Bedtools: Analyzing your aligned experiment

Use bedtools coverage to create a signal track

A ***signal track*** is a **bedGraph** (**BED3+**) file with an entry for every base in a defined set of regions (see https://genome.ucsc.edu/goldenpath/help/bedgraph.html). **bedGraph** files can be visualized in the Broad's **IGV** (**I**ntegrative **G**enomics **V**iewer) application (https://software.broadinstitute.org/software/igv/download) or in the **UCSC Genome Browser** (https://genome.ucsc.edu/).

The **bedtools coverage** function (https://bedtools.readthedocs.io/en/latest/content/tools/coverage.html), with the **-d** (per-base **d**epth) option produces output that can be made into a **bedGraph**. Here we'll analyze the per-base coverage of yeast RNAseq reads in our merged yeast gene regions.

Make sure you're in an **idev** session, then prepare a directory for this exercise.

Prepare for bedtools coverage

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGSday5
module load biocontainers
module load bedtools

mkdir -p $SCRATCH/core_ngs/bedtools_coverage
cp $CORENGS/catchup/bedtools_merge/merged*bed      $SCRATCH/core_ngs/bedtools_coverage/
cp $CORENGS/yeast_rnaseq/yeast_mrna.sort.filt.bam* $SCRATCH/core_ngs/bedtools_coverage/
```

Then calling **bedtools coverage** is easy. The "**A**" file will be our gene regions, and the "**B**" file will be the yeast RNAseq reads. We also use the **-d** (per-base **d**epth) and **-s** (force "strandedness") options.

```
cds; cd core_ngs/bedtools_coverage
bedtools coverage -s -d -a merged.good.sc_genes.bed -b yeast_mrna.sort.filt.bam > yeast_mrna.gene_coverage.txt

wc -l yeast_mrna.gene_coverage.txt # 8,829,317 lines!
```

It will complain a bit because our genes file includes the yeast plasmid "**2-micron**" but the RNAseq BAM doesn't include that contig. We'll ignore that warning.

The bedtools coverage output is a bit strange. It lists each region in the A file, followed by information from the B reads. Here the column order will be ***gene_chrom gene_start gene_end gene_name gene_score gene_strand offset_in_the_gene_region read_overlap count***.

Let's look at coverage for gene YAL067C:

```
cat yeast_mrna.gene_coverage.txt | grep -P 'YAL067C' | head -50
```

Will look like this:

```
chrI    7234    9016    YAL067C 1       -       1       0
chrI    7234    9016    YAL067C 1       -       2       0
chrI    7234    9016    YAL067C 1       -       3       0
chrI    7234    9016    YAL067C 1       -       4       0
chrI    7234    9016    YAL067C 1       -       5       0
chrI    7234    9016    YAL067C 1       -       6       0
chrI    7234    9016    YAL067C 1       -       7       0
chrI    7234    9016    YAL067C 1       -       8       0
chrI    7234    9016    YAL067C 1       -       9       0
chrI    7234    9016    YAL067C 1       -       10      0
chrI    7234    9016    YAL067C 1       -       11      0
chrI    7234    9016    YAL067C 1       -       12      0
chrI    7234    9016    YAL067C 1       -       13      0
chrI    7234    9016    YAL067C 1       -       14      0
chrI    7234    9016    YAL067C 1       -       15      0
chrI    7234    9016    YAL067C 1       -       16      0
chrI    7234    9016    YAL067C 1       -       17      1
chrI    7234    9016    YAL067C 1       -       18      1
chrI    7234    9016    YAL067C 1       -       19      1
chrI    7234    9016    YAL067C 1       -       20      1
chrI    7234    9016    YAL067C 1       -       21      1
chrI    7234    9016    YAL067C 1       -       22      1
chrI    7234    9016    YAL067C 1       -       23      1
chrI    7234    9016    YAL067C 1       -       24      1
chrI    7234    9016    YAL067C 1       -       25      1
chrI    7234    9016    YAL067C 1       -       26      1
chrI    7234    9016    YAL067C 1       -       27      1
chrI    7234    9016    YAL067C 1       -       28      1
chrI    7234    9016    YAL067C 1       -       29      1
chrI    7234    9016    YAL067C 1       -       30      1
chrI    7234    9016    YAL067C 1       -       31      1
chrI    7234    9016    YAL067C 1       -       32      1
chrI    7234    9016    YAL067C 1       -       33      1
chrI    7234    9016    YAL067C 1       -       34      1
chrI    7234    9016    YAL067C 1       -       35      1
chrI    7234    9016    YAL067C 1       -       36      1
chrI    7234    9016    YAL067C 1       -       37      1
chrI    7234    9016    YAL067C 1       -       38      2
chrI    7234    9016    YAL067C 1       -       39      2
chrI    7234    9016    YAL067C 1       -       40      2
chrI    7234    9016    YAL067C 1       -       41      3
chrI    7234    9016    YAL067C 1       -       42      3
chrI    7234    9016    YAL067C 1       -       43      3
chrI    7234    9016    YAL067C 1       -       44      3
chrI    7234    9016    YAL067C 1       -       45      4
chrI    7234    9016    YAL067C 1       -       46      4
chrI    7234    9016    YAL067C 1       -       47      4
chrI    7234    9016    YAL067C 1       -       48      4
chrI    7234    9016    YAL067C 1       -       49      4
chrI    7234    9016    YAL067C 1       -       50      4
```

A proper **bedGraph** file has only 4 columns: ***chrom start end value*** and does not need to include positions with 0 reads, so we'll convert the **bedtools coverage** output to **bedGraph** using **awk**. We re-sort the output so that plus and minus strand positions are adjacent.

```
cat yeast_mrna.gene_coverage.txt | awk '
BEGIN{FS=OFS="\t"}
{if ($8>0) {print $1,$2-1+$7,$2+$7,$8}}' | \
  sort -k1,1 -k2,2n -k3,3n > yeast_mrna.gene_coverage.almost.bedGraph

wc -l yeast_mrna.gene_coverage.almost.bedGraph  # 5,710,186 -- better, but still big
```

While we probably could consider this file to have **bedGraph** format, it's preferable to combine adjacent per-base coordinates with the same count into larger regions, e.g.

```
# per-base counts
chrI    7271    7272    2
chrI    7272    7273    2
chrI    7273    7274    2
chrI    7274    7275    3
chrI    7275    7276    3
chrI    7276    7277    3
chrI    7277    7278    3

# corresponding region counts
chrI    7271    7274    6
chrI    7274    7278    12
```

Here's some **awk** to do this:

```
cat yeast_mrna.gene_coverage.almost.bedGraph | awk '
BEGIN{FS=OFS="\t"; chr=""; start=-1; end=-1; count=0}
{if (chr != $1) { # new contig; finish previous
   if (count > 0) { print chr,start,end,count }
   chr=$1; start=$2; end=$3; count=$4
 } else if (($2==end || $2==end+1) && ($4==count)) { # same or adjacent position with same count
   end=$3;
 } else { # new region on same contig; finish prev
   if (count > 0) { print chr,start,end,count}
   start=$2; end=$3; count=$4
 }
}
END{ # finish last
  if (count > 0) { print chr,start,end,count }
}' > yeast_mrna.gene_coverage.bedGraph

wc -l yeast_mrna.gene_coverage.bedGraph  # 1,048,510 -- much better!
```

Make sure the total counts match!

```
cat yeast_mrna.gene_coverage.txt | awk '
  BEGIN{tot=0}{tot=tot+$8}END{print tot}'          # should be 86703686
cat yeast_mrna.gene_coverage.almost.bed | awk '
  BEGIN{tot=0}{tot=tot+$4}END{print tot}'          # should also be 86703686
cat yeast_mrna.gene_coverage.bedGraph | awk '
  BEGIN{tot=0}{tot=tot+$4*($3-$2)}END{print tot}'  # should also be 86703686
```

Now our **yeast_mrna.gene_coverage.bedGraph** file is a proper **bedGraph**, whose first lines look like this:

```
chrI    7250    7271    1
chrI    7271    7274    2
chrI    7274    7278    3
chrI    7278    7310    4
chrI    7310    7317    3
chrI    7317    7349    2
chrI    7349    7353    1
chrI    7500    7556    1
chrI    8851    8891    1
chrI    11919   11951   1
```

x

A brief introduction to bedtools

Now that we have a BAM file with only the reads we want included, we can do some more sophisticated analysis using **bedtools**. **Bedtools** changes from version to version, and here we are using version 2.22, the newest version, and what is currently installed on **stampede**. You can check what versions of **bedtools** are installed by using the following command on **stampede**:

```
module spider bedtools
```

First, log on to the **login8** node on **stampede** and make a directory in **scratch** called **bedtools** in your scratch folder. Then copy your filtered BAM file from the **samtools** section into this folder.

```
ssh user@login8.stampede.tacc.utexas.edu #if you are not already logged in!
cd $SCRATCH/core_ngs
mkdir bedtools
cd samtools
cp yeast_pairedend_sort.mapped.q1.bam ../bedtools
cd ../bedtools
```

If you were unable to make the filtered and sorted BAM file from the previous section, you can copy it over from my scratch directory:

```
cd bedtools
cp /scratch/01786/awh394/core_ngs/bedtools/yeast_pairedend_sort.mapped.q1.bam .
```

bedtools bamtofastq: converting a BAM file to a fastq file

Sometimes, especially when working with external data, we need to go from a BAM file back to a fastq file. This can be useful for re-aligning reads using a different aligner, different settings on the original aligner used. It can also be useful for extracting the sequence of interesting regions of the genome after you have manipulated your BAM file.

For this exercise, you'll be using bamtofastq. This function takes an aligned BAM file as input and outputs a fastq format file. You can use the options if you have paired end data to output R1 and R2 reads for your fastq file. This type of function is especially useful if you need to to analyze sequences after you've compared several BAM or bed files.

```
bedtools bamtofastq -i input.bam -fq output.fastq
```

Exercise 1: convert BAM to fastq and look at the quality scores

solution code

```
module load bedtools
bedtools bamtofastq -i yeast_pairedend_sort.mapped.q1.bam -fq yeast_pairedend_sort.mapped.q1.fastq #takes 1-2
minutes
more yeast_pairedend_sort.mapped.q1.fastq
```

Here is an example of two sequences (and their corresponding quality scores):

two lines of a fastq file

```
@HWI-ST1097:127:C0W5VACXX:5:2212:10568:79659
TACCCTCCAATTACCCATATCCAACCCACTGCCACTTACCCTACCATTACCCTACCATCCACCATGACCTACTCACCATACTGTTCTTCTACCCACCATAT
+
CCCFFFFFHHHHHJJJJJIIJJJJIJJIJJIJJIIIIJJJIJJIJJIJJIJJJJJJJJJJIIGGIGEGAEHFEFFEFFFDEEE@CCEDCDDD>ACBBDCA@
@HWI-ST1097:127:C0W5VACXX:5:2115:19940:13862
TAGGGTAAGTTTGAGATGGTATATACCCTACCATCCACCATGACCTACTCACCATACTGTTCTTCTACCCACCATATTGAAACGCTAACAAATGATCGTAA
+
?B@DF2ADHFHHFHJIIIGCIHIGGIJJJJGHIIIGIJEHHIGGGAHEGGFGHIECGIJIIIJIIIIIJJJJJJE>EHDHEEEBCDD?CDDBDDDDDDCDB
```

As we discussed earlier, the top line is the identifier for the sequence produced, the second line defines which bases were produced, the third line indicates the strand the sequence is aligned to, and the fourth line indicates the ASCII based quality scores for each character in the second line.

bedtools bamtobed: converting a BAM file into a bed file

While it's useful to be able to look at the fastq file, many analyses will be easiest to perform in bed format. Bed format is a simple tab delimited format that designates various properties about segments of the genome, defined by the chromosome, start coordinates and end coordinates. **Bedtools** provides a simple utility to convert BAM files over into bed files, termed bamtobed.

```
bedtools bamtobed -i input.bam > output.bed #output to a file
bedtools bamtobed -i input.bam | more #output to more
```

Note that the output will be piped to standard out unless you redirect to a program (head, more, less) or a file (output.bed). Now we'll put this example to use and convert our filtered BAM file from the samtools section into a bed file.

Exercise 2: Convert the filtered yeast paired end BAM to bed using bamtobed, look at your file in more, and find the number of lines in the file

Hint: direct the output to a file first, then use more to look at the converted file visually**;** use ctrl+c to quit more.

solution code

```
module load bedtools #if you haven't loaded it in for this session
bedtools bamtobed -i yeast_pairedend_sort.mapped.q1.bam > yeast_pairedend_sort.mapped.q1.bed

more yeast_pairedend_sort.mapped.q1.bed #to examine the bed file visually
wc -l yeast_pairedend_sort.mapped.q1.bed #to get the number of lines in a file
```

Here is what my output looks like:

output from the code above

```
wc -l yeast_pairedend_sort.mapped.q1.bed
528976 yeast_pairedend_sort.mapped.q1.bed
more yeast_pairedend_sort.mapped.q1.bed
chrI    219    320    HWI-ST1097:127:C0W5VACXX:5:2212:10568:79659/1    37    +
chrI    266    344    HWI-ST1097:127:C0W5VACXX:5:2115:19940:13862/2    29    +
chrI    368    469    HWI-ST1097:127:C0W5VACXX:5:2115:19940:13862/1    29    -
chrI    684    785    HWI-ST1097:127:C0W5VACXX:5:2212:10568:79659/2    37    -
chrI    871    955    HWI-ST1097:127:C0W5VACXX:5:1103:4918:43976/2    29    +
chrI    871    948    HWI-ST1097:127:C0W5VACXX:5:1104:2027:42518/2    29    +
chrI    871    948    HWI-ST1097:127:C0W5VACXX:5:1109:3153:38695/2    29    +
chrI    871    948    HWI-ST1097:127:C0W5VACXX:5:2109:6222:11815/2    29    +
chrI    871    948    HWI-ST1097:127:C0W5VACXX:5:2113:5002:59471/2    29    +
chrI    871    948    HWI-ST1097:127:C0W5VACXX:5:2113:7803:87146/2    29    +
chrI    971    1072    HWI-ST1097:127:C0W5VACXX:5:1103:4918:43976/1    29    -
chrI    978    1079    HWI-ST1097:127:C0W5VACXX:5:1104:2027:42518/1    29    -
chrI    978    1079    HWI-ST1097:127:C0W5VACXX:5:1109:3153:38695/1    29    -
chrI    978    1079    HWI-ST1097:127:C0W5VACXX:5:2109:6222:11815/1    29    -
chrI    978    1079    HWI-ST1097:127:C0W5VACXX:5:2113:5002:59471/1    29    -
chrI    978    1079    HWI-ST1097:127:C0W5VACXX:5:2113:7803:87146/1    29    -
chrI    978    1079    HWI-ST1097:127:C0W5VACXX:5:2203:1231:50183/1    37    -
```

Note the "stacks" of reads that are occurring on similar coordinates on the same strand of the genome. We'll deal with those in the **bedtools merge** section.

See also: **bedtools bedtobam**, if you need to get back to a bam file from a bed file (some programs take bam files as input). Documentation here.

bedtools coverage: how much of the genome does my data cover?

One way of characterizing data is to understand what percentage of the genome your data covers. What type of experiment you performed should affect the coverage of your data. A ChIP-seq experiment will cover binding sites, and a RNA-seq experiment will cover expressed transcripts. **Bedtools** coverage allows you to compare one bed file to another and compute the breadth and depth of coverage.

```
bedtools coverage -a experiment.bed -b reference_file.bed
```

The resulting output will contain several additional columns which summarize this information:

After each interval in B, **coverageBed** will report:

1. The number of features in A that overlapped (by at least one base pair) the B interval.
2. The number of bases in B that had non-zero coverage from features in A.
3. The length of the entry in B.
4. The fraction of bases in B that had non-zero coverage from features in A.

For this exercise, we'll use a bed file that summarizes the S. cerevisiae genome, version 3 (aka sacCer3). For this class, I've made a bed file out of the genome, using the file sacCer3.chrom.sizes. First go and copy the file over from my scratch directory:

```
cd bedtools #if you aren't already there
cp /scratch/01786/awh394/core_ngs.test/bedtools/sacCer3.chrom.sizes.bed .
```

```
more sacCer3.chrom.sizes.bed
chrIV     1    1531933
chrXV     1    1091291
chrVII    1    1090940
chrXII    1    1078177
chrXVI    1    948066
chrXIII   1    924431
chrII     1    813184
chrXIV    1    784333
chrX      1    745751
chrXI     1    666816
chrV      1    576874
chrVIII   1    562643
chrIX     1    439888
chrIII    1    316620
chrVI     1    270161
chrI      1    230218
chrM      1    85779
```

The format is bed3 - just chrom, start (which is always 1) and stop, which is always the length of the chromosome, for this type of bed file.

Now use **bedtools coverage** to find the coverage of the file output.bed over the sacCer3 genome and examine the output coverage.

Exercise 3: Find the coverage of your bed file over the sacCer3 genome

solution code

```
module load bedtools #again, if not already loaded
bedtools coverage -a sacCer3.chrom.sizes.bed -b yeast_pairedend_sort.mapped.q1.bed > sacCer3coverage.bed
more sacCer3coverage.bed #this file should have 17 lines, one for each chromosome
```

And here is what my output looks like:

```
more sacCer3coverage.bed
chrIV      1    1531933   70633    1026387   1531932   0.6699951
chrXV      1    1091291   47871    710376    1091290   0.6509507
chrVII     1    1090940   49762    722821    1090939   0.6625677
chrXII     1    1078177   48155    658373    1078176   0.6106359
chrXVI     1    948066    43531    612122    948065    0.6456540
chrXIII    1    924431    40054    618798    924430    0.6693833
chrII      1    813184    35818    539222    813183    0.6631004
chrXIV     1    784333    32565    513382    784332    0.6545468
chrX       1    745751    30743    472357    745750    0.6333986
chrXI      1    666816    27950    446567    666815    0.6697015
chrV       1    576874    26918    381078    576873    0.6605926
chrVIII    1    562643    23424    356421    562642    0.6334774
chrIX      1    439888    15953    276571    439887    0.6287319
chrIII     1    316620    13701    199553    316619    0.6302623
chrVI      1    270161    10662    167222    270160    0.6189740
chrI       1    230218    7972     128701    230217    0.5590421
chrM       1    85779     3264     58599     85778     0.6831472
```

It's worth noting that just computing coverage over the genome isn't the most useful thing, but you might compute coverage over a set of genes or regions of interest. Coverage is really useful coupled with intersect or subtract as well.

bedtools merge: collapsing bookended elements (or elements within a certain distance)

When we originally examined the bed files produced from our BAM file, we can see many reads that overlap over the same interval. While this level of detail is useful, for some analyses, we can collapse each read into a single line, and indicate how many reads occurred over that genomic interval. We can accomplish this using **bedtools** merge.

```
bedtools merge [OPTIONS] -i experiment.bed > experiment.merge.bed
```

Bedtools merge also directs the output to standard out, to make sure to point the output to a file or a program. **While we haven't discussed the options for each bedtools function in detail, here they are very important.** Many of the options define what to do with each column (-c) of the output (-o). This defines what type of operation to perform on each column, and in what order to output the columns. **Standard bed6 format is chrom, start, stop, name, score, strand** and controlling column operations allows you to control what to put into each column of output. The valid operations defined by the -o operation are as follows:

- sum, min, max, absmin, absmax,
- mean, median,
- collapse (i.e., print a delimited list (duplicates allowed)),
- distinct (i.e., print a delimited list (NO duplicates allowed)),
- count
- count_distinct (i.e., a count of the unique values in the column)

For this exercise, we'll be summing the number of reads over a region to get a score column, using distinct to choose a name, and using distinct again to keep track of the strand. For the -c options, define which columns to operate on, in the order you want the output. In this case, to keep the standard bed format, we'll list as -c 5,4,6 and -o count_distinct,sum,distinct, to keep the proper order of name, score, strand. Strandedness can also be controlled with **merge**, using the -s option.

Exercise 4: Use bedtools merge to merge an experiment, look at the output and see how many lines there are in the file.

Hint: make sure to remove whitespace between lists for the -c and -o options!

solution code

```
bedtools merge -s -c 4,5 -o count_distinct,sum -i yeast_pairedend_sort.mapped.q1.bed > yeast_pairedend_sort.
mapped.q1.merge.bed
more yeast_pairedend_sort.mapped.q1.merge.bed
wc -l yeast_pairedend_sort.mapped.q1.merge.bed

#without strand considered
bedtools merge -c 4,5,6 -o count_distinct,sum,distinct -i yeast_pairedend_sort.mapped.q1.bed >
yeast_pairedend_sort.noStrand.mapped.q1.merge.bed
```

```
wc -l yeast_pairedend_sort.noStrand.mapped.q1.merge.bed
40319 yeast_pairedend_sort.noStrand.mapped.q1.merge.bed #without the -s option

wc -l yeast_pairedend_sort.mapped.q1.merge.bed
76601 yeast_pairedend_sort.mapped.q1.merge.bed #with the -s option

more yeast_pairedend_sort.mapped.q1.merge.bed
chrI    219     344     +    2     66
chrI    368     469     -    1     29
chrI    684     785     -    1     37
chrI    871     955     +    6     174
chrI    971     1079    -    7     211
chrI    1216    1322    +    6     157
chrI    1347    1437    -    6     157
chrI    2892    2993    +    14    406
chrI    3010    3111    +    1     37
chrI    3013    3107    -    14    406

more yeast_pairedend_sort.noStrand.mapped.q1.merge.bed
chrI    219     344     2     66     +
chrI    368     469     1     29     -
chrI    684     785     1     37     -
chrI    871     955     6     174    +
chrI    971     1079    7     211    -
chrI    1216    1322    6     157    +
chrI    1347    1437    6     157    -
chrI    2892    2993    14    406    +
chrI    3010    3111    15    443    +,-
```

Note the change in column order in the first set of commands. We can use awk like this to change the column order, either piped in the original command or after the fact:

using awk for column reordering

```
#after the creation of the first file
cat yeast_pairedend_sort.mapped.q1.merge.bed | awk '{print $1 "\t" $2 "\t" $3 "\t" $5 "\t" $6 "\t" $4}' >
yeast_pairedend_sort.mapped.q1.merge.reorder.bed

#piped in-line
bedtools merge -s -c 4,5 -o count_distinct,sum -i yeast_pairedend_sort.mapped.q1.bed | awk '{print $1 "\t" $2
"\t" $3 "\t" $5 "\t" $6 "\t" $4}' > yeast_pairedend_sort.mapped.q1.merge.bed
```

bedtools intersect: identifying where two experiments overlap (or don't overlap)

One useful way to compare two experiments (especially biological replicates, or similar experiments in two yeast strains/cell lines/mouse strains) is to compare where reads in one experiment overlap with reads in another experiment. **Bedtools** offers a simple way to do this using the intersect function.

bedtools intersect options

```
bedtools intersect [OPTIONS] -a <FILE> \
                                                         -b <FILE1, FILE2, ..., FILEN>
```

The intersect function has many options that control how to report the intersection. We'll be focusing on just a few of these options, listed below.

-a and -b indicate what files to intersect. in -b, you can specify one, or several files to intersect with the file specified in -a.

- wa: Write the original entry in A for each overlap.
- wb: Write the original entry in B for each overlap. Useful for knowing what A overlaps. Restricted by -f and -r.
- loj: Perform a “left outer join”. That is, for each feature in A report each overlap with B. If no overlaps are found, report a NULL feature for B.
- wo: Write the original A and B entries plus the number of base pairs of overlap between the two features. Only A features with overlap are reported. Restricted by -f and -r.
- wao: Write the original A and B entries plus the number of base pairs of overlap between the two features. However, A features w/o overlap are also reported with a NULL B feature and overlap = 0. Restricted by -f and -r.
- f: Minimum overlap required as a fraction of A. Default is 1E-9 (i.e. 1bp).

- v: Only report those entries in A that have no overlap in B. Restricted by -f and -r. Useful to report what doesn't overlap, the inverse of typical usage.
- names: When using multiple databases (-b), provide an alias for each that will appear instead of a file Id when also printing the DB record.

In this section, we'll intersect two human experiments - one from sequencing RNA, and one from sequencing micro RNA. Copy these files over to your directory:

copy some files over to intersect

```
cd $SCRATCH/core_ngs/
mkdir intersect
cd intersect
cp /corral-repl/utexas/BioITeam/core_ngs_tools/alignment/bam/human_mirnaseq_hg19.bam .
cp /corral-repl/utexas/BioITeam/core_ngs_tools/alignment/bam/human_rnaseq_bwa.bam .
ls -lah
```

```
-rwxrwxr-x 1 awh394 G-801021  19M May 22 18:57 human_mirnaseq_hg19.bam
-rwxrwxr-x 1 awh394 G-801021 6.6M May 22 18:57 human_rnaseq_bwa.bam
```

Before we can intersect these files, we need to perform the pipeline we used in **samtools** to **index**, **sort** and **filter** the files, and **bedtools** to convert from BAM over to bed, then collapse down the files using **merge**. Below is a little workflow to help you through it on the files you just copied above.

My output (for length of bed files) is in the comments.

a samtools/bedtools workflow

```
module load samtools #if you haven't loaded it up this session

#sort both files
samtools sort human_mirnaseq_hg19.bam human_mirnaseq_hg19_sort # will take 1-2 minutes
samtools sort human_rnaseq_bwa.bam human_rnaseq_bwa_sort # will take 1-2 minutes

#index the new files
samtools index human_mirnaseq_hg19_sort.bam
samtools index human_rnaseq_bwa_sort.bam

#filter the sorted files, reindex the new filtered files
samtools view -b -F 0x04 -q 1 -o human_mirnaseq_hg19_sort.mapped.q1.bam human_mirnaseq_hg19_sort.bam
samtools view -b -F 0x04 -q 1 -o human_rnaseq_bwa_sort.mapped.q1.bam human_rnaseq_bwa_sort.bam
samtools index human_mirnaseq_hg19_sort.mapped.q1.bam
samtools index human_rnaseq_bwa_sort.mapped.q1.bam

#convert filtered bam files to bed format
module load bedtools #if you haven't loaded it in for this session

bedtools bamtobed -i human_mirnaseq_hg19_sort.mapped.q1.bam > human_mirnaseq_hg19_sort.mapped.q1.bed
bedtools bamtobed -i human_rnaseq_bwa_sort.mapped.q1.bam > human_rnaseq_bwa_sort.mapped.q1.bed

#check the length of the files:
wc -l *.bed
#  164806 human_mirnaseq_hg19_sort.mapped.q1.bed
#   22538 human_rnaseq_bwa_sort.mapped.q1.bed

#merge the bed files, check the length again
bedtools merge -s -c 4,5 -o count_distinct,sum -i human_mirnaseq_hg19_sort.mapped.q1.bed | awk '{print $1 "\t"
$2 "\t" $3 "\t" $5 "\t" $6 "\t" $4}' > human_mirnaseq_hg19_sort.mapped.q1.merge.bed
bedtools merge -s -c 4,5 -o count_distinct,sum -i human_rnaseq_bwa_sort.mapped.q1.bed | awk '{print $1 "\t" $2
"\t" $3 "\t" $5 "\t" $6 "\t" $4}' > human_rnaseq_bwa_sort.mapped.q1.merge.bed

wc -l *.merge.bed
# 14794 human_mirnaseq_hg19_sort.mapped.q1.merge.bed
#  7134 human_rnaseq_bwa_sort.mapped.q1.merge.bed
```

If we run low on time, you can copy the merged bed files over from my directory on scratch:

```
cds
cd intersect
cp /scratch/01786/awh394/core_ngs/intersect/human_mirnaseq_hg19_sort.mapped.q1.merge.bed .
cp /scratch/01786/awh394/core_ngs/intersect/human_rnaseq_bwa_sort.mapped.q1.merge.bed .
```

Exercise 5: Intersect two experiments using intersect and examine the output

My output is commented in this code block.

```
cd intersect
module load bedtools #if you haven't loaded it up yet this session
bedtools intersect -wo -a human_rnaseq_bwa_sort.mapped.q1.merge.bed -b human_mirnaseq_hg19_sort.mapped.q1.merge.
bed > hg19_rnaseq_mirnaseq_intersect.bed

wc -l hg19_rnaseq_mirnaseq_intersect.bed
#38 hg19_rnaseq_mirnaseq_intersect.bed

more hg19_rnaseq_mirnaseq_intersect.bed
#chr1    20987370    20987471    1    37    -    chr1    20987402    20987430    1    12    -    28
#chr1    25555557    25555616    1    37    +    chr1    25555612    25555636    1    2    -    4
#chr1    25555557    25555617    1    37    -    chr1    25555612    25555636    1    2    -    5
#chr1    28906396    28906497    1    37    +    chr1    28906368    28906405    6    246    -    9
#chr1    33245783    33245884    1    37    +    chr1    33245880    33245908    1    24    -    4
```

Using the options we've specified (-wo) the resulting file will have entries for file A, file B and the number of base pairs overlap between the feature in A and the features in B, but **we'll only retain lines where there is an overlap between A and B**. We could also use the -v option to only contain areas with NO intersection, or control the intersections with **-f** and **-r** options. **Bedtools intersect** is a powerful tool, and it's always a good idea to ask "what is this code going to do?" while you're testing analysis workflows. It can be very useful to pipe your output to **more** when you are unsure of the output of a command, as such:

pipe-ing output to more

```
bedtools intersect -wo -a human_rnaseq_bwa_sort.mapped.q1.merge.bed -b human_mirnaseq_hg19_sort.mapped.q1.merge.
bed | more
```

bedtools closest: when you want to know how far your regions are from a test set

The manual page for **bedtools closest** has a really nice image of how closest behaves with overlapping options. Bedtools closest first looks for any overlaps of B with A, if it finds an overlap, the overlap in B with the highest proportional overlap with A is reported. If there are no overlaps, then it looks for the closest genomic feature proximal to A (using distance from the start or end of A to do this).

bedtools intersect options

```
bedtools closest [OPTIONS] -a <FILE> \
                           -b <FILE1, FILE2, ..., FILEN>
```

Much like **bedtools intersect**, **bedtools closest** takes an A file and a series of B files. So if you wanted to determine the distance of your regions of interest to several different classes of genes, **bedtools closest** would be a useful tool for that analysis.

- s: Require same strandedness. That is, find the closest feature in B that overlaps A on the _same_ strand. By default, overlaps are reported without respect to strand.
- S: Require opposite strandedness. That is, find the closest featurein B that overlaps A on the _opposite_ strand. By default, overlaps are reported without respect to strand.
- d: In addition to the closest feature in B, report its distance to A as an extra column. The reported distance for overlapping features will be 0.
- D: Like *-d*, report the closest feature in B, and its distance to A as an extra column. However unlike *-d*, use negative distances to report upstream features.
    - *ref* Report distance with respect to the reference genome. B features with a lower (start, stop) are upstream.
    - *a* Report distance with respect to A. When A is on the - strand, "upstream" means B has a higher (start,stop).
    - *b* Report distance with respect to B. When B is on the - strand, "upstream" means A has a higher (start,stop).
- io: Ignore features in B that overlap A. That is, we want close, yet not touching features only.
- iu: Ignore features in B that are upstream of features in A. This option requires -D and follows its orientation rules for determining what is "upstream".
- id: Ignore features in B that are downstream of features in A. This option requires -D and follows its orientation rules for determining what is "downstream"
- names: When using multiple databases (-b), provide an alias for each that will appear instead of a file Id when also printing the DB record.

In this section, we'll intersect the human_rnaseq_bwa_sort.mapped.q1.merge.bed file with some protein coding genes from Gencode (hg19). First go copy a couple files from my scratch directory:

copy some gencode files over

```
cd $SCRATCH/core_ngs
mkdir closest
cd closest
cp /scratch/01786/awh394/core_ngs.test/closest/gencode.v19.proteincoding.genes.sort.merge.final .
cp ../intersect/human_rnaseq_bwa_sort.mapped.q1.merge.bed .
#or:
cp /scratch/01786/awh394/core_ngs/closest/human_rnaseq_bwa_sort.mapped.q1.merge.bed .
```

```
-rwxrwxr-x 1 awh394 G-801021 646K May 22 20:41 gencode.v19.proteincoding.genes.sort.merge.final
```

Exercise 6: Identify the closest protein coding genes (on the same strand) for the human rnaseq file using closest, then sort by the distance column (largest, then smallest distance first).

My output is commented in this code block.

```
cd closest
module load bedtools #if you haven't loaded it up yet this session
sort -k1,1 -k2,2n human_rnaseq_bwa_sort.mapped.q1.merge.bed > human_rnaseq_bwa.mapped.q1.merge.sort.bed #need
to sort both files to the same order
bedtools closest -s -d -a human_rnaseq_bwa.mapped.q1.merge.sort.bed -b gencode.v19.proteincoding.genes.sort.
merge.final > hg19_rnaseq_protcode_closest.bed

wc -l hg19_rnaseq_protcode_closest.bed
#7134 hg19_rnaseq_protcode_closest.bed  #same length as the original file

more hg19_rnaseq_protcode_closest.bed
#chr1    880458    880529    1    37    +    chr1    860260    879955    SAMD11    .    +    504
#chr1    881549    881650    1    37    -    chr1    879584    894689    NOC2L    .    -    0
#chr1    887884    887985    1    37    +    chr1    860260    879955    SAMD11    .    +    7930
#chr1    892309    892410    1    37    -    chr1    879584    894689    NOC2L    .    -    0
#chr1    892475    892576    1    23    +    chr1    895967    901095    KLHL17    .    +    3392

#sort by the distance to a gene, longest distances first
sort -k13,13nr hg19_rnaseq_protcode_closest.bed | more

#sort by the distance to a gene, shortest distances first
sort -k13,13n hg19_rnaseq_protcode_closest.bed | more
```

This is a nice way to examine your reads over annotated protein-coding genes. Note the strand specificity - only reads on the correct strand will be reported when there is a + strand gene and a - strand gene over the same location.

bedtools subtract: removing features from your bed file

Bedtools subtract takes an A file and a B file, then searches for features in B that overlap A. When/if these features are identified, the overlapping portion is removed from A and the remaining portion of A is reported. If a feature in B overlaps all of a feature in A, that feature will not be reported.

bedtools subtract options

```
bedtools subtract [OPTIONS] -a <BED/GFF/VCF> -b <BED/GFF/VCF>
```

Note that **bedtools subtract** is performed on two files, and unlike some of the other utilities we've used, you can't use multiple B features here. However, you can use **cat** to join together features you'd like to subtract from your A file.

- f: Minimum overlap required as a fraction of A. Default is 1E-9 (i.e. 1bp).
- F: Minimum overlap required as a fraction of B. Default is 1E-9 (i.e., 1bp).
- r: Require that the fraction of overlap be reciprocal for A and B. In other words, if -f is 0.90 and -r is used, this requires that B overlap at least 90% of A and that A also overlaps at least 90% of B.

- e: Require that the minimum fraction be satisfied for A _OR_ B. In other words, if -e is used with -f 0.90 and -F 0.10 this requires that either 90% of A is covered OR 10% of B is covered. Without -e, both fractions would have to be satisfied.**-s** Force “strandedness”. That is, only report hits in B that overlap A on the same strand. By default, overlaps are reported without respect to strand.
- S: Require different strandedness. That is, only report hits in B that overlap A on the _opposite_ strand. By default, overlaps are reported without respect to strand
- A: Remove entire feature if any overlap. That is, by default, only subtract the portion of A that overlaps B. Here, if any overlap is found (or `-f` amount), the entire feature is removed.
- N: Same as -A except when used with -f, the amount is the sum of all features (not any single feature)

Let's do a little set-up for the next exercise:

copy some gencode files over

```
cd $SCRATCH/core_ngs
mkdir subtract
cd subtract
cp /scratch/01786/awh394/core_ngs.test/closest/gencode.v19.proteincoding.genes.sort.merge.final .
cp /scratch/01786/awh394/core_ngs.test/closest/gencode.v19.genes.sort.merge.final .
```

Exercise 7: remove the protein-coding genes from a gencode list of genes using subtract, then give a count of the non-protein-coding gene entries

This allows you to identify which gene regions are not protein coding, and are likely pseudogenes, but could also be miRNAs, snRNAs or other genes that aren't translated into a peptide sequence.

My output is commented in this code block.

```
cd subtract
module load bedtools #if you haven't loaded it up yet this session
bedtools subtract -a gencode.v19.genes.sort.merge.final -b gencode.v19.proteincoding.genes.sort.merge.final >
gencode.v19.not.proteincoding.genes.bed

wc -l gencode.v19.not.proteincoding.genes.bed
#23483 gencode.v19.not.proteincoding.genes.bed

more gencode.v19.not.proteincoding.genes.bed
#chr1    11869    14412    DDX11L1    .    +
#chr1    14363    29806    WASH7P    .    -
#chr1    29554    31109    MIR1302-11    .    +
#chr1    34554    36081    FAM138A    .    -
#chr1    52473    54936    OR4G4P    .    +
#chr1    62948    63887    OR4G11P    .    +
```

While the above example is not super useful in all cases, one might use the above workflow to remove genes that aren't of interest from a larger set.

A little bit of filtering, using awk

As a final note, yesterday we taught you about using a lot of unix utilities, including **uniq**, **sort** and **cut**. One last utility I'd like to add, that is very useful for manipulating these types of tab delimited files, is **awk**. **Awk** isn't a command, but rather a little text manipulation language in it's own right (which we briefly used above to rearrange the columns in a file). While **awk** can be used to do many different things, here we'll primarily use it to sort tab delimited files based on the values present in those files. That is useful to filter your files for entries on a given chromosome, or greater than/less than a given score. If your dataset is large, this type of filtering can be invaluable! Below is an example of a simple **awk** script:

a simple awk script

```
cat file.bed | awk 'BEGIN{FS="\t";OFS="\t";}{if ($6 == '+'){print}}' > file.plusStrand.bed
```

1. In the first section, we open the bed file of interest. Then we pipe that filestream to the awk program.
2. The section: BEGIN{FS="\t";OFS="\t";} tells awk to begin a filter, the input file is tab delimited, and the output file is also tab delimited.
    a. Generally, you can leave this section constant (if you are working with tab delimited files).
3. The second section: {if ($6 == '+'){print}} is our selection and printing criteria.
    a. "$6" indicates column 6, and == indicates "equals" or "matches".
4. The {print} command tells awk to print the whole line if the statement for column 6 evaluates to true.
5. Thus, the output file only contains those lines which satisfy the criteria in the selection statement.

We can do this filtering on the `hg19_rnaseq_mirnaseq_intersect.bed` file we just created using **bedtools intersect**.

```
cd $SCRATCH/core_ngs/intersect/
cat hg19_rnaseq_mirnaseq_intersect.bed | awk 'BEGIN{FS="\t";OFS="\t";}{if ($6 == "+"){print}}' | more
```

You could also insist on columns 6 and 12 both being the plus strand as such:

```
cd $SCRATCH/core_ngs/intersect/
cat hg19_rnaseq_mirnaseq_intersect.bed | awk 'BEGIN{FS="\t";OFS="\t";}{if ($6 == "+" && $12 == "+"){print}}' |
more
```