University of Pennsylvania
BIOL4536 Fall 2023

HW#7

(ChIP-Seq)

Assigned October 18th
Due October 26th, 11:59pm

This is a big data (high-througput sequencing) analysis, which is bread-and-butter bioinformatics. But big data is cumbersome, so you're going to want to start early and budget a fair amount of time for this. Don't be intimidated by the length of this document, it's long because it is trying to provide nitty gritty detail to make your job as easy as possible. If anything is confusing, feel free to reach out to any of us for clarification.

We're going to run a ChIP-Seq analysis pipeline. This is big-data and will give you the experience of running a real big data analysis pipeline. The data are transcription factor binding assays of the transcription factor `FOXA1` in the human breast cancer cell line called MCF-7.

MAKE DIRS

Make a directory in your space called `HW7`. Inside `HW7` make the following 10 subdirectories:

```
genome
index
qc
trimmed
alignment
merged
filtered
graphs
peaks
differenial
```

CREATE GENOME ALIGNMENT INDEX

We're going to use a DNA aligner called `bowtie`. We need to first build a `bowtie` alignment index for the human genome.

The genome is kept here:
`/project/genome/Homo_sapiens.GRCh38.dna.primary_assembly.fa`

It is in shared space, you do not need to copy it to your space, in what follows you will reference it in place. You will need the `HW7/genome` local directory for other things.

Next we're going to build the alignment index.

IMPORTANT NOTE: the command has to be on one line, if you copy/paste from this PDF then you should be able to back arrow to the linebreaks and delete them (making sure to maintain a space between things that were on different lines). The following three graphics illustrate this for the first command:

After pasting it starts on three lines:

```
ggrant@workstation:~/HW7$ batch submit-job --job-name INDEX --vcpu 16 --memory 110000 --command "bowtie2-build
--threads 16 --seed 42 /project/genome/Homo_sapiens.GRCh38.dna.primary_assembly.fa
index/Homo_sapiens.GRCh38.dna.primary_assembly"
```

Back up to beginning of 3rd line and remove newline but keep a space. It's now on two lines. The second line wraps, but it has no linebreak in it.

```
ggrant@workstation:~/HW7$ batch submit-job --job-name INDEX --vcpu 16 --memory 110000 --command "bowtie2-build
--threads 16 --seed 42 /project/genome/Homo_sapiens.GRCh38.dna.primary_assembly.fa index/Homo_sapiens.GRCh38.dna.primary_
assembly"
```

Back up to beginning of 2nd line and remove newline but keep a space. It's now on one line and you can hit "Enter"

```
ggrant@workstation:~/HW7$ batch submit-job --job-name INDEX --vcpu 16 --memory 110000 --command "bowtie2-build --threads
16 --seed 42 /project/genome/Homo_sapiens.GRCh38.dna.primary_assembly.fa index/Homo_sapiens.GRCh38.dna.primary_assembly"
```

This is going to be a recurring problem throughout, you will have to remove various newlines after copy/pasting to make the command work. Don't forget to maintain a space wherever there was a newline.

Here is the command, execute it from the `HW7` directory.

```
batch submit-job --job-name INDEX --vcpu 16 --memory 110000 --command "bowtie2-build
--threads 16 --seed 42 /project/genome/Homo_sapiens.GRCh38.dna.primary_assembly.fa
index/Homo_sapiens.GRCh38.dna.primary_assembly"
```

This executes the index building command on a (cluster) node. This will probably take at least 30 minutes to run. Use the "`batch list-jobs`" command to monitor its progress. It's done when the status says "SUCCEEDED" (or "FAILED" in which case you'll have to debug, the command '`batch describe-jobs --jobs <jobID>`' will show you the log files of the job which should have useful error messages). You might want to close the lefthand panel (`CTRL-B`) to keep the output from wraping.

Since it's a cluster job, it's running on a separate machine, so you can log off and come back later, it will still finish. Soon we'll submit multiple jobs at the same time. The reason we're using the cluster for just one job this time is because it's a big job, so we can't have everybody running it at the same time on the one shared (master) node.

While it's running a bunch of temp files will occupy the `index` directory. They will be automatically deleted once it's done running. Once indexing is done, you should see the following files in the `index` directory (note, `lsr` is my alias for `ls -ltr`):

```
ggrant@workstation:~/hw7$ lsr index/
total 4087580
-rw-r--r-- 1 ggrant ggrant 737156939 Oct  8 18:55 Homo_sapiens.GRCh38.dna.primary_assembly.4.bt2
-rw-r--r-- 1 ggrant ggrant     11384 Oct  8 18:55 Homo_sapiens.GRCh38.dna.primary_assembly.3.bt2
-rw-r--r-- 1 ggrant ggrant 987098397 Oct  8 19:10 Homo_sapiens.GRCh38.dna.primary_assembly.1.bt2
-rw-r--r-- 1 ggrant ggrant 737156944 Oct  8 19:10 Homo_sapiens.GRCh38.dna.primary_assembly.2.bt2
-rw-r--r-- 1 ggrant ggrant 987098397 Oct  8 19:25 Homo_sapiens.GRCh38.dna.primary_assembly.rev.1.bt2
-rw-r--r-- 1 ggrant ggrant 737156944 Oct  8 19:25 Homo_sapiens.GRCh38.dna.primary_assembly.rev.2.bt2
ggrant@workstation:~/hw7$
```

THE DATA

The ChIP-Seq raw sequencing data are also in shared space:
`/project/chip-seq/fastq`
There are eight files:

```
SRR1635435.fastq.gz
SRR1635436.fastq.gz
SRR1635437.fastq.gz
SRR1635438.fastq.gz
SRR1635459.fastq.gz
SRR1635460.fastq.gz
SRR1635461.fastq.gz
SRR1635462.fastq.gz
```

The fastq files are the raw data. This is single-end data, one file per sample. They are `gzipped` so you can't inspect them with `head, tail, more, less, cat`. But try this (from the `/project/chip-seq/fastq` directory):

```
zcat SRR1635435.fastq.gz | more
```

You see what `zcat` does.

As discussed in the class slides, there are four lines per read, the first line is the read name and the second is the sequence itself.

(1) What is the read length in sample `SRR1635435`?

There are eight fastq files. These represent two replicates each of two experimental conditions. That's four. Plus, there are four so-called "input controls" which are the same samples but they have not undergone chromatin-immunoprecipitation. We'll need them as controls, as you will see later.

Next, we need to define a variable. Execute the following:

```
FILES="SRR1635435 SRR1635436 SRR1635437 SRR1635438 SRR1635459 SRR1635460 SRR1635461 SRR1635462"
```

Execute the following command to make sure the variable is defined.

```
echo $FILES
```

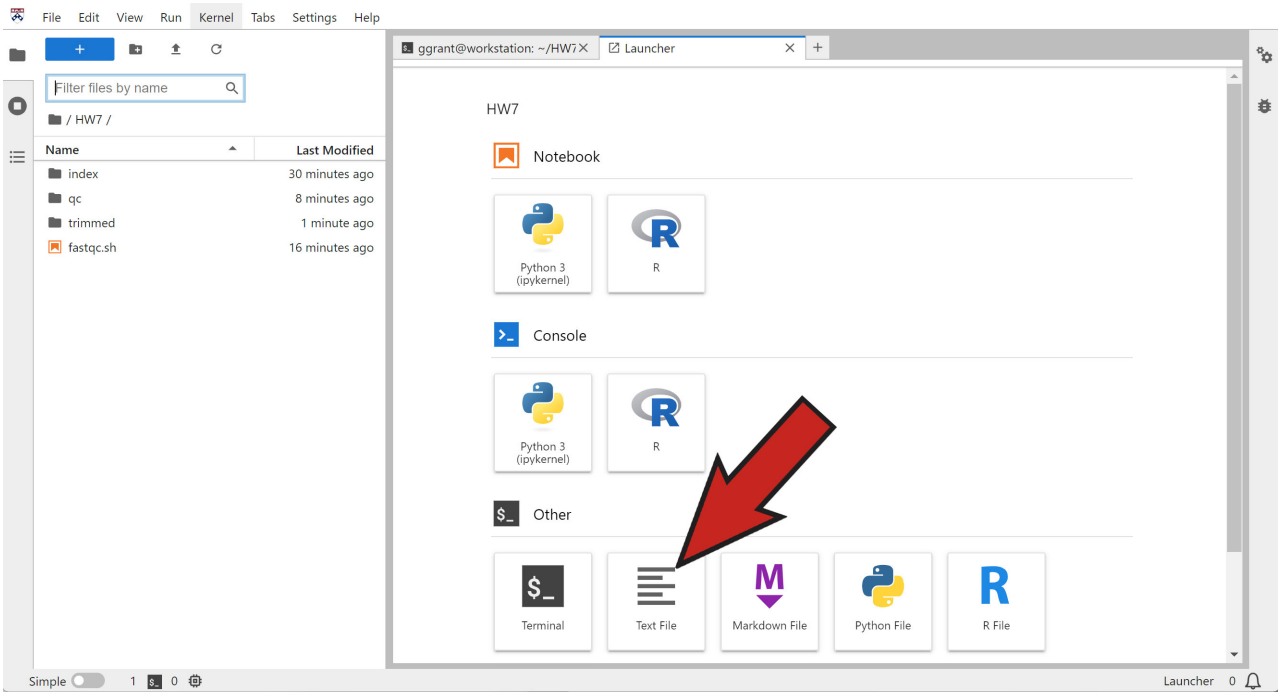
Note: If you start a new terminal, those vars are not persistent, they will need to be redefined. So if you do this over multiple days, make sure to keep the same session open or redefine the `FILES` *var. Similarly, if you open two terminals at the same time, you will need to define the vars in both, if you want to use them in both.*

QUALITY CONTROL

Next, we're going to do some Quality Control (QC) on the fastq files.

We are going to execute some more complicated commands that span multiple lines. The most headache-free way to do this is to put the commands in a shell script. A shell script is just a text

file of Unix commands. Feel free to use any text editor, there's one built into the system, here (red arrow):



Create a file in the `HW7` directory called `fastqc.sh` with the following code. If you copy/paste the code into that file (which you should do), you will have to remove the *one* line break *inside the quoted part* (but maintain a space, so `... fastqc --threads ...` and *not* `... fastqc--threads ...`. Don't remove the line breaks that are outside of the quotes.

```
for file in $FILES
do
    batch submit-job --job-name QC-$file --vcpu 2 --memory 3000 --command "fastqc
 --threads \$(nproc) --outdir qc /project/chip-seq/fastq/$file.fastq.gz"
done
```

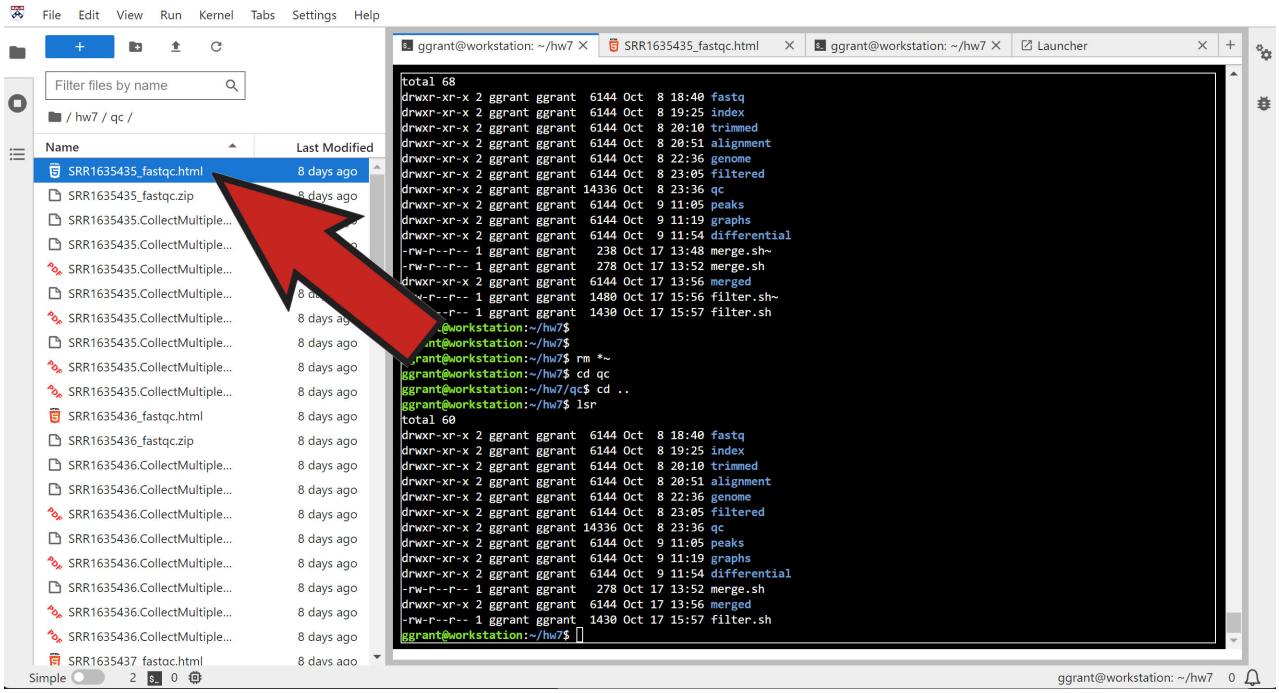
Execute it as follows:

```
source fastqc.sh
```

If it throws an error and doesn't even submit the job, look for typos, line breaks in the wrong place, missing spaces. If it fails to output what you expect, make sure you have the `FILES` variable set properly.

This should send eight jobs to eight cluster nodes. Notice how the loop fills in the variable '$file' for each of the eight files defined by the `FILES` variable that you defined earlier. That's what the dollar sign ($) achieves, it tells it you mean "variable" and not "string".

This should only take a few minutes. Monitor the progress using "`batch list-jobs`". If it failed, check your shell script for typos. There should be eight jobs on the list, one for each `fastq` file. Once it is done, go into the `HW7/qc` on the left panel file browser (see pic) and double-click on the `html` file `SRR1635435_fastqc.html`.



The report should come up. On the left of the report you should see 11 categories.

(2) Which ones did not pass with a green check?

(3) Go to the "Per base sequence content" graph, what does this say about the far right-hand positions in the reads? Can you conclude why it did not pass this metric with a green check?

Don't worry, the "per-base sequence content" issue is not a big issue since it just affects that last few bases of the reads. Typically one would use the QC reports to look for serious issues, but sequence data rarely pass all QC metrics and so we'll proceed today without worrying.

Next step in the pipeline is to "trim" the reads. This is a basic pre-processing step for ChIP-Seq to remove adapter sequences that may have gotten into the data.

TRIMMING

From the `HW7` directory, execute the following code in a shell script called `trim.sh`. *Again make sure to remove the line breaks so that the "*`batch`*" command is entirely on one line.*. This will again submit eight cluster jobs, monitor it until it finishes, should just take a few minutes.

```
for file in $FILES
do
    batch submit-job --job-name TRIM-$file --vcpu 2 --memory 7000 --command
    "trim_galore --cores \$(nproc) --gzip --output_dir
    trimmed /project/chip-seq/fastq/$file.fastq.gz"
done
```

(4) Look at the "trimming report" for file with ID `SRR1635460` and report how many reads were found to have adapters.

ALIGNMENT

Now comes the fun part, alignment. Make a subdirectory of HW7 called alignment and execute the following in a shell script called align.sh. *as usual the batch command needs to be on one line, so remove all (four) newlines and replace with spaces*)

```
for file in $FILES
do
    batch submit-job --job-name ALIGN-$file --vcpu 8 --memory 60000 --command
    "bowtie2 --threads \$(nproc) --seed 42 -x
    index/Homo_sapiens.GRCh38.dna.primary_assembly -U
    trimmed/${file}_trimmed.fq.gz | samtools sort --threads \$(nproc) -o
    alignment/$file.bam - "
done
```

This sends the eight alignments to (cluster) nodes. Monitor the progress, it will take at least 15 minutes if all goes well.

Once finished, move into the aligment directory. You should see eight bam files. Those are sam files but compressed. The zcat command won't help here because they're not properly gzipped. But try this command:

```
samtools view SRR1635460.bam | less -S
```

(5) What is the read ID and Cigar string of the first read in that bam file whose alignment involves an indel?

MERGING

This pipeline does not properly use (biological) replicates in a statistical way. All it does is merge replicates into one. That's what we'll do next.

Execute this declare command. You don't need to understand the declare command to follow this pipeline, just copy/paste it into the command line, you don't need to remove line breaks from this, it will cross six lines, but it should work. If not, you can always execute it from a shell script.

```
declare -A groupings=(
    [Input_Vehicle]="SRR1635435 SRR1635436"
    [Input_E2]="SRR1635437 SRR1635438"
    [FoxA1_Vehicle]="SRR1635459 SRR1635460"
    [FoxA1_E2]="SRR1635461 SRR1635462"
)
```

Execute the following as a shell script merge.sh. Do the usual thing with the line breaks, the entire batch command needs to be on one line (not three). This will submit to a cluster node a command called picard. Picard is a set of command line tools released by the BROAD institute, for manipulating high-throughput sequencing (HTS) data and formats such as SAM/BAM/CRAM and VC. This will take at least 10 minutes to run, monitor it (batch list-jobs) and make sure it succeeded.

```
for group in "${!groupings[@]}"
do
  files=( ${groupings[$group]} )
  batch submit-job --job-name MERGE-$group --vcpu 1 --memory 3500 --command
  "picard -Xmx3g MergeSamFiles --INPUT alignment/${files[0]}.bam --INPUT
  alignment/${files[1]}.bam --OUTPUT merged/$group.bam"
done
```

Look in the "merged" directory. You'll see there's now one file per condition. The ones called "`Input_E2.bam`" and "`Input_Vehicle.bam`" are the controls.

There are actually two types of controls here. The treatment group has been treated with estradiol. The "Vehicle" has not. The goal is to see how the treatment affects 'FoxA1" binding. While the Inputs are the (experimental) controls for the chromatin-immunoprecipitation assay, in that they were not immunoprecipiated at all (no antibody, no pulldown), but everything else was left the same.

Therefore, the meat-and-potatoes here is in the `FoxA1_E2` file. We will soon visualize this data in the genome browser to look for differential effects between `E2` and `Vehicle`, because that's where `FoxA1` binds *differentially*. But we musn't get confused by peaks that are also in the Input controls, because those are just strange artifacts of sequencing.

MARK DUPLICATES

Next we're going to "mark duplicates". Some regions of DNA just like to amplify. We calculated in class the probability of real dups and it's low. So this step will identify them, for later removal.

First redefine the `FILES` variable now to represent the four groups:

```
FILES="Input_Vehicle Input_E2 FoxA1_Vehicle FoxA1_E2"
```

Check to make sure the variable is defined:
```
echo $FILES
```

Now execute the following in a shell script `markdups.sh`. At this point I'm going to stop remiding you about the newline characters, but you'll need to keep doing that. This will take at least 15-20 minutes.

```
for file in $FILES
do
 batch submit-job --job-name DUPLICATES-$file --vcpu 1 --memory 3500 --command
 "picard -Xmx3g MarkDuplicates --INPUT merged/$file.bam --OUTPUT
 filtered/${file}_marked.bam --METRICS_FILE
 filtered/$file.MarkDuplicates.metrics.txt"
done
```

When this is finished it will have output several files to the `filtered` directory.

```
-rw-r--r-- 1 ggrant ggrant 1936920637 Oct 17 22:05 FoxA1_Vehicle_marked.bam
-rw-r--r-- 1 ggrant ggrant 2160556691 Oct 17 22:06 FoxA1_E2_marked.bam
-rw-r--r-- 1 ggrant ggrant 2504858222 Oct 17 22:08 Input_E2_marked.bam
-rw-r--r-- 1 ggrant ggrant 2595873317 Oct 17 22:08 Input_Vehicle_marked.bam
```

(6) The following command will show the record of one read (execute it from the HW7 directory:

```
samtools view merged/FoxA1_Vehicle.bam | head -n 100 | grep SRR1635460.8256696
```

And this will show the same read in the file marked for duplicates:

```
samtools view filtered/FoxA1_Vehicle_marked.bam | head -n 100 | grep SRR1635460.8256696
```

Note the different bitflag (column 2). What did it change from and to? Use this online bitflag interpreter and report the piece(s) of information that have actually changed.

```
https://broadinstitute.github.io/picard/explain-flags.html
```

FILTERING

Now comes the tedious part, a bunch of commands to clean up the data for various issues. This is called "preprocessing" and a successful high-throughput sequencing analysis always depends on it.

For the next steps, make sure the FILES variable is as it was redefined:
`echo $FILES`

```
ggrant@workstation:~/HW7/differential$ echo $FILES
Input_Vehicle Input_E2 FoxA1_Vehicle FoxA1_E2
```

First, we're going to remove known problematic regions where DNA sequencing doesn't work well for one reason or another. We're going to execute a handful of commands to achieve this. None of them need a shell script or to be submitted to the cluster, we will run them straight on the master. But if you want you can put all of them in one shell script, that's up to you.

Execute this command (make sure it's on one line, remove the line break) to establish the chromosome sizes. This uses an important command called `samtools` for parsing and manipulating SAM/BAM files.

```
samtools faidx --output genome/Homo_sapiens.GRCh38.dna.primary_assembly.fai
/project/genome/Homo_sapiens.GRCh38.dna.primary_assembly.fa
```

We only need the first two columns, so execute this `cut` command next:

```
cut -f 1,2 genome/Homo_sapiens.GRCh38.dna.primary_assembly.fai >
genome/Homo_sapiens.GRCh38.dna.primary_assembly.sizes
```

(7) The output of this is a file of chromosome sizes in your ∼/HW7/genome directory:
`Homo_sapiens.GRCh38.dna.primary_assembly.sizes`
There are more than the usual 25 (that's the 22 numbered ones, X, Y and MT (mitochondrial)). How many rows does the file actually have? The other ones are "contigs" which are pieces of human DNA that have been sequenced but they don't know where they go yet. Nonetheless they may be annotated with genes, so they are made available. The genome of most species have orphaned contigs like this. So they get their own name for now. They start with KI or GL. Use the appropriate grep command to

count how many start with KI and how many with GL.

Next we'll download a public file with the known blacklisted regions. You download web pages using the `wget` command as follows.

```
wget -q -O genome/GRCh38-blacklist.v3.bed
https://raw.githubusercontent.com/nf-core/chipseq/master/assets/blacklists/v3.0/GRCh38-blacklist.v3.bed
```

Next we'll use another important command `bedtools` for manipulating `bed` files. Bed files are files of genome coordinates. At minimum they have three columns: chromosome, start position and end position. First execute this bedtools command (as usual remove the line breaks)

```
bedtools sort -i genome/GRCh38-blacklist.v3.bed -g
genome/Homo_sapiens.GRCh38.dna.primary_assembly.sizes >
genome/Homo_sapiens.GRCh38.dna.primary_assembly.exclude_regions.bed
```

Now we have to invert the blacklisted regions, because the program we need to run wants a file of regions to include, not to exclude. Bedtools has a "complement" option to do just that. Execute it as follows (remove line breaks):

```
bedtools complement -i
genome/Homo_sapiens.GRCh38.dna.primary_assembly.exclude_regions.bed
-g genome/Homo_sapiens.GRCh38.dna.primary_assembly.sizes >
genome/Homo_sapiens.GRCh38.dna.primary_assembly.include_regions.bed
```

We need to download another config file from the web. Execute this from the `HW7` directory (make sure you're in the right place):

```
wget -q -O filtered/bamtools_filter_se.json
https://raw.githubusercontent.com/nf-core/chipseq/master/assets/bamtools_filter_se.json
```

Now we need to execute some cluster jobs. The next command will do the actual removing of dups, black-listed regions, etc. Run the following in its own shell script `job1.sh` (*as usual, the batch command needs to be on one line, not six, make sure to put spaces in place of the line breaks*)

```
for file in $FILES
do
    batch submit-job --job-name FILTER-$file --vcpu 4 --memory 15000
   --command "samtools view -F 0x004 -F 0x0400 -q 1 -L
   genome/Homo_sapiens.GRCh38.dna.primary_assembly.include_regions.bed -b
   filtered/${file}_marked.bam | bamtools filter -script
   filtered/bamtools_filter_se.json | samtools sort --threads
   \$(nproc) -o filtered/$file.bam - "
done
```

Wait for the previous to finish, it could take 10 or 15 minutes (recall you can monitor cluster job status with the command `batch list-jobs`)

Execute the following in a shell script called `job2.sh`. This uses `samtools` to create an index of the alignment (BAM) file for quick (random) access.

```
for file in $FILES
do
    batch submit-job --job-name INDEX-$file --vcpu 2 --memory 7000 --command
    "cd filtered; samtools index --threads \$(nproc) $file.bam"
done
```

Wait for the previous to finish and then execute the following in a shell script called `job3.sh`. This command calculates various statistics for the BAM files, in particular the number of mapped reads to be used (later) as a scaling factor.

```
for file in $FILES
do
    batch submit-job --job-name STATISTICS-$file --vcpu 2 --memory 7000
   --command "cd filtered; samtools stats --threads \$(nproc) --reference
   /project/genome/Homo_sapiens.GRCh38.dna.primary_assembly.fa -X $file.bam
   $file.bam.bai > $file.stats; samtools flagstat --threads \$(nproc) $file.bam >
   $file.flagstat; samtools idxstats --threads \$(nproc) $file.bam > $file.idxstats"
done
```

CONFIG FILES FOR GENOME BROWSER

Wait for the all of the previous cluster jobs to finish. We're finally ready to create the config files for the genome browser coverage plot tracks. Execute the following command from the `HW7` directory, using a shell script called `cov1.sh` (fix those newlines).

```
for file in $FILES
do
    batch submit-job --job-name BEDGRAPH-$file --vcpu 2 --memory 7000
    --command "SCALE_FACTOR=\$(grep '[0-9] mapped (' filtered/$file.flagstat
    | awk '{print 1000000/\$1}'); echo \$SCALE_FACTOR > graphs/$file.scale_factor.txt;
    bedtools genomecov -ibam filtered/$file.bam -bg -scale \$SCALE_FACTOR
    -fs 200 | sort -T '.' -k1,1 -k2,2n > graphs/$file.bedGraph"
done
```

When it's done, the `graphs` directory should have eight files

```
-rw-r--r-- 1 ggrant ggrant         10 Oct 18 01:20 Input_E2.scale_factor.txt
-rw-r--r-- 1 ggrant ggrant         10 Oct 18 01:20 FoxA1_E2.scale_factor.txt
-rw-r--r-- 1 ggrant ggrant         10 Oct 18 01:20 Input_Vehicle.scale_factor.txt
-rw-r--r-- 1 ggrant ggrant         10 Oct 18 01:20 FoxA1_Vehicle.scale_factor.txt
-rw-r--r-- 1 ggrant ggrant  721559170 Oct 18 01:22 FoxA1_Vehicle.bedGraph
-rw-r--r-- 1 ggrant ggrant  899689383 Oct 18 01:23 FoxA1_E2.bedGraph
-rw-r--r-- 1 ggrant ggrant 2236822015 Oct 18 01:26 Input_E2.bedGraph
-rw-r--r-- 1 ggrant ggrant 2282269817 Oct 18 01:26 Input_Vehicle.bedGraph
```

Because the bedGraph files are big files, we're only going to upload chromosome 1 to the genome browser. Use `grep` to put just the chromosome 1 lines of `FoxA1_E2.bedGraph` into a file named `FoxA1_E2_chr1.bedGraph`. Also give the file the following header line (make sure it's all on one line):

```
track type=bedGraph name=E2 description=E2 color=246,16,16
altColor=16,246,116 autoScale=on alwaysZero=on graphType=bar
```

NOTE: Make a backup copy of the file in case you accidentally destroy it while adding the header.

HINT: The easiest way to do this is probably to make the file with the header first, and then use double-redirect >> to concatenate the output of the `grep` *command.*

Now do the same for the other three bedGraph files, changing the "name" and "description" accordingly in the header lines. Use the names "Vehicle", "Input-E2" and "Input-Vehicle".

Use the following command to `gzip` the four files you just created:

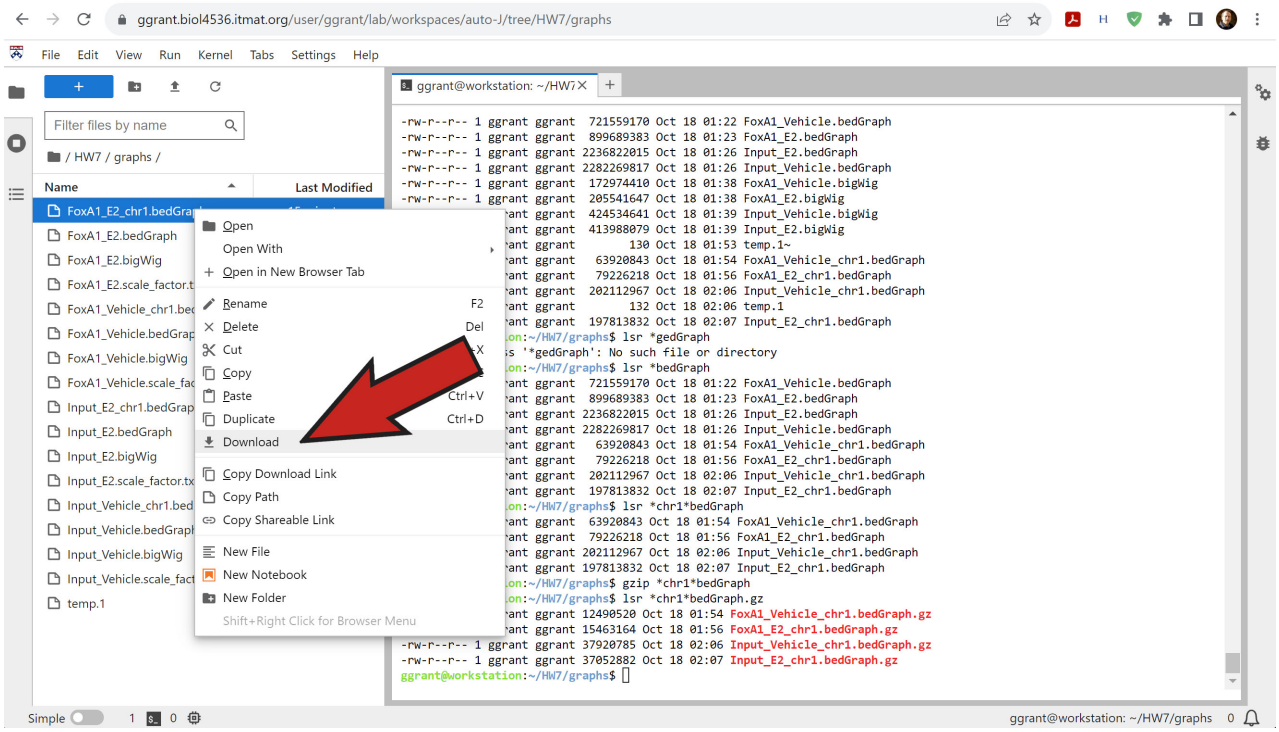
```
gzip *chr1*bedGraph
```

CONFIGURING THE GENOME BROWSER

You should now have the following four files.

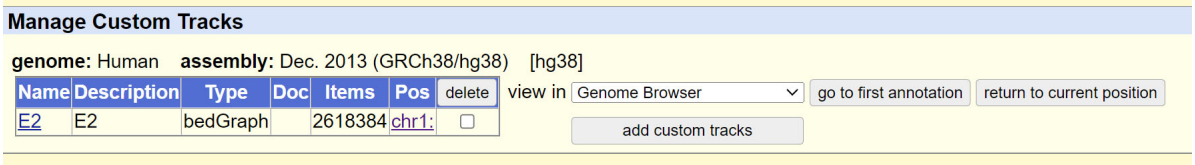
```
-rw-r--r-- 1 ggrant ggrant 12490520 Oct 18 01:54 FoxA1_Vehicle_chr1.bedGraph.gz
-rw-r--r-- 1 ggrant ggrant 15463164 Oct 18 01:56 FoxA1_E2_chr1.bedGraph.gz
-rw-r--r-- 1 ggrant ggrant 37920785 Oct 18 02:06 Input_Vehicle_chr1.bedGraph.gz
-rw-r--r-- 1 ggrant ggrant 37052882 Oct 18 02:07 Input_E2_chr1.bedGraph.gz
```

Now you'll have to transfer those files to your local machine (laptop or whatever) in order to upload to the genome browser. To do this, in the left panel, surf to the `HW7/graphs` directory and then right click on the file and choose download.

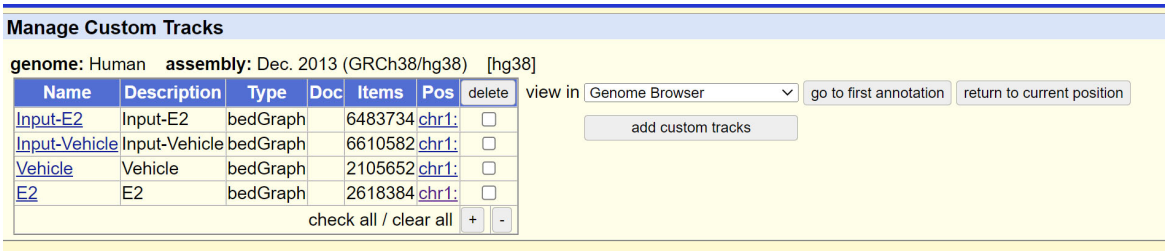


Once the four files are local, go to the genome browser, select Human build (GRCh38/hg38). Turn off all tracks except “Base Position” and “GENCODE V44”. Now go to the “Custom Tracks” page. Click “Choose File” and select one of the bedGraph files. Then hit “Submit” and wait.

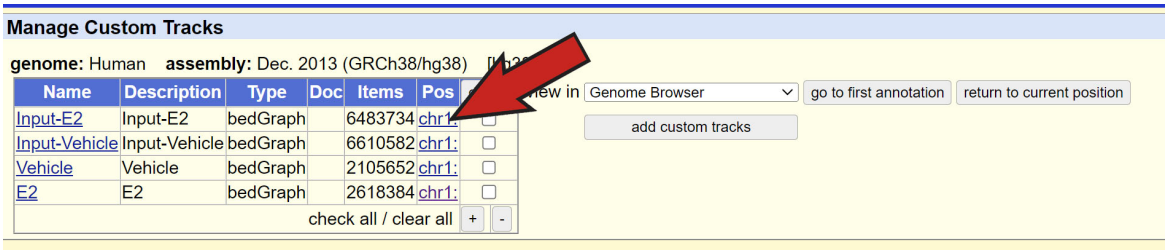
Wait for it to load, when it’s done you should see this:



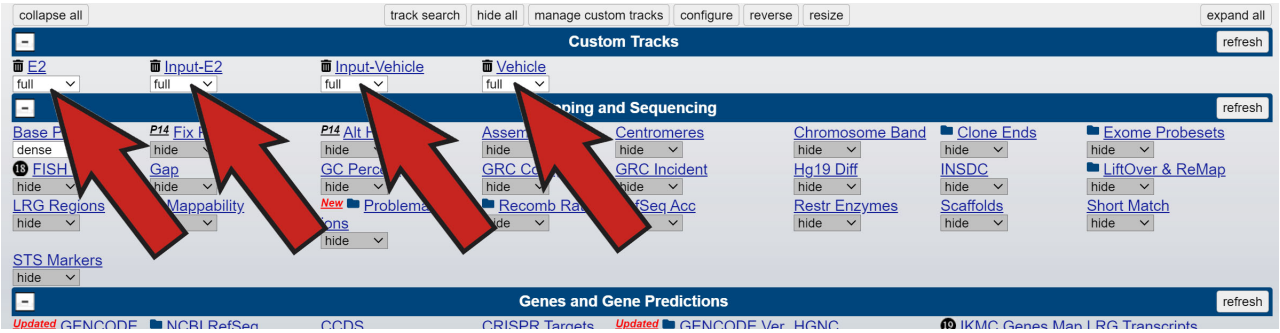
Upload the other three. You should see something like this



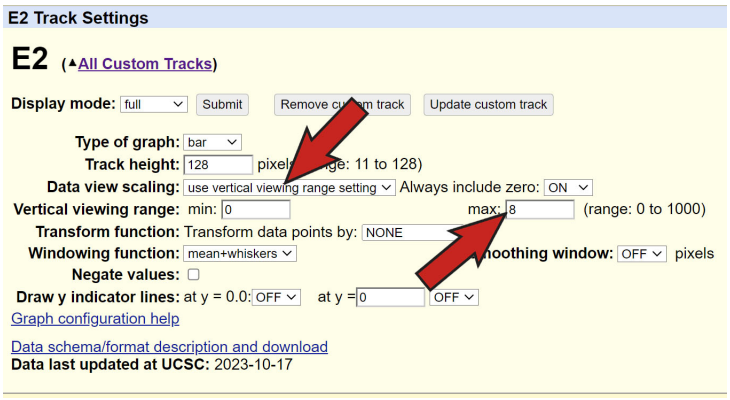
Click on one of these links to go to the Genome



Set the tracks to “full”



Click on the configuration link for one of the tracks and set “Data view scaling” and “max” as shown:



Now do the same for the other three tracks and surf to these coordinates:
`chr1:41,852,584-41,872,809`

(8) Do you see evidence for a differential effect? Explain. Provide a screen shot of the genome browser with the four tracks.

PEAK CALLING

Now that we have coverage plots, we can run a command to find the peaks.

First, define the variable:

```
CONDITIONS="Vehicle E2"
```

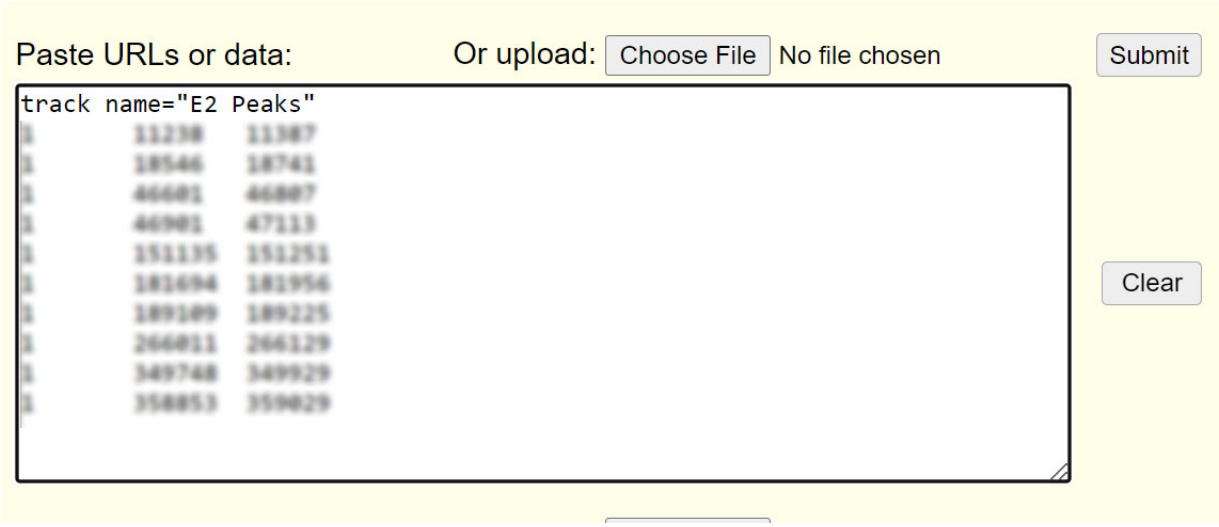
```
for condition in $CONDITIONS
do
    batch submit-job --job-name PEAK-$condition --vcpu 2 --memory 3500 --command
    "macs3 callpeak --gsize hs --format BAM --name $condition --treatment
    filtered/FoxA1_$condition.bam --control filtered/Input_$condition.bam --bdg
    --outdir peaks"
done
```

It should take about ten minutes to complete this step. Once done you should have these two files in the `peaks` directory. They end `.xls` but they’re just text files you can examine in the usual way.

```
-rw-r--r-- 1 ggrant ggrant 3261484 Oct 18 03:04 Vehicle_peaks.xls
-rw-r--r-- 1 ggrant ggrant 3510566 Oct 18 03:05 E2_peaks.xls
```

Take a look at the file `E2_peaks.xls`, you'll see the table of peaks has several columns including a p-value ($-\log_{10}(p\text{-val})$ more specifically).

(9) What is the most significant ($-\log_{10}(p\text{-val})$ among the first ten peaks called? (should be chromosome 1). Make a custom track called "E2 peaks" with just the first ten. Should look like this (*NOTE: I blurred out the actual numbers so you can do it yourself*)



Now surf to one (or more) of these peaks in the browser and supply a screen shot of it. It should show both the bar in the "E2 Peaks" track and the coverage plots of the four tracks.

DIFFERENTIAL ANALYSIS

```
batch submit-job --job-name DIFFERENTIAL --vcpu 4 --memory 15000 --command
  "macs3 bdgdiff --t1 peaks/Vehicle_treat_pileup.bdg --t2 peaks/E2_treat_pileup.bdg
  --c1 peaks/Vehicle_control_lambda.bdg --c2 peaks/E2_control_lambda.bdg
  --outdir differential -o Unique_Vehicle.bed Unique_E2.bed Common_Vehicle_E2.bed"
```

Once it is done, there should be three files in the `differential` directory.

```
ggrant@workstation:~/HW7/differential$ lsr
total 532
-rw-r--r-- 1 ggrant ggrant   5204 Oct 18 03:32 Unique_Vehicle.bed
-rw-r--r-- 1 ggrant ggrant  77388 Oct 18 03:32 Unique_E2.bed
-rw-r--r-- 1 ggrant ggrant 457234 Oct 18 03:32 Common_Vehicle_E2.bed
```

First we need to take the `differential/Unique_E2.bed` BED file and transform it a bit. We need to extract only the first three columns (chromosome, start of region, end of region), remove the non-standard chromosomes (if any) and add "chr" to all lines (so "1" becomes "chr1"), and finally correct the header line, which should not have chr prepended. The following command will achieve all of that (it's critical to remove that line break here):

```
cut -f 1-3 differential/Unique_E2.bed | grep -v "^GL" | grep -v "^KI" | awk '{print
"chr" $0}'| sed 's/chrtrack/track/'> Unique_E2_Peaks_Transformed.bed
```

Now, download the `Unique_E2_PeaksTransformed.bed` file to your local computer (your laptop or whatever).

Go to `http://great.stanford.edu`

Select Human: GRCh38 (UCSC hg38, Dec. 2013) as species assembly.

Upload the file `Unique_E2_Peaks_Transformed.bed` as the test regions BED file.

Click "Submit".

(10) Look at GO Biological Process section and report a hit you think is related to breast cancer and the corresponding (binomial) q-value and fold enrichment and a screen shot of the peak. (A q-value is like a p-value but it's "corrected" for multiple-testing - we'll learn about those in a couple weeks, for now treat it like a p-value.)

EXTRA CREDIT

(11) The fifth column of the file `arraydata1.txt` contains ID's. There are usually several separated by vertical lines. For example, like this:

`ref|NM_027530|gb|BC049127|gb|AK013185|gb|BC058259`

Write a script *using regexp capture* that does the following. For each row that has both an ID that starts "BC" followed by a number, and also an ID that starts "NM_" followed by a number (like in the example above), output the "BC" ID and the "NM" ID in two columns, separated by a tab. The first five rows of output should look like this:

```
BC030924        NM_007886
BC056205        NM_178884
BC019764        NM_025692
BC066042        NM_025920
BC060967        NM_027085
...
```

Have your script output to a file and report here the name and location of that file in your Unix space, so we can check it.
HINT: For how to use "capture" to get those numbers, see the example code we went over in class that was uploaded to Canvas.