasone
HinfKW20_beclo_R2_align.sam     HinfKW20_beclo_R2       beclomethasone
HinfKW20_beclo_R3_align.sam     HinfKW20_beclo_R3       beclomethasone
HinfKW20_pred_R1_align.sam      HinfKW20_pred_R1        prednisolone
HinfKW20_pred_R2_align.sam      HinfKW20_pred_R2        prednisolone
HinfKW20_pred_R3_align.sam      HinfKW20_pred_R3        prednisolone
sampleInfo.txt (END)
```

We can see a list of our samples in 3 tab-separated columns: **sample_ID**, **pe_file_prefix**, and **condition**. We'll also be using this information in the next steps when working with **DESeq2**.

5.0.2 Use your favourite text editor to open and edit the **RNASeqAnalysis.sh** bash script.

```
vi RNASeqAnalysis.sh
      :set number
      i
```

5.1.0 Use switches/flags in your bash script

Recall from our bash script last week, we provided a series of file inputs/arguments to our script **assemblyAndAnnotation.sh** based on their position in the command. This required the order of our inputs to be invariant.

We can, instead, generate a set of switches/arguments using the **`getopts`** construct. This construct looks for specific switches (ie -f, -p, -g) when you call on your bash script. Unlike our script from last week, this method will allow us to provide arbitrary positions for our input files rather than a set order. You can learn to create more advanced versions of these as well.

The basic version we will use takes advantage of the **`getopts`** construct which takes the form of

```
while getopts "<switch><switch><flag>:<flag>:" flag_var
do
  case statement
done
```

Recall that **switches** turn options on/off in your script while **flags** are followed by *some kind of argument* such as a file name or option. In our **`getopts`** invocation, the quoted list of letters denotes switches and flags as single letters separated without spaced but flags must be followed by a "**`:`**". We follow this with a variable that holds the current flag/switch being examined.

5.1.1 Complete the **`getopts`** statement by providing 3 flags to denote our sample file (f), data path (p), and annotation file (g).

```
while getopts "f:p:g:" flag                    # line 15
```

5.2.0 Use `case` statements to look search for possible values

The case statement is a simple way to look for specific variable values that will trigger specific actions within your script such as assigning values. In the context of our **`getopts`** while loop, it will look at the value of switches and flags provided by the user and assign appropriate values to variables within our script.

Our case statement is meant to check the flags provided for a specific flag and assign it to a variable where **`$OPTARG`** is a variable in **`getopts`** that holds the current flag input being examined. The case statement breaks down into the following pattern:

```
case EXPRESSION in
  PATTERN_1)      # List one or more patterns, separating with "|"
    STATEMENTS    # Do things here like assign variables
    ;;            # this signals the end of a single case
  PATTERN_2)
    STATEMENTS
    ;;
  *)
    STATEMENTS
    ;;
esac
```

5.2.1 Complete the **`case`** statement for -f which allows you to set the sample information file for the script.

```
f) SAMPLE_INFO=$OPTARG                         # line 20
  echo "Sample file is: $SAMPLE_INFO" >&2      # line 21 Output to stderr
  ;;                                           # line 22
```

Note that we are outputting information to standard error (stderr) by directing output with **>** to **&2** in line 21 but you could experiment with **&1** (stdout) as well.

5.3.0 Save command-line output and arithmetic calculations to a variable

At the end of our script we will want to know how many data sets we are working with. We could provide this information in the initial arguments of our bashscript but we are already working with a file (sampleInfo.txt) that will have that information held within. We just have to finesse it out with a simple awk command:

```
awk 'END {print NR}'    # This will output the TOTAL lines in our sample file
```

The question is, how do we ***save*** this information into a variable? You can use command substitution syntax that looks like this:

```
var=$(command_name arg1 arg2)
```

Suppose we wanted to perform arithmetic on a variable or set of variables? Perhaps you have an integer variable you want to add or subtract from. A similar syntax is used to help bash script know it is doing math instead of concatenating strings:

```
var=$(($value1<math_operator>$value2))
#or
var=$(($num-5))
```

5.3.1 Complete the calculation of **SAMPLE_NUM** on line 39 by combining command substitution and arithmetic syntax. Calculate the number of lines in our sample file and subtract 1.

```
SAMPLE_NUM=$(($(awk 'END {print NR}' "$SAMPLE_INFO")-1))                # Line 39
```

5.4.0 Generate an `awk` command that performs 3 tasks

Back to the structure of what's happening in our overall pipeline, we are ready now to use the sample information as a substitute to command-line calls. We can, in fact, generate 3 different calls for each replicate: a HISAT2 alignment; a SAM to BAM conversion/sort; and a removal of the large SAM file to save on space. Doing this "on the fly" will mean we aren't keeping a large group of SAM files as well as BAM files.

5.4.1 Complete the awk command by updating the **dataPath** variable information. This will get its value from the bash script before passing it along to awk. Note how we use the condition "**NR>1**" to skip the first line of the input file.

```
awk -v dataPath=$DATA_PATH 'NR>1 {align_cmd="hisat2 -x HinfKW20_ht2 -1
      "dataPath"/"$2"_for.fastq -2 "dataPath"/"$2"_rev.fastq -S "$1; \  # Line 56
```

5.5.0 Create a comma-separated list of numbers as a variable

Recall that after we create our **featureCounts** output we really just want to keep column 1 and the remaining columns with count information. The read-count columns will always begin at column 7 and continue through until the last column. In other words, the final column will be equal to 6 + the number of replicates!

The **cut** command requires a comma-separated list of numbers representing columns. How do we generate this in a bash script? In the command line, we can call on the command **seq** to generate a delimited sequence of values with the following syntax:

```
seq -s<delimiter> <start_value> <end_value>
```

5.5.1 Use command-line and arithmetic substitution to create a comma-separated list of values saved to the variable **`COL_VALS`** for use in the **`cut`** command.

```
                                                                    # Line 70
```

5.5.2 Save your bash script and exit your editor. Here's how to do it in **`vi`**:

```
[esc]
:wq
```

5.6.0 Set permissions and run your bash script

Now that we've completed our edits, it's time to run the bash script itself and see if it works.

5.6.1 Set the permissions on your bash script so that it can be executed by you.

```
chmod 744 RNASeqAnalysis.sh
```

5.6.2 Run your bash script by including the relevant arguments. Each replicate will take about 1 minute to complete so sit tight.

```
bash RNASeqAnalysis.sh -f sampleInfo.txt -p ../Data -g HinfKW20_genomic.gtf
```

5.6.3 Once your script has completed running (be patient!) you can look at your finalized, simplified featureCounts dataset.

```
less HinfKW20_simple_counts.txt
```

```
Geneid  HinfKW20_beclo_R1_align.sort.bam        HinfKW20_beclo_R2_align.sort.bam        H
infKW20_beclo_R3_align.sort.bam        HinfKW20_pred_R1_align.sort.bam HinfKW20_pred_R2_a
lign.sort.bam HinfKW20_pred_R3_align.sort.bam
HI_0001 4472    4229    4339    6525    7137    5472
HI_0002 460     639     689     604     714     823
HI_0003 311     311     354     199     173     194
HI_0004 198     206     247     149     147     151
HI_0005 177     224     178     148     128     137
HI_0006 1741    1728    1618    1710    1814    1319
HI_0007 373     391     343     237     321     362
HI_0008 313     315     266     114     214     267
HI_0009 152     131     157     77      116     74
```

6.0.0 Using R and DESeq2 for expression analysis

Now that we've prepared our count data in the format we require, it's time to pass it along to the **DESeq2** package. On Galaxy in Module 3, we ran **DESeq2** through an interactive GUI where we could pick options and supply data. All of this was, in fact, a complex "veneer" used to cover up the dark and codified underside that would supply that information to some series of **R scripts** that would run in **R**. While we won't replicate all of those functions today but we will generate and save the finalized analysis data.

Before we begin, however, we should compare and contrast **R** with the bash command line.

Bash Shell	**R Shell**
Runs text-based commands	Runs text-based commands
Packages that run functions	Libraries that have packages that run functions
Packages are loaded by default through `$PATH`	Packages are directly loaded by user via "`library()`"
Commands can be combined into bash scripts (`.sh`)	Commands can be combined into R scripts (`.R`)
Command arguments proceed after a command call in a space-separate list	Functions arguments proceed after function call in a comma-separated list within parenthesis.

As you can see while it may seem intimidating, many of the concepts from the command line carry over to a programming language like **R**. It really is just a matter of learning the syntax and tools while many of the ideas remain the same.

6.1.0 Activate `R` and load the `DESeq2` library

Before we begin our analysis we must set up our environment so we can work with it properly. This includes loading packages that we want to work with like **DESeq2**. While it has been "installed" in Anaconda, it is not active in the shell and must be loaded and run by **R**.

6.1.1 Activate your Anaconda **r_411ENV** environment and instantiate the **R** shell

```
conda activate --stack r_411ENV
R
```

6.1.2 Inside **R**, load the **DESeq2** library. We do this with the **library()** function.

```
library(DESeq2)
```

```
> library(DESeq2)
Loading required package: S4Vectors
Loading required package: stats4
Loading required package: BiocGenerics

Attaching package: 'BiocGenerics'

The following objects are masked from 'package:stats':

    IQR, mad, sd, var, xtabs

The following objects are masked from 'package:base':

    anyDuplicated, append, as.data.frame, basename, cbind, colnames,
    dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
    grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
    order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
    rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
    union, unique, unsplit, which.max, which.min
```

6.2.0 Load your data files with `read.csv()`

Now that we have our environment prepared, we need to import our files into the shell as data frames (a tabular data object like an excel sheet). Once imported into the R shell, they will be accessible as variables that we can manipulate or pass as arguments to other functions.

6.2.1 Check your working directory and move to the correct data directory if you are not where you want to be.

```
getwd()     # Once inside this will tell you your working directory

setwd("~/FGDS/Module7/RNASeq")      # equivalent to "cd" in the shell
```

6.2.2 After you are sure you are in the correct directory, load your count data from **HinfKW20_simple_counts.txt** and save it as the variable **`countData`**.

```
countData <- read.csv("HinfKW20_simple_counts.txt", header=TRUE, sep="\t")
```

Command	Meaning
`countData`	The variable we want to import our data into
`read.csv`	A function to read different file types
`"HinfKW20_simple_counts.txt"`	The path to our data
`header=TRUE`	Flag to tell read.csv that our first line is header information
`sep="\t"`	Denotes that our data columns are tab-separated

```
head(countData)                     # Look at the first 6 rows of your dataframe
```

```
> head(countData)
   Geneid HinfKW20_beclo_R1_align.sort.bam HinfKW20_beclo_R2_align.sort.bam
1 HI_0001                            4472                            4229
2 HI_0002                             460                             639
3 HI_0003                             311                             311
4 HI_0004                             198                             206
5 HI_0005                             177                             224
6 HI_0006                            1741                            1728
  HinfKW20_beclo_R3_align.sort.bam HinfKW20_pred_R1_align.sort.bam
1                             4339                            6525
2                              689                             604
3                              354                             199
4                              247                             149
5                              178                             148
6                             1618                            1710
```

6.2.3 Import your metadata from **sampleInfo.txt** and save it as the variable **`metaData`** before viewing it.

```
metaData <- read.csv("sampleInfo.txt",                    )
head(metaData)
```

```
> head(metaData)
                    sample_ID    pe_file_prefix      condition
1 HinfKW20_beclo_R1_align.sam HinfKW20_beclo_R1 beclomethasone
2 HinfKW20_beclo_R2_align.sam HinfKW20_beclo_R2 beclomethasone
3 HinfKW20_beclo_R3_align.sam HinfKW20_beclo_R3 beclomethasone
4  HinfKW20_pred_R1_align.sam  HinfKW20_pred_R1   prednisolone
5  HinfKW20_pred_R2_align.sam  HinfKW20_pred_R2   prednisolone
6  HinfKW20_pred_R3_align.sam  HinfKW20_pred_R3   prednisolone
>
```

6.3.0 Generate a DESeq2 analysis

In order to generate our **DESeq2** analysis, we need to complete one more preparation step by creating an input-compatible object for **DESeq2** to work with. We do this with a function provided by the **DESeq2** package before using the resulting output to generate an analysis.

6.3.1 Generate a **DESeq2**-compatible object with the **DESeqDataSetFromMatrix()** function. Save the object as the variable **ddsMatrix**.

```
dds <- DESeqDataSetFromMatrix(countData = countData, colData = metaData, design
      =~condition, tidy=TRUE)
```

Command	Meaning
`Dds`	The variable we want to hold our DESeqDataSet object
`DESeqDataSetFromMatrix`	Combines read count and metadata into a new matrix object
`countData = countData`	The simplified featureCounts data to use for the algorithm
`colData = metadata`	Use metadata to match with the countData columns. Correct order here matters!
`design =~condition`	How are the study groups designed? There may be multiple attributes you wish to group them by.
`tidy=TRUE`	Denotes that our countData has a column of row names (geneIDs)

6.3.2 Generate a DESeq2 analysis using the **ddsMatrix** object.

```
ddsAnalysis <- DESeq(dds)
```

```
> dds <- DESeq(ddsMatrix)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
> _
```

6.3.3 Extract your **DESeq2** results and sort them by ascending order of **padj** value. The resulting **dds** object you've created holds a lot of information which you need to extract using some functions it provides for access.

```
# extract the results from dds and save them to a new data frame
ddsResult <- results(ddsAnalysis)

head(ddsResult)

# sort the data frame
ddsResult <-

head(ddsResult)
```

6.3.4 Export your data to a tab-delimited file using the **write.table()** function. You could choose to do further analysis on this data like making visualizations in **R** or **Python** or you can work with the data further in another program.

```
write.table(ddsResult, file="HinfKW20_DESeq2_analysis.tsv", col.names=NA,
      row.names=TRUE, sep="\t")
quit()
```

6.4.0 Run your `DESEq2` analysis from an R script

You've now completed a series of steps used to produce some basic data from the **`DESeq2`** package. As mentioned at the start of this section, we can actually encapsulate these steps into a single file much like we would a bash script. This R script can then be called from the command prompt.

6.4.1 Review the R script you downloaded earlier entitled **DESeq.analysis.R**

```
less DESeq.analysis.R
```

6.4.2 Run the R script from the command line using the built-in **`Rscript`** command within your **`r_411ENV`**.

```
Rscript DESeq.analysis.R
```

Want to learn more about R? There is a world of resources out there for you to explore. We only had time to touch on using the DESeq2 package (more tutorials here) but there are other analysis packages you can use as well as a suite of tools that can help you format and visualize your data.

Other helpful resources and classes include datacamp.com and, of course, there are some introductory data science courses offered through Cell and Systems Biology as well.

7.0.0 Class and course summary

That concludes our final lecture bringing you up to speed on running our own reference alignment, variant calling, and RNAseq analysis pipelines through the command line interface. Your journey through the course is complete although you still have a term project and a whole world to keep exploring! Altogether we've explored the following in this module:

- Generating a reference alignment from raw fastq data with bowtie2
- Identifying variants from your alignments and filtering them into a VCF file.
- Using HISAT2 to align RNA-Seq read data and featureCounts to generate read counts from said alignments.
- Creating a custom bash script to cycle through multiple replicate data files and generate a final featureCounts dataset.
- Using R to analyse read count data with the DESeq2 package and creating an R-script that can be run from the command line.

7.1.0 Post-course survey

We have created a post-course survey you can fill out anonymously. You can use this survey as an opportunity to tell us about your experience and help shape the future offerings of this series. Please take 5-10 minutes to fill out the survey. We really appreciate your feedback and future students will too!

https://forms.gle/jSsJ7eUmxbuXuTvu8

7.2.0 Post-lecture assessment (10% of final grade)

Soon after this lecture, a homework assignment will be made available on Quercus in the assignment section. It will build on the ideas and/or data generated within this lecture. Each homework assignment will be worth 10% of your final mark. If you have assignment-related questions, please try the following steps in the order presented:

- Check the internet for a solution – read forums and learn to navigate for answers.
- Generate a discussion on Quercus outlining what you've tried so far and see if other students can contribute to a solution.
- Contact course teaching assistants or the instructor.

7.3.0 Bonus assessment (5% of final grade)

An additional bonus assignment will soon be made available on Quercus in the assignment section. This assignment will test ideas from each module across the entire course. There will be 7 sections each equivalent to a 1% bonus on your final grade, with a maximum of 5%. Additional instructions will be found within the assignment.

7.4.0 Term project (30% of final grade)

Term projects will be due 2 weeks after final lecture on December 31st 2021. Consider the format of the course material and as a model for how to present your own term project. **Please read the supplied term project rubric carefully.**

- Aim to produce a clear introduction with an overview of your pipeline.
- Use sections to explain different steps within your pipeline while giving examples of how to use commands and what the results look like.
- Use your own, simulated, or data curated from other manuscripts (cite these properly!)

7.3.1 Frequently asked questions:

- My introduction is more than 750 words. Is that okay?
    - I won't penalize you for being verbose, but I prefer that you are clear and concise. I'm not looking for a novel nor do I want to read about your cherished childhood memories that led you to this point in your data science journey. Give me what I need to know about your project so that I can understand why it's important, why you're doing it and why you're taking the direction you are with your analyses. Feel free to make your childhood story an appendix if you are *sure* it must be included somewhere.

- Can I use programs or packages from outside the course?
    - Absolutely. Just be sure to explain your choices and how to use them (either via command-line or Galaxy).

- Do I have to use every concept you taught in the course?
    - Absolutely not. Set out with a plan in mind, explain how you're going to accomplish your goals and use the commands you need to do that. The more you show me (to a reasonable extent), the more I can assess your skills but if you don't use some aspects like RegEx or sftp, I certainly won't penalize you. What's important is that your pipeline works, it takes advantage of what we've learned (like bash scripts if applicable), and it is well-commented or well-documented.

- My pipeline isn't perfect, can I get an extension until it works the way I want it to?
    - The simple answer is No. The complicated answer is maybe but on a case-by-case basis. Your pipeline will hardly ever be perfect the first time you produce it. It's a work in progress and I want to see your progress. If you are having trouble with your commands, you still have two weeks to contact me or the TAs or search the internet. Again, it just needs to get the job done as best as you can. When I do mark your assignments, I'll suggest how you can fix or streamline your pipeline to meet your goals. If it's a bottleneck to your next step, you can always produce a "final" formatted file as you'd like to see it and use that on later steps if those work as planned.

- My data file is a GIGANTIC sequencing file for analysis can I submit that? If I run a subset of data, my results will look terrible!
    - When you submit your projects, you will have run the pipeline and you can use your original data file(s) when you generate the PDF with all of your output. The data file you actually submit can be a smaller subset that I use in case I need to run parts of your code. Of course make a note of this somewhere in your intro/background too so that I know what I'm looking at. Alternatively take a screenshot of your data so I can get an idea of what it looks like going in and/or coming out of a command or series of commands.
    - You can also submit smaller files like your bash scripts etc., if you want me to look at those and help improve them.