

Analysis using BEDTools



Reservations

Use our summer school *reservation* (**CoreNGS-Fri**) when submitting batch jobs to get higher priority on the **ls6** normal queue **today**:

```
sbatch --reservation=CoreNGS-Fri <batch_file>.slurm
idev -m 180 -N 1 -A OTH21164 -r CoreNGS-Fri
```

- [The BED format](#)
- [BEDTools overview](#)
 - [BEDTools versions](#)
 - [Input format considerations](#)
 - [About strandedness](#)
- [About GFF/GTF annotation files](#)
 - [Filter annotations based on desired feature type](#)
 - [Convert GFF/GTF format to BED with ID in the name field](#)
- [Exercises](#)
 - [Use bedtools merge to collapse overlapping annotations](#)
 - [Use bedtools multicov to count feature overlaps](#)
 - [Use bedtools genomecov to create a signal track](#)

The BED format

BED (Browser Extensible Data) format is a simple **text format for location-oriented data** (genomic regions) developed to support UCSC Genome Browser tracks. Standard **BED** files have 3 to 6 **Tab**-separated columns, although up to 12 columns are defined. (Read more about [the UCSC Genome Browser's official BED format](#).)



Memorize the 6 main BED fields

These 6 **BED** fields are **so important** that you should memorize them. Keep repeating "**chrom, start, end, name, score, strand**" until the words trip off your tongue 😊

1. **chrom** (required) – string naming the **chromosome or other contig**
2. **start** (required) – the **0-based** start position of the region
3. **end** (required) – the **1-based** end position of the region
4. **name** (optional) – an **arbitrary string** describing the region
 - for **BED** files loaded as UCSC Genome Browser tracks, this text is displayed above the region
5. **score** (optional) – an **integer score** for the region
 - for **BED** files to be loaded as UCSC Genome Browser tracks, this should be a number between 0 and 1000, higher = "better"
 - for non-GenBrowse **BED** files, this can be any integer value (e.g. the length of the region)
6. **strand** (optional) – a single character describing the region's strand
 - **+** – **plus strand** (Watson strand) region
 - **-** – **minus strand** (Crick strand) region
 - **.** – **no strand** – the region is not associated with a strand (e.g. a transcription factor binding region)

Important rules for BED format:

- The number of fields per line must be consistent throughout any single **BED** file
 - e.g. they must all have 3 fields or all have 6 fields
- The first base on a contig is numbered **0**
 - versus **1** for **BAM** file positions
 - so the a **BED start** of 99 is actually the 100th base on the contig
 - but **end** positions are 1-based
 - so a **BED end** of 200 is the 200th base on the contig
 - the length of a **BED** region is **end - start**
 - not **end - start + 1**, as it would be if both coordinates with 0-based or both 1-based
 - this difference is the single greatest source of errors dealing with **BED** files!

Note that the UCSC Genome Browser also defines many BED-like data formats (e.g. **bedGraph**, **narrowPeak**, **tagAlign** and various RNA element formats). See supported [UCSC Genome Browser data formats](#) for more information and examples.

In addition to standard-format **BED** files, one can create **custom BED** files that have at least 3 of the standard fields (**chrom, start, end**), followed by any number of custom fields. For example:

- A **BED3+** file contains the 3 required **BED** fields, followed by some number of user-defined columns (all records with the same number)
- A **BED6+** file contains the 3 required **BED** fields, 3 additional standard BED fields (**name, score, strand**), followed by some number of user-defined columns

As we will see, **BEDTools** functions require **BED3+** input files, or **BED6+** if **strand-specific** operations are requested.

BEDTools overview

The [BEDTools suite](#) is a set of utilities for manipulating **BED** and **BAM** files. We call it the "Swiss army knife" for genomic region analyses because its sub-commands are so numerous and versatile. Some of the most common **bedtools** operations perform set-theory functions on regions: intersection (**intersect**), union (**merge**), set difference (**subtract**) – but there are many others. The table below lists some of the most useful sub-commands along with applicable use cases.

Sub-command	Description	Use case(s)
bamtobed	Convert BAM files to BED format.	You want to have the contig , start , end , and strand information for each mapped alignment record in separate fields. Recall that the strand is encoded in a BAM flag (0x10) and the exact end coordinate requires parsing the CIGAR string.
bamtofastq	Extract FASTQ sequences from BAM alignment records.	You have downloaded a BAM file from a public database, but it was not aligned against the reference version you want to use (e.g. it is hg19 and you want an hg38 alignment). To re-process, you need to start with the original FASTQ sequences.
getfasta	Get FASTA entries corresponding to regions.	You want to run motif analysis , which requires the original FASTA sequences, on a set of regions of interest. In addition to the BAM file, you must provide FASTA file(s) for the genome/reference used for alignment (e.g. the FASTA file used to build the aligner index).
genomecov	Generate per-base genome-wide signal trace	Produce a per-base genome-wide signal (in bedGraph format), for example for a ChIP-seq or ATAC-seq experiment. After conversion to binary bigWig format, such tracks can be visualized in the Broad's IGV (Integrative Genome Browser) application, or configured in the UCSC Genome Browser as custom tracks.
coverage	Compute coverage of your regions	- You have performed a WGS (whole genome sequencing) experiment and want to know if has resulted in the desired coverage depth. - Calculate what proportion of the (known) transcriptome is covered by your RNA-seq alignments. Provide the transcript regions as a BED or GFF/GTF file.
multicov	Count overlaps between one or more BAM files and a set of regions of interest.	Count RNA-seq alignments that overlap a set of genes of interest. While this task is usually done with a specialized RNA-seq quantification tool (e.g. featureCounts or HTSeq), bedtools multicov can provide a quick estimate, e.g. for QC purposes.
merge	Combine a set of possibly-overlapping regions into a single set of non-overlapping regions.	Collapse overlapping gene annotations into per-strand non-overlapping regions before counting (e.g with featureCounts or HTSeq). If this is not done, the source regions will potentially be counted multiple times, once for each (overlapping) target region it intersects.
subtract	Remove unwanted regions.	Remove rRNA gene regions from a merged gene annotations file before counting.
intersect	Determine the overlap between two sets of regions.	Similar to multicov , but can also report the overlapping regions, not just count them.
closest	Find the genomic features nearest to a set of regions.	For a set of significant ChIP-seq transcription factor (TF) binding regions (" peaks ") that have been identified, determine nearby genes that may be targets of TF regulation.

We will explore a few of these functions in our exercises.

BEDTools versions

BEDTools is under active development and is always being refined and extended. Unfortunately, sometimes changes are made that are incompatible with previous **BEDTools** versions. For example, a major change to the way **bedtool merge** functions was made after **bedtools v2.17.0**.

So it is important to know which version of **BEDTools** you are using, and read the documentation carefully to see if changes have been made since your version.

Login to **Is6**, start and **idev** session, then load the **BioContainers bedtools** module, and check its version.

Start an idev session

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGS-Fri
# or
idev -m 90 -N 1 -A OTH21164 -p development

module load biocontainers
module load bedtools
bedtools --version # should be bedtools v2.27.1
```

Input format considerations

- Most **BEDTools** functions now accept either **BAM** or **BED** files as input.
 - **BED** format files must be **BED3+**, or **BED6+** if strand-specific operations are requested.
- When comparing against a set of regions, those regions are usually supplied in either **BED** or **GTF/GFF**.
- All text-format input files (**BED**, **GTF/GFF**, **VCF**) should use **Unix line endings (linefeed)** only).

The most important thing to remember about comparing regions using **BEDTools**, is that **all input files must share the same set of contig names** and be based on the same reference! For example, if an alignment was performed against a human **GRCh38** reference genome from **Gencode**, use annotations from the corresponding **GFF/GTF** annotations.

About strandedness

By default many **bedtools** utilities that perform overlapping, consider reads overlapping the feature on **either strand**, but can be made **strand-specific** with the **-s** or **-S** option. This **strandedness** options for **bedtools** utilities refers the orientation of the **R1** read with respect to the feature's (gene's) strand.

- **-s** says the **R1** read is **sense** stranded (on the **same** strand as the gene).
- **-S** says the **R1** read is **antisense** stranded (the **opposite** strand as the gene).

RNA-seq libraries can be constructed with 3 types of **strandedness**:

1. **sense stranded** – the **R1** read should be on the **same strand** as the gene.
2. **antisense stranded** – the **R1** read should be on the **opposite strand** as the gene.
3. **unstranded** – the **R1** could be on **either** strand

Which type of RNA-seq library you have depends on the library preparation method – so ask your sequencing center! Our yeast RNA-seq library is **sense stranded** (note that most RNA-seq libraries prepared by GSAF are **antisense stranded**).

If you have a stranded RNA-seq library, you should use either **-s** or **-S** to avoid false counting against a gene on the wrong strand.

About GFF/GTF annotation files

Annotation files that you retrieve from public databases are often in **GTF** (Gene Transfer Format) or one of the in **GFF** (General Feature Format) formats (usually **GFF3** these days).

Unfortunately, both formats are obscure and hard to work with directly. While **bedtools** does accept annotation files in **GFF/GTF** format, you will not like the results. This is because the most useful information in a **GFF/GTF** file is in a loosely-structured **attributes** field.

Also unfortunately, there are a number of variations of both annotation formats However both **GTF** and **GFF** share the first 8 **Tab**-separated fields:

1. **seqname** - The name of the chromosome or contig.
2. **source** - Name of the program that generated this feature, or other data source (e.g. database)
3. **feature_type** - Type of the feature, for example:
 - **CDS** (coding sequence), **exon**
 - **gene**, **transcript**
 - **start_codon**, **stop_codon**
4. **start** - Start position of the feature, with sequence numbering starting at 1.
5. **end** - End position of the feature, with sequence numbering starting at 1.
6. **score** - A numeric value. Often but not always an integer.
7. **strand** - Defined as **+** (forward), **-** (reverse), or **.** (no relevant strand)
8. **frame** - For a **CDS**, one of 0, 1 or 2, specifying the reading frame of the first base; otherwise '.'

The **Tab**-separated columns will care about are (1) **seqname**, (3) **feature_type** and (4,5) **start**, **end**. The reason we care is that when working with annotations, we usually only want to look at annotations of a particular type, most commonly **gene**, but also **transcript** or **exon**.

So where is the real annotation information, such as the unique gene ID or gene name? Both formats also have a **9th** field, which is usually populated by a set of **name/value** pair **attributes**, and that's where the useful information is (e.g. the unique feature identifier, name, and so forth).

Take a quick look at a yeast annotation file, **sacCer_R64-1-1_20110208.gff** using **less**.

Start an idev session

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGS-Fri
# or
idev -m 90 -N 1 -A OTH21164 -p development

module load biocontainers
module load bedtools
bedtools --version # should be bedtools v2.27.1
```

Look at GFF annotation entries with less

```
mkdir -p $SCRATCH/core_ngs/bedtools
cd $SCRATCH/core_ngs/bedtools
cp $CORENGS/yeast_rnaseq/yeast_mrna.sort.filt.bam* .
cp $CORENGS/catchup/references/gff/sacCer3.R64-1-1_20110208.gff .

# Use the less pager to look at multiple lines
less sacCer3.R64-1-1_20110208.gff

# Look at just the most-important Tab-separated columns
cat sacCer3.R64-1-1_20110208.gff | grep -v '#' | cut -f 1,3-5 | head -20

# Include the ugly 9th column where attributes are stored
cat sacCer3.R64-1-1_20110208.gff | grep -v '#' | cut -f 1,3,9 | head
```

In addition to comment lines (starting with #), you can see the **chr1** contig names in column 1 and various feature types in column 3. You also see tags like **Name=YAL067C;gene=SEO1**; among the *attributes* on some records, but in general the *attributes* column information is *really ugly*.

To summarize, we have two problems to solve:

1. We only care about a subset of feature types (here genes), and
2. We want the important annotation information – gene names and IDs – to appear as regular columns instead of weird name/value pairs.

Filter annotations based on desired feature type

One of the first things you want to know about your annotation file is what gene features it contains. Here's how to find that: (Read more about what's going on here at [piping a histogram](#))

```
mkdir -p $SCRATCH/core_ngs/bedtools
cd $SCRATCH/core_ngs/bedtools
cp $CORENGS/catchup/references/gff/sacCer3.R64-1-1_20110208.gff .
```

Create a histogram of all the feature types in a GFF

```
cd $SCRATCH/core_ngs/bedtools
cat sacCer3.R64-1-1_20110208.gff | grep -v '^#' | cut -f 3 | \
  sort | uniq -c | sort -k1,1nr | more
```

You should see something like this.

Histogram of yeast annotation features

```
7077 CDS
6607 gene
480 noncoding_exon
383 long_terminal_repeat
376 intron
337 ARS
299 tRNA
190 region
129 repeat_region
102 nucleotide_match
89 transposable_element_gene
77 snoRNA
50 LTR_retrotransposon
32 telomere
31 binding_site
27 rRNA
24 five_prime_UTR_intron
21 pseudogene
17 chromosome
16 centromere
15 ncRNA
8 external_transcribed_spacer_region
8 internal_transcribed_spacer_region
6 snRNA
3 gene_cassette
2 insertion
```

Let's create a file that contains only the 6607 *gene* entries:

Filter GFF gene feature with awk

```
cat sacCer3.R64-1-1_20110208.gff | grep -v '#' | \
  awk 'BEGIN{FS=OFS="\t"}{ if($3=="gene"){print} }' \
  > sc_genes.gff
wc -l sc_genes.gff
```

The line count of `sc_genes.gff` should be 6607 – one for each gene entry.

Convert GFF/GTF format to BED with ID in the name field

Our `sc_genes.gff` annotation subset now contains only the 6607 genes in the *Saccharomyces cerevisiae* genome. This addresses our first problem, but entries in this file still have the important information – the gene ID and name – in the loosely-structured 9th *attributes* field.

If we want to associate reads with features, we need to have the feature names where they are easy to extract!

What most folks do is find some way to convert their **GFF/GTF** file to a **BED** file, parsing out some (or all) of the name/value attribute pairs into **BED** file columns after the standard 6. You can find such conversion programs on the web – or write one yourself. Or you could use the BiolTeam conversion script, [/work/projects/BiolTeam/common/script/gtf_to_bed.pl](#). While it will not work 100% of the time, it manages to do a decent job on most **GFF/GTF** files. And it's pretty easy to run.



Let Anna know if you run into problems

If this script doesn't work on your annotation file, please let Anna know. She is always looking for cases where the conversion fails, and will try to fix it.

Here we just give the script the **GFF** file to convert, plus a **1** that tells it to *URL decode* weird looking text (e.g. our *Note* attribute values).

```
mkdir -p $SCRATCH/core_ngs/bedtools
cd $SCRATCH/core_ngs/bedtools
cp $CORENGS/catchup/references/gff/sacCer3.R64-1-1_20110208.gff .
```

Convert GFF to BED with BioTeam script

```
/work/projects/BioITeam/common/script/gtf_to_bed.pl sc_genes.gff 1 \  
> sc_genes.converted.bed
```

The program reads the input file twice – once to gather all the attribute names, and then a second time to write the attribute values in well-defined columns. You'll see output like this:

```
-----  
Gathering all attribute names for GTF 'sc_genes.gff'...  
  urlDecode = 1, tagAttr = tag  
Done!  
  6607 lines read  
  6607 locus entries  
  8 attributes found:  
(Alias ID Name Note Ontology_term dbxref gene orf_classification)  
-----  
Writing BED output for GTF 'sc_genes.gff'...  
Done! Wrote 6607 locus entries from 6607 lines
```

To find out what the resulting columns are, look at the header line out the output **BED** file:

```
head -1 sc_genes.converted.bed
```

For me the resulting 16 attributes are as follows (they may have a different order for you). I've numbered them below for convenience.

Converted BED attributes

1. chrom	2. start	3. end	4. featureType	5. length	6. strand
7. source	8. frame	9. Alias	10. ID	11. Name	12. Note
13. Ontology_term	14. dbxref	15. gene	16. orf_classification		

The final transformation is to do a bit of re-ordering, dropping some fields. We'll do this with [awk](#), because [cut](#) can't re-order fields. While this is not strictly required, it can be helpful to have the critical fields (including the gene ID) in the 1st 6 columns. We do this separately for the header line and the rest of the file so that the BED file we give [bedtools](#) does not have a header (but we know what those fields are). We would normally preserve valuable annotation information such as [Ontology_term](#), [dbxref](#) and [Note](#), but drop them here for simplicity.

```
mkdir -p $SCRATCH/core_ngs/bedtools  
cd $SCRATCH/core_ngs/bedtools  
cp $CORENGS/catchup/bedtools_merge/*.gff .  
cp $CORENGS/catchup/bedtools_merge/sc_genes.converted.bed
```

Re-order the final BED fields

```
head -1 sc_genes.converted.bed | sed 's/\r//' | awk '  
BEGIN{FS=OFS="\t"}{print $1,$2,$3,$10,$5,$6,$15,$16}  
' > sc_genes.bed.hdr  
  
tail -n +2 sc_genes.converted.bed | sed 's/\r//' | awk '  
BEGIN{FS=OFS="\t"}  
{ if($15 == "") {$15 = $10} # make sure gene name is populated  
  print $1,$2,$3,$10,$5,$6,$15,$16}  
' > sc_genes.bed
```

One final detail. Annotation files you download may have non-Unix (*linefeed*-only) line endings. Specifically, they may use Windows line endings (*carriage return + linefeed*). (Read about [Line ending nightmares](#).) The expression `sed 's/\r/'` uses the [sed](#) (substitution editor) tool to replace *carriage return* characters (`\r`) with nothing, removing them from the output.

Finally, the 8 re-ordered attributes are:

Re-ordered BED attributes

1. chrom 2. start 3. end 4. ID 5. length 6. strand
7. gene 8. orf_classification

****Whew**!** That was a lot of work. Welcome to the world of annotation wrangling – it's never pretty! But at least the result is much nicer looking. Examine the results using [more](#) or [less](#) or [head](#):

Examine our BED-format annotations

```
cat sc_genes.bed | head -20
```

Doesn't this look better? (I've tidied up the output a bit below.)

chrI	334	649	YAL069W	315	+	YAL069W	Dubious
chrI	537	792	YAL068W-A	255	+	YAL068W-A	Dubious
chrI	1806	2169	YAL068C	363	-	PAU8	Verified
chrI	2479	2707	YAL067W-A	228	+	YAL067W-A	Uncharacterized
chrI	7234	9016	YAL067C	1782	-	SE01	Verified
chrI	10090	10399	YAL066W	309	+	YAL066W	Dubious
chrI	11564	11951	YAL065C	387	-	YAL065C	Uncharacterized
chrI	12045	12426	YAL064W-B	381	+	YAL064W-B	Uncharacterized
chrI	13362	13743	YAL064C-A	381	-	YAL064C-A	Uncharacterized
chrI	21565	21850	YAL064W	285	+	YAL064W	Verified
chrI	22394	22685	YAL063C-A	291	-	YAL063C-A	Uncharacterized
chrI	23999	27968	YAL063C	3969	-	FL09	Verified
chrI	31566	32940	YAL062W	1374	+	GDH3	Verified
chrI	33447	34701	YAL061W	1254	+	BDH2	Uncharacterized
chrI	35154	36303	YAL060W	1149	+	BDH1	Verified
chrI	36495	36918	YAL059C-A	423	-	YAL059C-A	Dubious
chrI	36508	37147	YAL059W	639	+	ECM1	Verified
chrI	37463	38972	YAL058W	1509	+	CNE1	Verified
chrI	38695	39046	YAL056C-A	351	-	YAL056C-A	Dubious
chrI	39258	41901	YAL056W	2643	+	GPB2	Verified

Note that value in the 8th column. In the yeast annotations from **SGD** there are 3 gene classifications: **Verified**, **Uncharacterized** and **Dubious**. The **Dubious** ones have no experimental evidence so are generally excluded.

```
mkdir -p $SCRATCH/core_ngs/bedtools
cd $SCRATCH/core_ngs/bedtools
cp $CORENGS/catchup/bedtools_merge/*.gff .
cp $CORENGS/catchup/bedtools_merge/sc_genes* .
```

Exercise: How many genes in our sc_genes.bed file are in each category?

Use **cut** to isolate that field, **sort** to sort the resulting values into blocks, then **uniq -c** to count the members of each block.

```
cut -f 8 sc_genes.bed | sort | uniq -c
```

You should see this:

```
810 Dubious
897 Uncharacterized
4896 Verified
4 Verified|silenced_gene
```

If you want to further order this output listing the most abundant category first, add another **sort** statement:

```
cut -f 8 sc_genes.bed | sort | uniq -c | sort -k1,1nr
```

The `-k 1,1nr` options says to sort on the **1st** field (*whitespace* delimited) of input, using **numeric** sorting, in **reverse** order (i.e., largest first). Which produces:

```
4896 Verified
897 Uncharacterized
809 Dubious
4 Verified|silenced_gene
```

Exercises

Use bedtools merge to collapse overlapping annotations

One issue that often arises when dealing with **BED** regions is that they can overlap one another. For example, on the yeast genome, which has very few non-coding areas, there are some overlapping **ORFs** (Open Reading Frames), especially **Dubious** ORFs that overlap **Verified** or **Uncharacterized** ones. When **bedtools** looks for overlaps, it will count a read that overlaps **any** of those overlapping **ORFs** – so some reads can be counted twice.

One way to avoid this double-counting is to collapse the overlapping regions into a **merged** set of **non-overlapping regions** – and that's what the **bedtools merge** utility does (<http://bedtools.readthedocs.io/en/latest/content/tools/merge.html>).

Here we're going to use **bedtools merge** to collapse our gene annotations into a non-overlapping set, first for all genes, then for only non-**Dubious** genes.

The output from **bedtools merge** always starts with 3 columns: **chrom**, **start** and **end** of the merged region only.

Using the **-c** (column) and **-o** (operation) options, you can have information added in subsequent fields. Each comma-separated column number following **-c** specifies a **column** to operate on, and the corresponding comma-separated function name following the **-o** specifies the **operation** to perform on that column in order to produce an additional output field.

For example, our **sc_genes.bed** file has a gene name in column 4, and for each (possibly merged) gene region, we want to know the **number** of gene regions that were collapsed into the region, and also **which** gene names were collapsed.

We can do this with **-c 6,4,4 -o distinct,count,collapse**, which says that three custom output columns should be added:

- the 1st custom column should result from collapsing distinct (unique) values of gene file column 6 (the strand, **+** or **-**)
 - since we will ask for stranded merging, the merged regions will always be on the same strand, so this value will always be **+** or **-**
- the 2nd custom output column should result from **counting** the gene names in column 4 for all genes that were merged, and
- the 3rd custom output should be a comma-separated **collapse** list of those same column 4 gene names

bedtools merge also requires that the input **BED** file be sorted by locus (**chrom + start**), so we do that first, then we request a strand-specific merge (**-s**):

```
mkdir -p $SCRATCH/core_ngs/bedtools
cd $SCRATCH/core_ngs/bedtools
cp $CORENGS/yeast_rnaseq/*.gff .
cp $CORENGS/yeast_rnaseq/sc_genes.bed* .
cp $CORENGS/yeast_rnaseq/yeast_mrna.sort.filt.bam* .
module load biocontainers
module load bedtools
```

Use bedtools merge to collapse overlapping gene annotations

```
cd $SCRATCH/core_ngs/bedtools
sort -k1,1 -k2,2n sc_genes.bed > sc_genes.sorted.bed
bedtools merge -i sc_genes.sorted.bed -s -c 6,4,4 -o distinct,count,collapse > merged.sc_genes.txt
```

The first few lines of the **merged.sc_genes.txt** file look like this (I've tidied it up a bit):

2-micron	251	1523	+	1	R0010W
2-micron	1886	3008	-	1	R0020C
2-micron	3270	3816	+	1	R0030W
2-micron	5307	6198	-	1	R0040C
chrI	334	792	+	2	YAL069W, YAL068W-A
chrI	1806	2169	-	1	YAL068C
chrI	2479	2707	+	1	YAL067W-A
chrI	7234	9016	-	1	YAL067C
chrI	10090	10399	+	1	YAL066W
chrI	11564	11951	-	1	YAL065C

Output column 4 has the region's *strand*. Column 5 is the count of merged regions, and column 6 is a comma-separated list of the merged gene names.

Exercise: Compare the number of regions in the merged and before-merge gene files.

```
wc -l sc_genes.bed merged.sc_genes.txt
```

There were 6607 genes before merging and 6485 after.

Exercise: How many regions represent only 1 gene, 2 genes, or more?

Output column 5 has the gene count.

```
cut -f 5 merged.sc_genes.txt | sort | uniq -c | sort -k2,2n
```

Produces this histogram:

```
6374 1
105 2
4 3
1 4
1 7
```

There are 111 regions (105 + 4 + 1 + 1) where more than one gene contributed.

Exercise: Repeat the steps above, but first create a good.sc_genes.bed file that does not include Dubious ORFs.

```
cd $SCRATCH/core_ngs/bedtools
grep -v 'Dubious' sc_genes.bed > good.sc_genes.bed

sort -k1,1 -k2,2n good.sc_genes.bed > good.sc_genes.sorted.bed
bedtools merge -i good.sc_genes.sorted.bed -s \
  -c 6,4,4 -o distinct,count,collapse > merged.good.sc_genes.txt

wc -l good.sc_genes.bed merged.good.sc_genes.txt
```

There were 5797 "good" (non-*Dubious*) genes before merging and 5770 after.

```
cut -f 5 merged.good.sc_genes.txt | sort | uniq -c | sort -k2,2n
```

Produces this histogram:

```
5750 1
18 2
1 4
1 7
```

Now there are only 20 regions where more than one gene was collapsed. Clearly eliminating the *Dubious* ORFs helped.

So there's one more thing we need to do to create a valid **BED** format file. Our `merged.good.sc_genes.txt` columns are *chrom*, *start*, *end*, *strand*, *merged_region_count*, *merged_region(s)*, but the **BED6** specification is: *chrom*, *start*, *end*, *name*, *score*, *strand*.

To make a valid **BED6** file, we'll include the first 3 output columns of `merged.good.sc_genes.txt` (*chrom*, *start*, *end*), but if *strand* is to be included, it should be in column 6. Column 4 should be *name* (we'll put the collapsed gene name list there), and column 5 a *score* (we'll put the region count there).

We can use `awk` to re-order the fields:

```
cat merged.good.sc_genes.txt | awk '
BEGIN{FS=OFS="\t"}
{print $1,$2,$3,$6,$5,$4}' > merged.good.sc_genes.bed
```

Use bedtools multicov to count feature overlaps

We're now (finally!) actually going to do some gene-based analyses of a yeast RNA-seq dataset using [bedtools](#) and the **BED**-formatted, merged yeast gene annotation file we created above.

In this section we'll use [bedtools multicov](#) to count RNA-seq reads that overlap our gene features. The [bedtools multicov](#) command (<http://bedtools.readthedocs.io/en/latest/content/tools/multicov.html>) takes a feature file (**GFF/BED/VCF**) and counts how many reads from one or more input **BAM** files overlap those features. The input **BAM** file(s) must be position-sorted and indexed.

Make sure you're in an [idev](#) session, since we will be doing some significant computation, and make [bedtools](#) and [samtools](#) available.

Start an idev session

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGS-Fri
# or
idev -m 90 -N 1 -A OTH21164 -p development
```

Copy over the yeast RNA-seq files we'll need (also copy the **GFF** gene annotation file if you didn't make one).

Setup for BEDTools multicov

```
# Get the merged yeast genes bed file if you didn't create one
mkdir -p $SCRATCH/core_ngs/bedtools_multicov
cd $SCRATCH/core_ngs/bedtools_multicov
cp $CORENGS/catchup/bedtools_merge/merged*bed .

# Copy the BAM file
cd $SCRATCH/core_ngs/bedtools_multicov
cp $CORENGS/yeast_rnaseq/yeast_mrna.sort.filt.bam* .
```

Exercises: How many reads are represented in the `yeast_mrna.sort.filt.bam` file? How many mapped? How many proper pairs? How many duplicates? What is the distribution of mapping qualities? What is the average mapping quality?

[samtools flagstat](#) for the different read counts.

[samtools view](#) + [cut](#) + [sort](#) + [uniq -c](#) for mapping quality distribution

[samtools view](#) + [awk](#) for average mapping quality

```
cd $SCRATCH/core_ngs/bedtools_multicov
samtools flagstat yeast_mrna.sort.filt.bam | tee yeast_mrna.flagstat.txt
```

samtools flagstat output

```
3347559 + 0 in total (QC-passed reads + QC-failed reads)
24317 + 0 secondary
0 + 0 supplementary
922114 + 0 duplicates
3347559 + 0 mapped (100.00% : N/A)
3323242 + 0 paired in sequencing
1661699 + 0 read1
1661543 + 0 read2
3323242 + 0 properly paired (100.00% : N/A)
3323242 + 0 with itself and mate mapped
0 + 0 singletons (0.00% : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

There are 3323242 total reads, all mapped and all properly paired. So this must be a quality-filtered **BAM**. There are 922114 duplicates, or about 28%.

To get the distribution of mapping qualities:

```
samtools view yeast_mrna.sort.filt.bam | cut -f 5 | sort | uniq -c
```

distribution of mapping qualities

498	20
6504	21
1012	22
355	23
1054	24
2800	25
495	26
14133	27
282	28
358	29
954	30
1244	31
358	32
6143	33
256	34
265	35
1112	36
905	37
309	38
4845	39
5706	40
427	41
1946	42
1552	43
1771	44
6140	45
1771	46
3049	47
3881	48
3264	49
4475	50
15692	51
25378	52
16659	53
18305	54
7108	55
2705	56
59867	57
2884	58
2392	59
3118705	60

To compute average mapping quality:

```
samtools view yeast_mrna.sort.filt.bam | awk '
BEGIN{FS="\t"; sum=0; tot=0}
{sum = sum + $5; tot = tot + 1}
END{printf("mapping quality average: %.1f for %d reads\n", sum/tot,tot) }'
```

Mapping qualities range from 20 to 60 – excellent quality! Because the majority reads have mapping quality 60, the average is 59. So again, there must have been quality filtering performed on upstream alignment records.

Here's how to run **bedtools multicov** in **stranded mode**, directing the **standard output** to a file:

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

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

Run bedtools multicov to count BAM alignments overlapping a set of genes

```
cd $SCRATCH/core_ngs/bedtools_multicov
bedtools multicov -s -bams yeast_mrna.sort.filt.bam \
-bed merged.good.sc_genes.bed > yeast_mrna_gene_counts.bed
```

Exercise: How many records of output were written? Where is the count of overlaps per output record?

```
wc -l yeast_mrna_gene_counts.bed
```

6485 records were written, one for each feature in the [merged.sc_genes.bed](#) file.

The overlap count was added as the last field in each output record (here field 7, since the input annotation file had 6 columns).

Exercise: How many features have non-zero overlap counts?

```
cut -f 7 yeast_mrna_gene_counts.bed | grep -v '^0' | wc -l
# or
cat yeast_mrna_gene_counts.bed | \
awk '{if ($7 > 0) print $7}' | wc -l
```

Most of the genes (6141/6485) have non-zero read overlap counts.

Exercise: What is the total count of reads mapping to gene features?

```
cat yeast_mrna_gene_counts.bed | awk '
BEGIN{FS="\t";sum=0;tot=0}
{if($7 > 0) { sum = sum + $7; tot = tot + 1 }}
END{printf("%d overlapping reads in %d genes\n", sum, tot) }'
```

There are 1,152,831 overlapping reads in 6,141 non-0 gene annotations.

Use bedtools genomecov 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 that shows the **count of overlapping bases** for the regions (see <https://genome.ucsc.edu/goldenpath/help/bedgraph.html>). **bedGraph** files can be visualized in the Broad's **IGV** (Integrative Genomics Viewer) application (<https://software.broadinstitute.org/software/igv/download>) or in the **UCSC Genome Browser** (<https://genome.ucsc.edu/>).

- Go to the **UCSC Genome Browser**: <https://genome.ucsc.edu/>
- Select **Genomes** from the top menu bar
- Select Human from POPULAR SPECIES
 - under **Human Assembly** select **Feb 2009 (GrCh37/hg19)**
 - select GO
- In the hg19 browser page, the **Layered H3K27Ac** track is a **signal track**
 - the x-axis is the genome position
 - the y-axis represents the count of ChIP-seq reads that overlap each position
 - where the ChIP'd protein is **H3K27AC** (histone **H3**, acetylated on the Lysine at amino acid position 27)

The **bedtools genomecov** function (<https://bedtools.readthedocs.io/en/latest/content/tools/coverage.html>), with the **-bg** (**bedgraph**) option produces output in **bedGraph** format. 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 CoreNGS-day5
# or
idev -m 90 -N 1 -A OTH21164 -p development

module load biocontainers
module load bedtools

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

Then calling **bedtools genomecov** is easy. The **-bg** option says to report the depth in bedGraph format.

```
cd $SCRATCH/core_ngs/bedtools_genomecov
bedtools genomecov -bg -ibam yeast_mrna.sort.filt.bam > yeast_mrna.genomecov.bedGraph

wc -l yeast_mrna.genomecov.bedGraph # 1519274 lines
```

The **bedGraph (BED3+)** format has only 4 columns: **chrom start end value** and does not need to include positions with 0 reads. Here the count is the number of reads covering each base in the region given by **chrom start end**, as you can see looking at the first few lines with **head**:

```
chrI    4348    4390    2
chrI    4390    4391    1
chrI    4745    4798    2
chrI    4798    4799    1
chrI    4949    4957    2
chrI    4957    4984    4
chrI    4984    4997    6
chrI    4997    4998    5
chrI    4998    5005    4
chrI    5005    5044    2
chrI    5044    5045    1
chrI    6211    6268    2
chrI    6268    6269    1
chrI    7250    7257    3
chrI    7257    7271    4
chrI    7271    7274    6
chrI    7274    7278    7
chrI    7278    7310    8
chrI    7310    7315    6
chrI    7315    7317    5
```

Because this **bedGraph** file is for the small-ish (12Mb) yeast genome, and for reads that cover only part of that genome, it is not too big – only ~34M. But depending on the species and read depth, **bedGraph** files can get very large, so there is a corresponding binary format called **bigWig** (see <https://genome.ucsc.edu/goldenpath/help/bigWig.html>). The program to convert a **bedGraph** file to **bigWig** format is part of the **UCSC Tools** suite of programs. Look for it with **module spider**, and note that you can get information about all the tools in it using **module spider** with a specific container version:

```
# look for the ucsc tools package
module spider ucsc

# specifying a specific container version will show more information about the package
module spider ucsc_tools/ctr-357--0

# displays information including the programs in the package:
- bedGraphToBigWig
- bedToBigBed
- faToTwoBit
- liftOver
- my_print_defaults
- mysql_config
- nibFrag
- perror
- twoBitToFa
- wigToBigWig
```

Looking at the help for [bedGraphToBigWig](#), we'll need a file of chromosome sizes. We can create one from our **BAM** header, using a [Perl](#) substitution script, which I prefer to [sed](#) (see [Tips and tricks#perlpatternsubstitution](#)):

```
module load ucsc_tools

cd $SCRATCH/core_ngs/bedtools_genomecov
bedGraphToBigWig # look at its usage

# create the needed chromosome sizes file from our BAM header
module load samtools
samtools view -H yeast_mrna.sort.filt.bam | grep -P 'SN[:]' | \
  perl -pe 's/.*SN[:]/' | perl -pe 's/LN[:]/' > sc_chrom_sizes.txt

cat sc_chrom_sizes.txt

# displays:
chrI    230218
chrII   813184
chrIII  316620
chrIV   1531933
chrV    576874
chrVI   270161
chrVII  1090940
chrVIII 562643
chrIX   439888
chrX    745751
chrXI   666816
chrXII  1078177
chrXIII 924431
chrXIV  784333
chrXV   1091291
chrXVI  948066
chrM    85779
```

Finally, call [bedGraphToBigWig](#) after sorting the **bedGraph** file again using the [sort](#) format [bedGraphToBigWig](#) likes. (You can try calling [bedGraphToBigWig](#) without sorting to see the error).

```
cd $SCRATCH/core_ngs/bedtools_genomecov
export LC_COLLATE=C
sort -k1,1 -k2,2n yeast_mrna.genomecov.bedGraph > yeast_mrna.genomecov.sorted.bedGraph
bedGraphToBigWig yeast_mrna.genomecov.sorted.bedGraph sc_chrom_sizes.txt yeast_mrna.genomecov.bw
```

See the size difference between the **bedGraph** and the **bigWig** files. The **bigWig** (9.7M) is less than 1/3 the size of the **bedGraph** (34M).

```
cd $SCRATCH/core_ngs/bedtools_genomecov
ls -lh yeast_mrna.genome*
```

Since the **bigWig** file is binary, not text, you can't use commands like [cat](#), [head](#), [tail](#) on them directly and get meaningful output. Instead, just as [zcat](#) converts [gzip](#)'d files to text, and [samtools view](#) converts binary **BAM** files to text, the [bigWigToBedGraph](#) program can convert binary **bigWig** format to text. That's a different **BioContainers** module ([ucsc-bigwigtoBedGraph](#)) and the default container version doesn't work, so we'll specifically load one that does:

```
# The default version of is broken, so load this specific biocontainers version
module load ucsc-bigwigtoBedGraph/ctr-357--1

# see usage for bigWigToBedGraph:
bigWigToBedGraph

cd $SCRATCH/core_ngs/bedtools_genomecov
# use the program to view a few lines of the binary bigWig file
bigWigToBedGraph yeast_mrna.genomecov.bw stdout | head
```