Bedtools Documentation

Release 2.17.0

Quinlan lab @ UVa

January 27, 2013

CONTENTS

Collectively, the **bedtools** utilities are a swiss-army knife of tools for a wide-range of genomics analysis tasks. The most widely-used tools enable *genome arithmetic*: that is, set theory on the genome. For example, **bedtools** allows one to *intersect*, *merge*, *count*, *complement*, and *shuffle* genomic intervals from multiple files in widely-used genomic file formats such as BAM, BED, GFF/GTF, VCF.

While each individual tool is designed to do a relatively simple task (e.g., *intersect* two interval files), quite sophisticated analyses can be conducted by combining multiple bedtools operations on the UNIX command line.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Overview

1.1.1 Background

The development of bedtools was motivated by a need for fast, flexible tools with which to compare large sets of genomic features. Answering fundamental research questions with existing tools was either too slow or required modifications to the way they reported or computed their results. We were aware of the utilities on the UCSC Genome Browser and Galaxy websites, as well as the elegant tools available as part of Jim Kent's monolithic suite of tools ("Kent source"). However, we found that the web-based tools were too cumbersome when working with large datasets generated by current sequencing technologies. Similarly, we found that the Kent source command line tools often required a local installation of the UCSC Genome Browser. These limitations, combined with the fact that we often wanted an extra option here or there that wasn't available with existing tools, led us to develop our own from scratch. The initial version of bedtools was publicly released in the spring of 2009. The current version has evolved from our research experiences and those of the scientists using the suite over the last year. The bedtools suite enables one to answer common questions of genomic data in a fast and reliable manner. The fact that almost all the utilities accept input from "stdin" allows one to "stream / pipe" several commands together to facilitate more complicated analyses. Also, the tools allow fine control over how output is reported. The initial version of bedtools supported solely 6-column BED files. *However, we have subsequently added support for sequence alignments in* BAM *format, as well as for features in* GFF , *"blocked" BED format, and* VCF *format*. The tools are quite fast and typically finish in a matter of a few seconds, even for large datasets. This manual seeks to describe the behavior and available functionality for each bedtool. Usage examples are scattered throughout the text, and formal examples are provided in the last two sections, we hope that this document will give you a sense of the flexibility of the toolkit and the types of analyses that are possible with bedtools. If you have further questions, please join the bedtools discussion group, visit the Usage Examples on the Google Code site (usage, advanced usage), or take a look at the nascent "Usage From the Wild" page.

1.1.2 Summary of available tools.

bedtools support a wide range of operations for interrogating and manipulating genomic features. The table below summarizes the tools available in the suite.

Utility	Description
annotate	Annotate coverage of features from multiple files.
bamtobed	Convert BAM alignments to BED (& other) formats.
bamtofastq	Convert BAM records to FASTQ records.
bed12tobed6	Breaks BED12 intervals into discrete BED6 intervals.
bedpetobam	Convert BEDPE intervals to BAM records.
bedtobam	Convert intervals to BAM records.

Continued on next page

Table 1.1 – continued from previous page

Utility	Description
closest	Find the closest, potentially non-overlapping interval.
cluster	Cluster (but don't merge) overlapping/nearby intervals.
complement	Extract intervals _not_ represented by an interval file.
coverage	Compute the coverage over defined intervals.
expand	Replicate lines based on lists of values in columns.
flank	Create new intervals from the flanks of existing intervals.
genomecov	Compute the coverage over an entire genome.
getfasta	Use intervals to extract sequences from a FASTA file.
groupby	Group by common cols. & summarize oth. cols. (~ SQL "groupBy")
igv	Create an IGV snapshot batch script.
intersect	Find overlapping intervals in various ways.
jaccard	Calculate the Jaccard statistic b/w two sets of intervals.
links	Create a HTML page of links to UCSC locations.
makewindows	Make interval "windows" across a genome.
map	Apply a function to a column for each overlapping interval.
maskfasta	Use intervals to mask sequences from a FASTA file.
merge	Combine overlapping/nearby intervals into a single interval.
multicov	Counts coverage from multiple BAMs at specific intervals.
multiinter	Identifies common intervals among multiple interval files.
nuc	Profile the nucleotide content of intervals in a FASTA file.
overlap	Computes the amount of overlap from two intervals.
pairtobed	Find pairs that overlap intervals in various ways.
pairtopair	Find pairs that overlap other pairs in various ways.
random	Generate random intervals in a genome.
reldist	Calculate the distribution of relative distances b/w two files.
shuffle	Randomly redistrubute intervals in a genome.
slop	Adjust the size of intervals.
sort	Order the intervals in a file.
subtract	Remove intervals based on overlaps b/w two files.
tag	Tag BAM alignments based on overlaps with interval files.
unionbedg	Combines coverage intervals from multiple BEDGRAPH files.
window	Find overlapping intervals within a window around an interval.

1.1.3 Fundamental concepts.

What are genome features and how are they represented?

Throughout this manual, we will discuss how to use bedtools to manipulate, compare and ask questions of genome "features". Genome features can be functional elements (e.g., genes), genetic polymorphisms (e.g. SNPs, INDELs, or structural variants), or other annotations that have been discovered or curated by genome sequencing groups or genome browser groups. In addition, genome features can be custom annotations that an individual lab or researcher defines (e.g., my novel gene or variant).

The basic characteristics of a genome feature are the chromosome or scaffold on which the feature "resides", the base pair on which the feature starts (i.e. the "start"), the base pair on which feature ends (i.e. the "end"), the strand on which the feature exists (i.e. "+" or "-"), and the name of the feature if one is applicable.

The two most widely used formats for representing genome features are the BED (Browser Extensible Data) and GFF (General Feature Format) formats. bedtools was originally written to work exclusively with genome features described using the BED format, but it has been recently extended to seamlessly work with BED, GFF and VCF files.

Existing annotations for the genomes of many species can be easily downloaded in BED and GFF format from the UCSC Genome Browser's "Table Browser" (http://genome.ucsc.edu/cgi-bin/hgTables?command=start) or from the "Bulk Downloads" page (http://hgdownload.cse.ucsc.edu/downloads.html). In addition, the Ensemble Genome Browser contains annotations in GFF/GTF format for many species (http://www.ensembl.org/info/data/ftp/index.html)

Overlapping / intersecting features.

Two genome features (henceforth referred to as "features") are said to overlap or intersect if they share at least one base in common. In the figure below, Feature A intersects/overlaps Feature B, but it does not intersect/overlap Feature C.

TODO: place figure here

Comparing features in file "A" and file "B".

The previous section briefly introduced a fundamental naming convention used in bedtools. Specifically, all bedtools that compare features contained in two distinct files refer to one file as feature set "A" and the other file as feature set "B". This is mainly in the interest of brevity, but it also has its roots in set theory. As an example, if one wanted to look for SNPs (file A) that overlap with exons (file B), one would use bedtools intersect in the following manner:

```
bedtools intersect -a snps.bed -b exons.bed
```

There are two exceptions to this rule: 1) When the "A" file is in BAM format, the "-abam" option must bed used. For example:

```
bedtools intersect -abam alignedReads.bam -b exons.bed
```

And 2) For tools where only one input feature file is needed, the "-i" option is used. For example:

```
bedtools merge -i repeats.bed
```

BED starts are zero-based and BED ends are one-based.

bedtools users are sometimes confused by the way the start and end of BED features are represented. Specifically, bedtools uses the UCSC Genome Browser's internal database convention of making the start position 0-based and the end position 1-based: (http://genome.ucsc.edu/FAQ/FAQtracks#tracks1) In other words, bedtools interprets the "start" column as being 1 basepair higher than what is represented in the file. For example, the following BED feature represents a single base on chromosome 1; namely, the 1st base:

```
chr1    0         1     first_base
```

Why, you might ask? The advantage of storing features this way is that when computing the length of a feature, one must simply subtract the start from the end. Were the start position 1-based, the calculation would be (slightly) more complex (i.e. (end-start)+1). Thus, storing BED features this way reduces the computational burden.

GFF starts and ends are one-based.

In contrast, the GFF format uses 1-based coordinates for both the start and the end positions. bedtools is aware of this and adjusts the positions accordingly. In other words, you don't need to subtract 1 from the start positions of your GFF features for them to work correctly with bedtools.

VCF coordinates are one-based.

The VCF format uses 1-based coordinates. As in GFF, bedtools is aware of this and adjusts the positions accordingly. In other words, you don't need to subtract 1 from the start positions of your VCF features for them to work correctly with bedtools.

File B is loaded into memory (most of the time).

Whenever a bedtool compares two files of features, the "B" file is loaded into memory. By contrast, the "A" file is processed line by line and compared with the features from B. Therefore to minimize memory usage, one should set the smaller of the two files as the B file. One salient example is the comparison of aligned sequence reads from a current DNA sequencer to gene annotations. In this case, the aligned sequence file (in BED format) may have tens of millions of features (the sequence alignments), while the gene annotation file will have tens of thousands of features. In this case, it is wise to sets the reads as file A and the genes as file B.

Feature files *must* be tab-delimited.

This is rather self-explanatory. While it is possible to allow BED files to be space-delimited, we have decided to require tab delimiters for three reasons:

1. By requiring one delimiter type, the processing time is minimized.
2. Tab-delimited files are more amenable to other UNIX utilities.
3. GFF files can contain spaces within attribute columns. This complicates the use of space-delimited files as spaces must therefore be treated specially depending on the context.

All bedtools allow features to be "piped" via standard input.

In an effort to allow one to combine multiple bedtools and other UNIX utilities into more complicated "pipelines", all bedtools allow features to be passed to them via standard input. Only one feature file may be passed to a bedtool via standard input. The convention used by all bedtools is to set either file A or file B to "stdin" or "-". For example:

```
cat snps.bed | bedtools intersect -a stdin -b exons.bed
cat snps.bed | bedtools intersect -a - -b exons.bed
```

In addition, all bedtools that simply require one main input file (the -i file) will assume that input is coming from standard input if the -i parameter is ignored. For example, the following are equivalent:

```
cat snps.bed | bedtools sort -i stdin
cat snps.bed | bedtools sort
```

Most bedtools write their results to standard output.

To allow one to combine multiple bedtools and other UNIX utilities into more complicated "pipelines", most bedtools report their output to standard output, rather than to a named file. If one wants to write the output to a named file, one can use the UNIX "file redirection" symbol ">" to do so. Writing to standard output (the default):

```
bedtools intersect -a snps.bed -b exons.bed
chr1 100100 100101 rs233454
chr1 200100 200101 rs446788
chr1 300100 300101 rs645678
```

Writing to a file:

```
bedtools intersect -a snps.bed -b exons.bed > snps.in.exons.bed

cat snps.in.exons.bed
chr1 100100 100101 rs233454
chr1 200100 200101 rs446788
chr1 300100 300101 rs645678
```

What is a "genome" file?

Some of the bedtools (e.g., `genomecov`, `complement`, `slop`) need to know the size of the chromosomes for the organism for which your BED files are based. When using the UCSC Genome Browser, Ensemble, or Galaxy, you typically indicate which species / genome build you are working. The way you do this for bedtools is to create a "genome" file, which simply lists the names of the chromosomes (or scaffolds, etc.) and their size (in basepairs). Genome files must be tab-delimited and are structured as follows (this is an example for C. elegans):

```
chrI 15072421
chrII 15279323
...
chrX 17718854
chrM 13794
```

bedtools includes predefined genome files for human and mouse in the /genomes directory included in the bedtools distribution. Additionally, the "chromInfo" files/tables available from the UCSC Genome Browser website are acceptable. For example, one can download the hg19 chromInfo file here: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/chromInfo.txt.gz

Paired-end BED files (BEDPE files).

We have defined a new file format (BEDPE) to concisely describe disjoint genome features, such as structural variations or paired-end sequence alignments. We chose to define a new format because the existing BED block format (i.e. BED12) does not allow inter-chromosomal feature definitions. Moreover, the BED12 format feels rather bloated when one want to describe events with only two blocks.

Use "-h" for help with any bedtool.

Rather straightforward. If you use the "-h" option with any bedtool, a full menu of example usage and available options (when applicable) will be reported.

BED features must not contain negative positions.

bedtools will typically reject BED features that contain negative positions. In special cases, however, BEDPE positions may be set to -1 to indicate that one or more ends of a BEDPE feature is unaligned.

The start position must be <= to the end position.

bedtools will reject BED features where the start position is greater than the end position.

Headers are allowed in GFF and BED files

bedtools will ignore headers at the beginning of BED and GFF files. Valid header lines begin with a "#" symbol, the work "track", or the word "browser". For example, the following examples are valid headers for BED or GFF files:

```
track name=aligned_read description="Illumina aligned reads”
chr5 100000 500000 read1 50 +
chr5 2380000 2386000 read2 60 -

#This is a fascinating dataset
chr5 100000 500000 read1 50 +
chr5 2380000 2386000 read2 60 -

browser position chr22:1-20000
chr5 100000 500000 read1 50 +
chr5 2380000 2386000 read2 60 -
```

GZIP support: BED, GFF, VCF, and BEDPE file can be “gzipped”

bedtools will process gzipped BED, GFF, VCF and BEDPE files in the same manner as uncompressed files. Gzipped files are auto-detected thanks to a helpful contribution from Gordon Assaf.

Support for “split” or “spliced” BAM alignments and “blocked” BED features

As of Version 2.8.0, five bedtools (`intersect`, `coverage`, `genomecob`, `bamToBed`, and `bed12ToBed6`) can properly handle “split”/”spliced” BAM alignments (i.e., having an “N” CIGAR operation) and/or “blocked” BED (aka BED12) features.

`intersect`, `coverage`, and `genomecov` will optionally handle “split” BAM and/or “blocked” BED by using the `-split` option. This will cause intersects or coverage to be computed only for the alignment or feature blocks. In contrast, without this option, the intersects/coverage would be computed for the entire “span” of the alignment or feature, regardless of the size of the gaps between each alignment or feature block. For example, imagine you have a RNA-seq read that originates from the junction of two exons that were spliced together in a mRNA. In the genome, these two exons happen to be 30Kb apart. Thus, when the read is aligned to the reference genome, one portion of the read will align to the first exon, while another portion of the read will align ca. 30Kb downstream to the other exon. The corresponding CIGAR string would be something like (assuming a 76bp read): 30M*3000N*46M. In the genome, this alignment “spans” 3076 bp, yet the nucleotides in the sequencing read only align “cover” 76bp. Without the `-split` option, coverage or overlaps would be reported for the entire 3076bp span of the alignment. However, with the `-split` option, coverage or overlaps will only be reported for the portions of the read that overlap the exons (i.e. 30bp on one exon, and 46bp on the other).

Using the -split option with bamToBed causes “spliced/split” alignments to be reported in BED12 format. Using the -split option with `bed12tobed6` causes “blocked” BED12 features to be reported in BED6 format.

Writing uncompressed BAM output.

When working with a large BAM file using a complex set of tools in a pipe/stream, it is advantageous to pass uncompressed BAM output to each downstream program. This minimizes the amount of time spent compressing and decompressing output from one program to the next. All bedtools that create BAM output (e.g. `intersect`, `window`) will now optionally create uncompressed BAM output using the `-ubam` option.

1.1.4 Implementation and algorithmic notes.

bedtools was implemented in C++ and makes extensive use of data structures and fundamental algorithms from the Standard Template Library (STL). Many of the core algorithms are based upon the genome binning algorithm described in the original UCSC Genome Browser paper (Kent et al, 2002). The tools have been designed to inherit core data structures from central source files, thus allowing rapid tool development and deployment of improvements and corrections. Support for BAM files is made possible through Derek Barnett's elegant C++ API called BamTools.

1.2 Installation

`bedtools` is intended to run in a "command line" environment on UNIX, LINUX and Apple OS X operating systems. Installing `bedtools` involves either downloading the source code and compiling it manually, or installing stable release from package managers such as homebrew (for OS X).

1.2.1 Installing stable releases

Compiling from source via Google Code

Stable, versioned releases of bedtools are made available The following commands will install `bedtools` in a local directory on an UNIX or OS X machine. Note that the **"<version>"** refers to the latest posted version number on http://bedtools.googlecode.com/.

Note: The bedtools Makefiles utilize the GCC compiler. One should edit the Makefiles accordingly if one wants to use a different compiler.

```
$ curl http://bedtools.googlecode.com/files/BEDTools.<version>.tar.gz > BEDTools.tar.gz
$ tar -zxvf BEDTools.tar.gz
$ cd BEDTools-<version>
$ make
```

At this point, one should copy the binaries in ./bin/ to either `usr/local/bin/` or some other repository for commonly used UNIX tools in your environment. You will typically require administrator (e.g. "root" or "sudo") privileges to copy to `usr/local/bin/`. If in doubt, contact you system administrator for help.

Installing with package managers

In addition, stable releases of `bedtools` are also available through package managers such as homebrew (for OS X), `apt-get` and `yum`.

Fedora/Centos. Adam Huffman has created a Red Hat package for bedtools so that one can easily install the latest release using "yum", the Fedora package manager. It should work with Fedora 13, 14 and EPEL5/6 (for Centos, Scientific Linux, etc.).

```
yum install BEDTools
```

Debian/Ubuntu. Charles Plessy also maintains a Debian package for bedtools that is likely to be found in its derivatives like Ubuntu. Many thanks to Charles for doing this.

```
apt-get install bedtools
```

Homebrew. Carlos Borroto has made BEDTools available on the bedtools package manager for OSX.

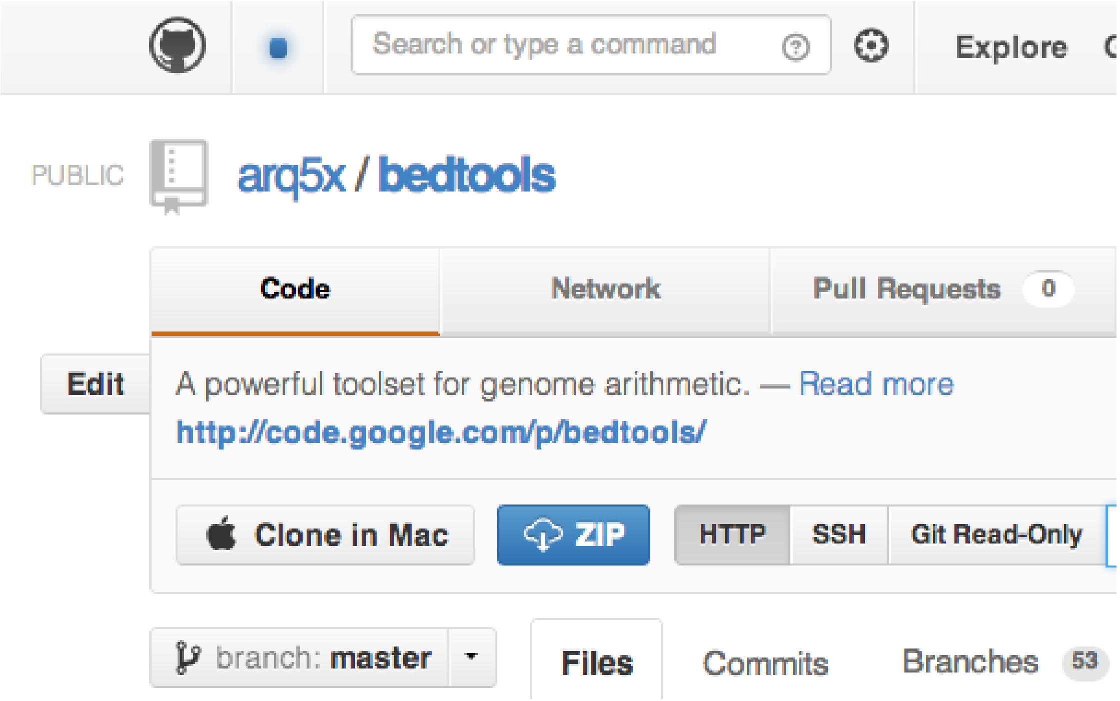
```
brew install bedtools
```

1.2.2 Development versions

The development version of bedtools is maintained in a Github repository. Bug fixes are addressed in this repository prior to release, so there may be situations where you will want to use a development version of bedtools prior to its being promoted to a stable release. One would either clone the repository with `git`, as follows and then compile the source code as describe above:

```
git clone https://github.com/arq5x/bedtools.git
```

or, one can download the source code as a `.zip` file using the Github website. Once the zip file is downloaded and uncompressed with the `unzip` command, one can compile and install using the instructions above.



A powerful toolset for genome arithmetic. — Read more
http://code.google.com/p/bedtools/

1.3 Quick start

1.3.1 Install bedtools

```
curl http://bedtools.googlecode.com/files/BEDTools.<version>.tar.gz > BEDTools.tar.gz
tar -zxvf BEDTools.tar.gz
cd BEDTools
make
sudo cp bin/* /usr/local/bin/
```

1.3.2 Use bedtools

Below are examples of typical bedtools usage. Using the "-h" option with any bedtools will report a list of all command line options.

Report the base-pair overlap between the features in two BED files.

```
bedtools intersect -a reads.bed -b genes.bed
```

Report those entries in A that overlap NO entries in B. Like "grep -v"

```
bedtools intersect  -a reads.bed -b genes.bed
```

Read BED A from STDIN. Useful for stringing together commands. For example, find genes that overlap LINEs but not SINEs.

```
bedtools intersect -a genes.bed -b LINES.bed | \
  bedtools intersect -a stdin -b SINEs.bed
```

Find the closest ALU to each gene.

```
bedtools closest -a genes.bed -b ALUs.bed
```

Merge overlapping repetitive elements into a single entry, returning the number of entries merged.

```
bedtools merge -i repeatMasker.bed -n
```

Merge nearby repetitive elements into a single entry, so long as they are within 1000 bp of one another.

```
bedtools merge -i repeatMasker.bed -d 1000
```

1.4 General usage

1.4.1 Supported file formats

BED format

As described on the UCSC Genome Browser website (see link below), the BED format is a concise and flexible way to represent genomic features and annotations. The BED format description supports up to 12 columns, but only the first 3 are required for the UCSC browser, the Galaxy browser and for bedtools. bedtools allows one to use the "BED12" format (that is, all 12 fields listed below). However, only intersectBed, coverageBed, genomeCoverageBed, and bamToBed will obey the BED12 "blocks" when computing overlaps, etc., via the **"-split"** option. For all other tools, the last six columns are not used for any comparisons by the bedtools. Instead, they will use the entire span (start to end) of the BED12 entry to perform any relevant feature comparisons. The last six columns will be reported in the output of all comparisons.

The file description below is modified from: http://genome.ucsc.edu/FAQ/FAQformat#format1.

1. **chrom** - The name of the chromosome on which the genome feature exists.

- *Any string can be used*. For example, "chr1", "III", "myChrom", "contig1112.23".
- *This column is required.*

2. **start** - The zero-based starting position of the feature in the chromosome.

- *The first base in a chromosome is numbered 0.*
- *The start position in each BED feature is therefore interpreted to be 1 greater than the start position listed in the feature. For example, start=9, end=20 is interpreted to span bases 10 through 20,inclusive.*
- *This column is required.*

3. **end** - The one-based ending position of the feature in the chromosome.

- *The end position in each BED feature is one-based. See example above.*

- *This column is required.*

4. **name** - Defines the name of the BED feature.

- *Any string can be used.* For example, “LINE”, “Exon3”, “HWIEAS_0001:3:1:0:266#0/1”, or “my_Feature”.
- *This column is optional.*

5. **score** - The UCSC definition requires that a BED score range from 0 to 1000, inclusive. However, bedtools allows any string to be stored in this field in order to allow greater flexibility in annotation features. For example, strings allow scientific notation for p-values, mean enrichment values, etc. It should be noted that this flexibility could prevent such annotations from being correctly displayed on the UCSC browser.

- *Any string can be used.* For example, 7.31E-05 (p-value), 0.33456 (mean enrichment value), “up”, “down”, etc.
- *This column is optional.*

6. **strand** - Defines the strand - either ‘+’ or ‘-‘.

- *This column is optional.*

7. **thickStart** - The starting position at which the feature is drawn thickly.

- *Allowed yet ignored by bedtools.*

8. **thickEnd** - The ending position at which the feature is drawn thickly.

- *Allowed yet ignored by bedtools.*

9. **itemRgb** - An RGB value of the form R,G,B (e.g. 255,0,0).

- *Allowed yet ignored by bedtools.*

10. **blockCount** - The number of blocks (exons) in the BED line.

- *Allowed yet ignored by bedtools.*

11. **blockSizes** - A comma-separated list of the block sizes.

- *Allowed yet ignored by bedtools.*

12. **blockStarts** - A comma-separated list of block starts.

- *Allowed yet ignored by bedtools.*

bedtools requires that all BED input files (and input received from stdin) are **tab-delimited**. The following types of BED files are supported by bedtools:

1. **BED3**: A BED file where each feature is described by **chrom**, **start**, and **end**.
   For example: chr1 11873 14409
2. **BED4**: A BED file where each feature is described by **chrom**, **start**, **end**, and **name**.
   For example: chr1 11873 14409 uc001aaa.3
3. **BED5**: A BED file where each feature is described by **chrom**, **start**, **end**, **name**, and **score**.
   For example: chr1 11873 14409 uc001aaa.3 0
4. **BED6**: A BED file where each feature is described by **chrom**, **start**, **end**, **name**, **score**, and **strand**.
   For example: chr1 11873 14409 uc001aaa.3 0 +
5. **BED12**: A BED file where each feature is described by all twelve columns listed above.
   For example: chr1 11873 14409 uc001aaa.3 0 + 11873
   11873 0 3 354,109,1189, 0,739,1347,

BEDPE format

We have defined a new file format (BEDPE) in order to concisely describe disjoint genome features, such as structural variations or paired-end sequence alignments. We chose to define a new format because the existing "blocked" BED format (a.k.a. BED12) does not allow inter-chromosomal feature definitions. In addition, BED12 only has one strand field, which is insufficient for paired-end sequence alignments, especially when studying structural variation.

The BEDPE format is described below. The description is modified from: http://genome.ucsc.edu/FAQ/FAQformat#format1.

1. **chrom1** - The name of the chromosome on which the **first** end of the feature exists.

- *Any string can be used.* For example, "chr1", "III", "myChrom", "contig1112.23".
- *This column is required.*
- *Use "." for unknown.*

2. **start1** - The zero-based starting position of the **first** end of the feature on **chrom1**.

- *The first base in a chromosome is numbered 0.*
- *As with BED format, the start position in each BEDPE feature is therefore interpreted to be 1 greater than the start position listed in the feature. This column is required.*
- *Use -1 for unknown.*

3. **end1** - The one-based ending position of the first end of the feature on **chrom1**.

- *The end position in each BEDPE feature is one-based.*
- *This column is required.*
- *Use -1 for unknown.*

4. **chrom2** - The name of the chromosome on which the **second** end of the feature exists.

- *Any string can be used.* For example, "chr1", "III", "myChrom", "contig1112.23".
- *This column is required.*
- *Use "." for unknown.*

5. **start2** - The zero-based starting position of the **second** end of the feature on **chrom2**.

- *The first base in a chromosome is numbered 0.*
- *As with BED format, the start position in each BEDPE feature is therefore interpreted to be 1 greater than the start position listed in the feature. This column is required.*
- *Use -1 for unknown.*

6. **end2** - The one-based ending position of the **second** end of the feature on **chrom2**.

- *The end position in each BEDPE feature is one-based.*
- *This column is required.*
- *Use -1 for unknown.*

7. **name** - Defines the name of the BEDPE feature.

- *Any string can be used.* For example, "LINE", "Exon3", "HWIEAS_0001:3:1:0:266#0/1", or "my_Feature".
- *This column is optional.*

8. **score** - The UCSC definition requires that a BED score range from 0 to 1000, inclusive. *However, bedtools allows any string to be stored in this field in order to allow greater flexibility in annotation features*. For example, strings allow scientific notation for p-values, mean enrichment values, etc. It should be noted that this flexibility could prevent such annotations from being correctly displayed on the UCSC browser.

- *Any string can be used*. For example, 7.31E-05 (p-value), 0.33456 (mean enrichment value), “up”, “down”, etc.
- *This column is optional*.

9. **strand1** - Defines the strand for the first end of the feature. Either ‘+’ or ‘-‘.

- *This column is optional*.
- *Use ”.” for unknown*.

10. **strand2** - Defines the strand for the second end of the feature. Either ‘+’ or ‘-‘.

- *This column is optional*.
- *Use ”.” for unknown*.

11. **Any number of additional, user-defined fields** - bedtools allows one to add as many additional fields to the normal, 10-column BEDPE format as necessary. These columns are merely “passed through” **pairToBed** and **pairToPair** and are not part of any analysis. One would use these additional columns to add extra information (e.g., edit distance for each end of an alignment, or “deletion”, “inversion”, etc.) to each BEDPE feature.

- *These additional columns are optional*.

Entries from an typical BEDPE file:

```
chr1  100   200   chr5  5000  5100  bedpe_example1  30   +  -
chr9  1000  5000  chr9  3000  3800  bedpe_example2  100  +  -
```

Entries from a BEDPE file with two custom fields added to each record:

```
chr1  10    20    chr5  50    60    a1     30      +    -  0  1
chr9  30    40    chr9  80    90    a2     100     +    -  2  1
```

GFF format

The GFF format is described on the Sanger Institute’s website (http://www.sanger.ac.uk/resources/software/gff/spec.html). The GFF description below is modified from the definition at this URL. All nine columns in the GFF format description are required by bedtools.

1. **seqname** - The name of the sequence (e.g. chromosome) on which the feature exists.

- *Any string can be used*. For example, “chr1”, “III”, “myChrom”, “contig1112.23”.
- *This column is required*.

2. **source** - The source of this feature. This field will normally be used to indicate the program making the prediction, or if it comes from public database annotation, or is experimentally verified, etc.

- *This column is required*.

3. **feature** - The feature type name. Equivalent to BED’s **name** field.

- *Any string can be used*. For example, “exon”, etc.
- *This column is required*.

4. **start** - The one-based starting position of feature on **seqname**.

- *This column is required*.

- *bedtools accounts for the fact the GFF uses a one-based position and BED uses a zero-based start position.*

5. **end** - The one-based ending position of feature on **seqname**.

- *This column is required.*

6. **score** - A score assigned to the GFF feature. Like BED format, bedtools allows any string to be stored in this field in order to allow greater flexibility in annotation features. We note that this differs from the GFF definition in the interest of flexibility.

- *This column is required.*

7. **strand** - Defines the strand. Use '+', '-' or '.'

- *This column is required.*

8. **frame** - The frame of the coding sequence. Use '0', '1', '2', or '.'.

- *This column is required.*

9. **attribute** - Taken from http://www.sanger.ac.uk/resources/software/gff/spec.html: From version 2 onwards, the attribute field must have an tag value structure following the syntax used within objects in a .ace file, flattened onto one line by semicolon separators. Tags must be standard identifiers ([A-Za-z][**AZa-z0-9_**]*). Free text values must be quoted with double quotes. *Note: all non-printing characters in such free text value strings (e.g. newlines, tabs, control characters, etc) must be explicitly represented by their C (UNIX) style backslash-escaped representation (e.g. newlines as 'n', tabs as 't').* As in ACEDB, multiple values can follow a specific tag. The aim is to establish consistent use of particular tags, corresponding to an underlying implied ACEDB model if you want to think that way (but acedb is not required).

- *This column is required.*

An entry from an example GFF file :

```
seq1 BLASTX similarity 101 235 87.1 + 0 Target "HBA_HUMAN" 11 55 ;
E_value 0.0003 dJ102G20 GD_mRNA coding_exon 7105 7201 . - 2 Sequence
"dJ102G20.C1.1"
```

Genome file format

Some of the bedtools (e.g., genomeCoverageBed, complementBed, slopBed) need to know the size of the chromosomes for the organism for which your BED files are based. When using the UCSC Genome Browser, Ensemble, or Galaxy, you typically indicate which which species/genome build you are working. The way you do this for bedtools is to create a "genome" file, which simply lists the names of the chromosomes (or scaffolds, etc.) and their size (in basepairs).

Genome files must be **tab-delimited** and are structured as follows (this is an example for *C. elegans*):

```
chrI  15072421
chrII 15279323
...
chrX  17718854
chrM  13794
```

bedtools includes pre-defined genome files for human and mouse in the **/genomes** directory included in the bedtools distribution.

SAM/BAM format

The SAM / BAM format is a powerful and widely-used format for storing sequence alignment data (see http://samtools.sourceforge.net/ for more details). It has quickly become the standard format to which most DNA sequence alignment programs write their output. Currently, the following bedtools support input in BAM format: `intersect`, `window`, `coverage`, `genomecov`, `pairtobed`, `bamtobed`. Support for the BAM format in bedtools allows one to (to name a few): compare sequence alignments to annotations, refine alignment datasets, screen for potential mutations and compute aligned sequence coverage.

VCF format

The Variant Call Format (VCF) was conceived as part of the 1000 Genomes Project as a standardized means to report genetic variation calls from SNP, INDEL and structural variant detection programs (see http://www.1000genomes.org/wiki/doku.php?id=1000_genomes:analysis:vcf4.0 for details). bedtools now supports the latest version of this format (i.e, Version 4.0). As a result, bedtools can be used to compare genetic variation calls with other genomic features.

1.5 The BEDTools suite

bedtools consists of a suite of sub-commands that are invoked as follows:

```
bedtools [sub-command] [options]
```

For example, to intersect two BED files, one would invoke the following:

```
bedtools intersect -a a.bed -b b.bed
```

1.5.1 The full list of *bedtools* sub-commands.

annotate

`bedtools annotate`, well, annotates one BED/VCF/GFF file with the coverage and number of overlaps observed from multiple other BED/VCF/GFF files. In this way, it allows one to ask to what degree one feature coincides with multiple other feature types with a single command.

Usage and option summary

Usage:

```
bedtools annotate [OPTIONS] -i <BED/GFF/VCF> -files FILE1 FILE2 FILE3 ... FILEn
```

(or):

```
annotateBed [OPTIONS] -i <BED/GFF/VCF> -files FILE1 FILE2 FILE3 ... FILEn
```

Option	Description
-names	A list of names (one per file) to describe each file in -i. These names will be printed as a header line.
-counts	Report the count of features in each file that overlap -i. Default behavior is to report the fraction of -i covered by each file.
-both	Report the count of features followed by the % coverage for each annotation file. Default is to report solely the fraction of -i covered by each file.
-s	Force strandedness. That is, only include hits in A that overlap B on the same strand. By default, hits are included 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.

Default behavior - annotate one file with coverage from others.

By default, the fraction of each feature covered by each annotation file is reported after the complete feature in the file to be annotated.

```
$ cat variants.bed
chr1 100  200   nasty 1  -
chr2 500  1000  ugly  2  +
chr3 1000 5000  big   3  -

$ cat genes.bed
chr1 150  200   geneA 1  +
chr1 175  250   geneB 2  +
chr3 0    10000 geneC 3  -

$ cat conserve.bed
chr1 0    10000 cons1 1  +
chr2 700  10000 cons2 2  -
chr3 4000 10000 cons3 3  +

$ cat known_var.bed
chr1 0    120   known1   -
chr1 150  160   known2   -
chr2 0    10000 known3   +

$ bedtools annotate -i variants.bed -files genes.bed conserve.bed known_var.bed
chr1  100     200     nasty   1       -       0.500000        1.000000        0.300000
chr2  500     1000    ugly    2       +       0.000000        0.600000        1.000000
chr3  1000    5000    big     3       -       1.000000        0.250000        0.000000
```

`-count` Report the count of hits from the annotation files

```
$ bedtools annotate -counts -i variants.bed -files genes.bed conserve.bed known_var.bed
chr1  100     200     nasty   1       -       2       1       2
chr2  500     1000    ugly    2       +       0       1       1
chr3  1000    5000    big     3       -       1       1       0
```

`-both` Report both the count of hits and the fraction covered from the annotation files

```
$ bedtools annotate -both -i variants.bed -files genes.bed conserve.bed known_var.bed
#chr  start   end     name    score   +/-     cnt1    pct1    cnt2    pct2    cnt3    pct3
chr1  100     200     nasty   1       -       2       0.500000        1       1.000000        2
chr2  500     1000    ugly    2       +       0       0.000000        1       0.600000        1
chr3  1000    5000    big     3       -       1       1.000000        1       0.250000        0
```

`-s` Restrict the reporting to overlaps on the same strand.

```
$ bedtools annotate -s -i variants.bed -files genes.bed conserve.bed known_var.bed
chr1  100     200     nasty   1       -       0.000000        0.000000        0.000000
chr2  500     1000    ugly    2       +       0.000000        0.000000        0.000000
chr3  1000    5000    big     3       -       1.000000        0.000000        0.000000
```

`-S` Restrict the reporting to overlaps on the opposite strand.

```
$ bedtools annotate -S -i variants.bed -files genes.bed conserve.bed known_var.bed
chr1  100     200     nasty   1       -       0.500000        1.000000        0.300000
chr2  500     1000    ugly    2       +       0.000000        0.600000        1.000000
chr3  1000    5000    big     3       -       0.000000        0.250000        0.000000
```

bamtobed

`bedtools bamtobed` is a conversion utility that converts sequence alignments in BAM format into BED, BED12, and/or BEDPE records.

Usage and option summary

Usage:

`bedtools bamtobed [OPTIONS] -i <BAM>`

(or):

`bamToBed [OPTIONS] -i <BAM>`

Option	Description
-bedpe	Write BAM alignments in BEDPE format. Only one alignment from paired-end reads will be reported. Specifically, it each mate is aligned to the same chromosome, the BAM alignment reported will be the one where the BAM insert size is greater than zero. When the mate alignments are interchromosomal, the lexicographically lower chromosome will be reported first. Lastly, when an end is unmapped, the chromosome and strand will be set to "." and the start and end coordinates will be set to -1. *By default, this is disabled and the output will be reported in BED format*.
-bed12	Write "blocked" BED (a.k.a. BED12) format. This will convert "spliced" BAM alignments (denoted by the "N" CIGAR operation) to BED12.
-split	Report each portion of a "split" BAM (i.e., having an "N" CIGAR operation) alignment as a distinct BED intervals.
-ed	Use the "edit distance" tag (NM) for the BED score field. Default for BED is to use mapping quality. Default for BEDPE is to use the *minimum* of the two mapping qualities for the pair. When -ed is used with -bedpe, the total edit distance from the two mates is reported.
-tag	Use other *numeric* BAM alignment tag for BED score. Default for BED is to use mapping quality. Disallowed with BEDPE output.
-color	An R,G,B string for the color used with BED12 format. Default is (255,0,0).
-cigar	Add the CIGAR string to the BED entry as a 7th column.

Default behavior

By default, each alignment in the BAM file is converted to a 6 column BED. The BED "name" field is comprised of the RNAME field in the BAM alignment. If mate information is available, the mate (e.g., "/1" or "/2") field will be appended to the name.

```
$ bedtools bamtobed -i reads.bam | head -3
chr7   118970079   118970129   TUPAC_0001:3:1:0:1452#0/1   37   -
chr7   118965072   118965122   TUPAC_0001:3:1:0:1452#0/2   37   +
chr11  46769934    46769984    TUPAC_0001:3:1:0:1472#0/1   37   -
```

`-tag` Set the score field based on BAM tags

One can override the choice of the BAM *MAPQ* as the result BED record's *score* field by using the `-tag` option. In the example below, we use the `-tag` option to select the BAM edit distance (the *NM* tag) as the score column in the resulting BED records.

```
$ bedtools bamtobed -i reads.bam -tag NM | head -3
chr7   118970079   118970129   TUPAC_0001:3:1:0:1452#0/1   1    -
chr7   118965072   118965122   TUPAC_0001:3:1:0:1452#0/2   3    +
chr11  46769934    46769984    TUPAC_0001:3:1:0:1472#0/1   1    -
```

`-bedpe` Set the score field based on BAM tags

The `-bedpe` option converts BAM alignments to BEDPE format, thus allowing the two ends of a paired-end alignment to be reported on a single text line. Specifically, it each mate is aligned to the same chromosome, the BAM alignment reported will be the one where the BAM insert size is greater than zero. When the mate alignments are interchromosomal, the lexicographically lower chromosome will be reported first. Lastly, when an end is unmapped, the chromosome and strand will be set to "." and the start and end coordinates will be set to -1.

Note: When using this option, it is required that the BAM file is sorted/grouped by the read name. This allows bamToBed to extract correct alignment coordinates for each end based on their respective CIGAR strings. It also assumes that the alignments for a given pair come in groups of twos. There is not yet a standard method for reporting multiple alignments using BAM. bamToBed will fail if an aligner does not report alignments in pairs.

```
$ bedtools bamtobed -i reads.ba -bedpe | head -3
chr7   118965072   118965122   chr7   118970079   118970129 TUPAC_0001:3:1:0:1452#0 37     +     -
chr11  46765606    46765656    chr11  46769934    46769984 TUPAC_0001:3:1:0:1472#0 37     +     -
chr20  54704674    54704724    chr20  54708987    54709037 TUPAC_0001:3:1:1:1833#0 37     +
```

One can easily use samtools and bamToBed together as part of a UNIX pipe. In this example, we will only convert properly-paired (`FLAG == 0x2`) reads to BED format.

```
$ samtools view -bf 0x2 reads.bam | bedtools bamtobed -i stdin | head
chr7   118970079   118970129   TUPAC_0001:3:1:0:1452#0/1   37   -
chr7   118965072   118965122   TUPAC_0001:3:1:0:1452#0/2   37   +
chr11  46769934    46769984    TUPAC_0001:3:1:0:1472#0/1   37   -
chr11  46765606    46765656    TUPAC_0001:3:1:0:1472#0/2   37   +
chr20  54704674    54704724    TUPAC_0001:3:1:1:1833#0/1   37   +
chr20  54708987    54709037    TUPAC_0001:3:1:1:1833#0/2   37   -
chrX   9380413     9380463     TUPAC_0001:3:1:1:285#0/1    0    -
chrX   9375861     9375911     TUPAC_0001:3:1:1:285#0/2    0    +
chrX   131756978   131757028   TUPAC_0001:3:1:2:523#0/1    37   +
chrX   131761790   131761840   TUPAC_0001:3:1:2:523#0/2    37   -
```

`-split` Creating BED12 features from "spliced" BAM entries.

`bedtools bamtobed` will, by default, create a BED6 feature that represents the entire span of a spliced/split BAM alignment. However, when using the `-split` command, a BED12 feature is reported where BED blocks will be created for each aligned portion of the sequencing read.

```
Chromosome  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Exons       ***************                                    **********

BED/BAM A      ^^^^^^^^^^^^....................................^^^^

Result      ===============                                    ====
```

bamtofastq

`bedtools bamtofastq` is a conversion utility for extracting FASTQ records from sequence alignments in BAM format.

Usage and option summary

Usage:

`bedtools bamtofastq [OPTIONS] -i <BAM> -fq <FASTQ>`

(or):

`bamToFastq [OPTIONS] -i <BAM> -fq <FASTQ>`

Option	Description
-fq2	FASTQ for second end. Used if BAM contains paired-end data. BAM should be sorted by query name (`samtools sort -n aln.bam aln.qsort`) if creating paired FASTQ with this option.
-tags	Create FASTQ based on the mate info in the BAM R2 and Q2 tags.

Default behavior

By default, each alignment in the BAM file is converted to a FASTQ record in the `-fq` file. The order of the records in the resulting FASTQ exactly follows the order of the records in the BAM input file.

```
$ bedtools bamtofastq -i NA18152.bam -fq NA18152.fq

$ head -8 NA18152.fq
@NA18152-SRR007381.35051
GGAGACATATCATATAAGTAATGCTAGGGTGAGTGGTAGGAAGTTTTTTCATAGGAGGTGTATGAGTTGGTCGTAGCGGAATCGGGGGTATGCTGTTCGA
+
<<<;;<;<;;<;;;;;;;;;;;;<<<:;;;;;;;;;;;;;;;;:::::;;;;;<<;;;;;;;;;;;;;;;;;;;;;;;;;;;;<<<<<;;;;;;;;;;;;
@NA18152-SRR007381.637219
AATGCTAGGGTGAGTGGTAGGAAGTTTTTTCATAGGAGGTGTATGAGTTGGTCGTAGCGGAATCGGGGGTATGCTGTTCGAATTCATAAGAACAGGGAGG
+
<<<<<<<<<<;;<;<;;;;<<;<888888899<;;;;;;<;;;;;;;;;;;;;;;;;;;;;;;;<<<<<;;;;;;;;;<;<<<<<;;;;;;;;;;;;;<<
```

`-fq2` Creating two FASTQ files for paired-end sequences.

If your BAM alignments are from paired-end sequence data, one can use the `-fq2` option to create two distinct FASTQ output files — one for end 1 and one for end 2.

Note: When using this option, it is required that the BAM file is sorted/grouped by the read name. This keeps the resulting records in the two output FASTQ files in the same order. One can sort the BAM file by query name with `samtools sort -n aln.bam aln.qsort`.

```
$ samtools sort -n aln.bam aln.qsort

$ bedtools bamtofastq -i aln.qsort.bam \
                      -fq aln.end1.fq \
                      -fq2 aln.end2.fq

$ head -8 aln.end1.fq
@SRR069529.2276/1
CAGGGAGAAGGAGGTAGGAAAGAGAAAGGACCAGGGAGGGGCGCATACACAGGACGCTCCGTGCGGTGATAGCAGCACCACACTGTGTTCAGTCGTCTGG
+
=;@>==##############################################################################################
@SRR069529.2406/1
GCTGGGAAAAGGATTCAGGATGTTGGTTTCTATCTTTGAGTTGCTGCTGTGCGGCTGTCCCTACACTCGCAGTACCCCTCGGACACCGTCTACTGTGGAG
+
=5@><<:?<?

$ head -8 aln.end2.fq
@SRR069529.2276/2
AGACCCAGAGAGGGACAGGATCTGTCCCAGATCATAAAATAGGGGGAGTGCTCCGTAGAGGCGTGCGCGGTGGCACCGTGCAGTAGTACGGGTGAGCGGG
+
####################################################################################################
@SRR069529.2406/2
TTCCCTACCCCTGGGGTCAGGGACTACAGCCAAGGGGAGAACTTTAGCAAGTAGACGTTAGTTATTTTGATTCCAGTGGGGACGCGCGTGTAGCGAGTTG
+
@>=AABB?AAACABBA>@?AAAA>B@@AB@AA:B@AA@??############################################################
```

bed12tobed6

bed12ToBed6 is a convenience tool that converts BED features in BED12 (a.k.a. “blocked” BED features such as genes) to discrete BED6 features. For example, in the case of a gene with six exons, bed12ToBed6 would create six separate BED6 features (i.e., one for each exon).

Usage and option summary

Usage:

```
bed12ToBed6 [OPTIONS] -i <BED12>
```

Option	Description
-i	The BED12 file that should be split into discrete BED6 features. *Use “stdin” when using piped input.*

Default behavior

Figure:

```
head data/knownGene.hg18.chr21.bed | tail -n 3
chr21 10079666  10120808   uc002yiv.1  0  -  10081686  1 0 1 2 0 6 0 8
      0     4   528,91,101,215, 0,1930,39750,40927,
chr21 10080031  10081687   uc002yiw.1  0  -  10080031  1 0 0 8 0 0 3 1
      0     2   200,91,    0,1565,
chr21 10081660  10120796   uc002yix.2  0  -  10081660  1 0 0 8 1 6 6 0
      0     3   27,101,223,0,37756,38913,

head data/knownGene.hg18.chr21.bed | tail -n 3 | bed12ToBed6 -i stdin
chr21 10079666  10080194  uc002yiv.1 0  -
chr21 10081596  10081687  uc002yiv.1 0  -
chr21 10119416  10119517  uc002yiv.1 0  -
chr21 10120593  10120808  uc002yiv.1 0  -
chr21 10080031  10080231  uc002yiw.1 0  -
chr21 10081596  10081687  uc002yiw.1 0  -
chr21 10081660  10081687  uc002yix.2 0  -
chr21 10119416  10119517  uc002yix.2 0  -
chr21 10120573  10120796  uc002yix.2 0  -
```

bedpetobam

bedtobam

bedToBam converts features in a feature file to BAM format. This is useful as an efficient means of storing large genome annotations in a compact, indexed format for visualization purposes.

Usage and option summary

Usage:

```
bedToBam [OPTIONS] -i <BED/GFF/VCF> -g <GENOME> > <BAM>
```

Option	Description
-mapq	Set a mapping quality (SAM MAPQ field) value for all BED entries. *Default: 255*
-ubam	Write uncompressed BAM output. The default is write compressed BAM output.
-bed12	Indicate that the input BED file is in BED12 (a.k.a "blocked" BED) format. In this case, bedToBam will convert blocked BED features (e.g., gene annotaions) into "spliced" BAM alignments by creating an appropriate CIGAR string.

Default behavior

The default behavior is to assume that the input file is in unblocked format. For example:

```
head -5 rmsk.hg18.chr21.bed
chr21 9719768  9721892  ALR/Alpha  1004  +
chr21 9721905  9725582  ALR/Alpha  1010  +
chr21 9725582  9725977  L1PA3 3288 +
chr21 9726021  9729309  ALR/Alpha  1051  +
chr21 9729320  9729809  L1PA3 3897 -

bedToBam -i rmsk.hg18.chr21.bed -g human.hg18.genome > rmsk.hg18.chr21.bam

samtools view rmsk.hg18.chr21.bam | head -5
ALR/Alpha  0   chr21 9719769  255  2124M *  0  0  *  *
ALR/Alpha  0   chr21 9721906  255  3677M *  0  0  *  *
L1PA3      0   chr21 9725583  255  395M  *  0  0  *  *
ALR/Alpha  0   chr21 9726022  255  3288M *  0  0  *  *
L1PA3      16  chr21 9729321  255  489M  *  0  0  *  *
```

Creating "spliced" BAM entries from "blocked" BED features

Optionally, **bedToBam** will create spliced BAM entries from "blocked" BED features by using the -bed12 option. This will create CIGAR strings in the BAM output that will be displayed as "spliced" alignments. The image illustrates this behavior, as the top track is a BAM representation (using bedToBam) of a BED file of UCSC genes.

For example:

```
bedToBam -i knownGene.hg18.chr21.bed -g human.hg18.genome -bed12 > knownGene.bam

samtools view knownGene.bam | head -2
```

```
uc002yip.1  16   chr21 9928614   2                       5                        5

298M1784N71M1411N93M3963N80M1927N106M3608N81M1769N62M11856N89M98N82M816N61M6910N65M
738N64M146N100M1647N120M6478N162M1485N51M6777N60M9274N54M880N54M1229N54M2377N54M112
68N58M2666N109M2885N158M     *   0  0  *  *
uc002yiq.1  16   chr21 9928614   2                       5                        5

298M1784N71M1411N93M3963N80M1927N106M3608N81M1769N62M11856N89M98N82M816N61M6910N65M
738N64M146N100M1647N120M6478N162M1485N51M6777N60M10208N54M1229N54M2377N54M11268N58M
2666N109M2885N158M       *   0   0  *  *
```

closest

Similar to **intersectBed, closestBed** searches for overlapping features in A and B. In the event that no feature in B overlaps the current feature in A, **closestBed** will report the *closest* (that is, least genomic distance from the start or end of A) feature in B. For example, one might want to find which is the closest gene to a significant GWAS polymorphism. Note that **closestBed** will report an overlapping feature as the closest—that is, it does not restrict to closest *non-overlapping* feature.

5.6.1 Usage and option summary

Usage:

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

Option	Description
-s	Force strandedness. That is, find the closest feature in B overlaps A on the same strand. *By default, this is disabled.*
-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.
-t	How ties for closest feature should be handled. This occurs when two features in B have exactly the same overlap with a feature in A. *By default, all such features in B are reported.* Here are the other choices controlling how ties are handled: *all-* Report all ties (default). *first-* Report the first tie that occurred in the B file. *last-* Report the last tie that occurred in the B file.

Default behavior

closestBed first searches for features in B that overlap a feature in A. If overlaps are found, the feature in B that overlaps the highest fraction of A is reported. If no overlaps are found, **closestBed** looks for the feature in B that is *closest* (that is, least genomic distance to the start or end of A) to A. For example, in the figure below, feature B1 would be reported as the closest feature to A1.

```
Chromosome  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

BED FILE A                             *************

BED File B         ^^^^^^^^                            ^^^^^^

Result                                                 ======
```

For example:

```
cat A.bed
chr1  100  200

cat B.bed
chr1  500  1000
chr1  1300 2000

closestBed -a A.bed -b B.bed
chr1  100  200  chr1  500  1000
```

`-s` Enforcing "strandedness"

This option behaves the same as the -s option for intersectBed while scanning for the closest (overlapping or not) feature in B. See the discussion in the intersectBed section for details.

`-t` Controlling how ties for "closest" are broken

When there are two or more features in B that overlap the *same fraction* of A, **closestBed** will, by default, report both features in B. Imagine feature A is a SNP and file B contains genes. It can often occur that two gene annotations (e.g. opposite strands) in B will overlap the SNP. As mentioned, the default behavior is to report both such genes in B. However, the -t option allows one to optionally choose the just first or last feature (in terms of where it occurred in the input file, not chromosome position) that occurred in B.

For example (note the difference between -l 200 and -l 300):

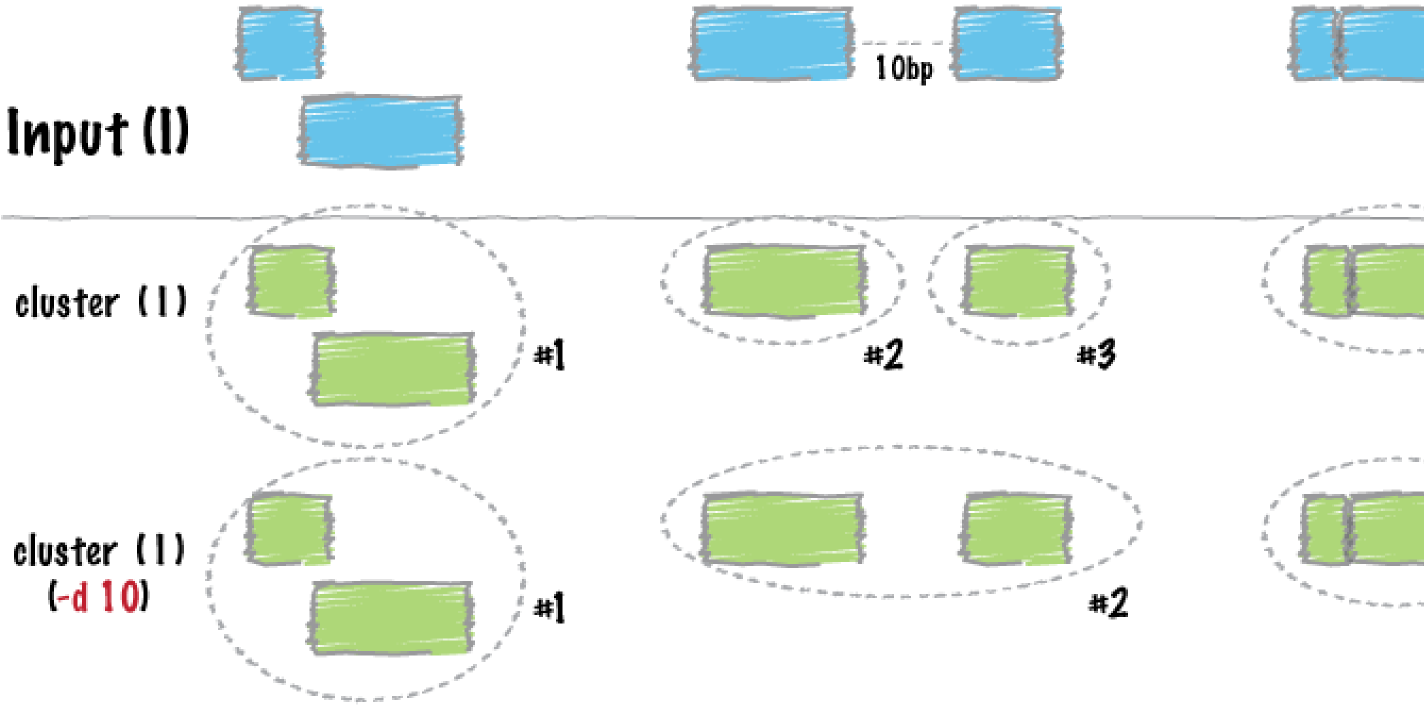
```
cat A.bed
chr1  100  101  rs1234

cat B.bed
chr1  0  1000  geneA  100  +
chr1  0  1000  geneB  100  -

closestBed -a A.bed -b B.bed
chr1  100  101  rs1234  chr1  0  1000  geneA  100  +
chr1  100  101  rs1234  chr1  0  1000  geneB  100  -

closestBed -a A.bed -b B.bed -t all
chr1  100  101  rs1234  chr1  0  1000  geneA  100  +
chr1  100  101  rs1234  chr1  0  1000  geneB  100  -

closestBed -a A.bed -b B.bed -t first
chr1  100  101  rs1234  chr1  0  1000  geneA  100  +

closestBed -a A.bed -b B.bed -t last
chr1  100  101  rs1234  chr1  0  1000  geneB  100  -
```

-d Reporting the distance to the closest feature in base pairs

ClosestBed will optionally report the distance to the closest feature in the B file using the **-d** option. When a feature in B overlaps a feature in A, a distance of 0 is reported.

```
cat A.bed
chr1  100  200
chr1  500  600

cat B.bed
chr1  500  1000
chr1  1300 2000

closestBed -a A.bed -b B.bed -d
chr1  100  200  chr1  500  1000  300
chr1  500  600  chr1  500  1000  0
```

cluster



Similar to *merge*, `cluster` report each set of overlapping or "book-ended" features in an interval file. In contrast to `merge`, `cluster` does not flatten the cluster of intervals into a new meta-interval; instead, it assigns an unique cluster ID to each record in each cluster. This is useful for having fine control over how sets of overlapping intervals in a single interval file are combined.

Note: `bedtools cluster` requires that you presort your data by chromosome and then by start position (e.g., `sort k1,1 -k2,2n in.bed > in.sorted.bed` for BED files).

See Also:

merge

Usage and option summary

Usage:

`bedtools cluster [OPTIONS] -i <BED/GFF/VCF>`

(or):

`clusterBed [OPTIONS] -i <BED/GFF/VCF>`

Option	Description
-s	Force strandedness. That is, only cluster features that are the same strand. *By default, this is disabled.*
-d	Maximum distance between features allowed for features to be clustered. *Default is 0. That is, overlapping and/or book-ended features are clustered.*

Default behavior

By default, `bedtools cluster` collects overlapping (by at least 1 bp) and/or bookended intervals into distinct clusters. In the example below, the 4th column is the cluster ID.

```
$ cat A.bed
chr1  100  200
chr1  180  250
chr1  250  500
chr1  501  1000

$ bedtools cluster -i A.bed
chr1  100     200     1
chr1  180     250     1
chr1  250     500     1
chr1  501     1000    2
```

`-s` Enforcing "strandedness"

The `-s` option will only cluster intervals that are overlapping/bookended *and* are on the same strand.

```
$ cat A.bed
chr1  100  200   a1  1 +
chr1  180  250   a2  2 +
chr1  250  500   a3  3 -
chr1  501  1000  a4  4 +

$ bedtools cluster -i A.bed -s
chr1  100     200     a1      1       +       1
chr1  180     250     a2      2       +       1
```

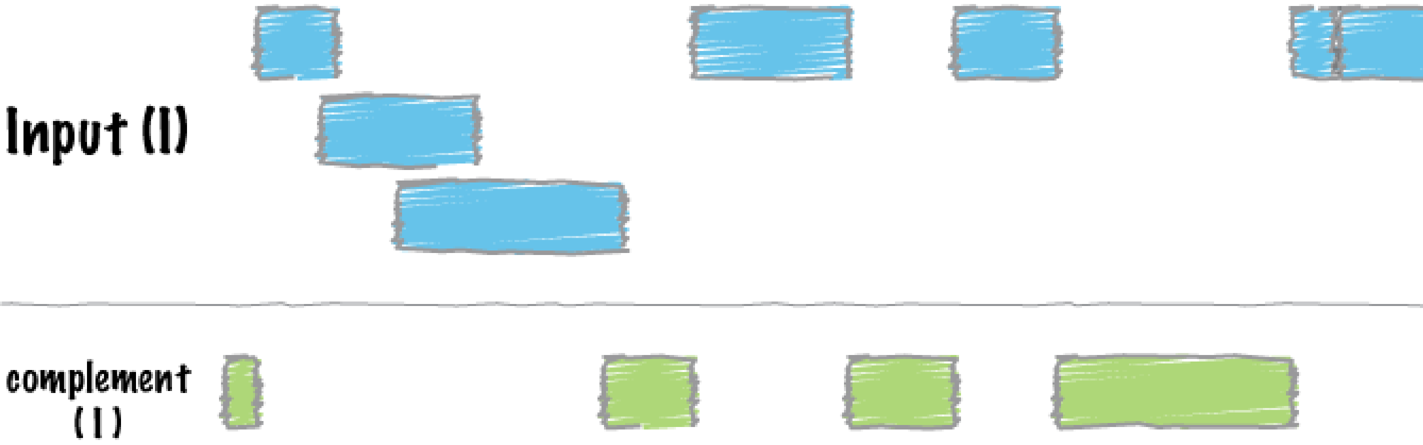
```
chr1  501    1000    a4     4      +      2
chr1  250    500     a3     3      -      3
```

`-d` Controlling how close two features must be in order to cluster

By default, only overlapping or book-ended features are combined into a new feature. However, one can force `cluster` to combine more distant features with the `-d` option. For example, were one to set `-d` to 1000, any features that overlap or are within 1000 base pairs of one another will be clustered.

```
$ cat A.bed
chr1  100  200
chr1  501  1000

$ bedtools cluster -i A.bed
chr1  100  200    1
chr1  501  1000   2

$ bedtools cluster -i A.bed -d 1000
chr1  100  200    1
chr1  501  1000   1
```

complement



`bedtools complement` returns all intervals in a genome that are **not** covered by at least one interval in the input BED/GFF/VCF file.

See Also:

merge

Usage and option summary

Usage:

```
bedtools complement -i <BED/GFF/VCF> -g <GENOME>
```

(or):

```
complementBed -i <BED/GFF/VCF> -g <GENOME>
```

Default behavior

By default, `bedtools complement` returns all genomic intervals that are not covered by at least one record from the input file.

```
$ cat A.bed
chr1  100  200
chr1  400  500
chr1  500  800

$ cat my.genome
chr1  1000
chr2  800

$ bedtools complement -i A.bed -g my.genome
chr1  0   100
chr1  200 400
chr1  800 1000
chr2  0   800
```

coverage

coverageBed computes both the *depth* and *breadth* of coverage of features in file A across the features in file B. For example, **coverageBed** can compute the coverage of sequence alignments (file A) across 1 kilobase (arbitrary) windows (file B) tiling a genome of interest. One advantage that **coverageBed** offers is that it not only *counts* the number of features that overlap an interval in file B, it also computes the fraction of bases in B interval that were overlapped by one or more features. Thus, **coverageBed** also computes the *breadth* of coverage for each interval in B.

Usage and option summary

Usage:

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

Option	Description
-abam	BAM file A. Each BAM alignment in A is compared to B in search of overlaps. Use "stdin" if passing A with a UNIX pipe: For example: samtools view -b <BAM> | intersectBed -abam stdin -b genes.bed
-s	Force strandedness. That is, only features in A are only counted towards coverage in B if they are the same strand. *By default, this is disabled and coverage is counted without respect to strand.*
-hist	Report a histogram of coverage for each feature in B as well as a summary histogram for _all_ features in B. Output (tab delimited) after each feature in B: 1) depth 2) # bases at depth 3) size of B 4) % of B at depth
-d	Report the depth at each position in each B feature. Positions reported are one based. Each position and depth follow the complete B feature.
-split	Treat "split" BAM or BED12 entries as distinct BED intervals when computing coverage. For BAM files, this uses the CIGAR "N" and "D" operations to infer the blocks for computing coverage. For BED12 files, this uses the BlockCount, BlockStarts, and BlockEnds fields (i.e., columns 10,11,12).

Default behavior

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.

Below are the number of features in A (N=...) overlapping B and fraction of bases in B with coverage.

```
Chromosome  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

BED FILE B  ***************     ***************     ******   **************

BED File A  ^^^^ ^^^^              ^^             ^^^^^^^^^   ^^^ ^^ ^^^^
              ^^^^^^^^                                     ^^^^^ ^^^^^ ^^

Result      [  N=3, 10/15 ]     [  N=1, 2/16 ]    [N=1,6/6]   [N=5, 11/12 ]
```

For example:

```
cat A.bed
chr1  10  20
chr1  20  30
chr1  30  40
chr1  100 200

cat B.bed
chr1  0   100
chr1  100 200
chr2  0   100

coverageBed -a A.bed -b B.bed
chr1  0   100  3  30  100 0.3000000
chr1  100 200  1  100 100 1.0000000
chr2  0   100  0  0   100 0.0000000
```

`-s` Calculating coverage by strand

Use the "**-s**" option if one wants to only count coverage if features in A are on the same strand as the feature / window in B. This is especially useful for RNA-seq experiments.

For example (note the difference in coverage with and without **-s**:

```
cat A.bed
chr1  10  20  a1  1  -
chr1  20  30  a2  1  -
chr1  30  40  a3  1  -
chr1  100 200 a4  1  +

cat B.bed
chr1  0   100 b1  1  +
chr1  100 200 b2  1  -
chr2  0   100 b3  1  +

coverageBed -a A.bed -b B.bed
chr1  0   100 b1  1  +  3  30  100  0.3000000
chr1  100 200 b2  1  -  1  100 100  1.0000000
chr2  0   100 b3  1  +  0  0   100  0.0000000

coverageBed -a A.bed -b B.bed -s
chr1  0   100 b1  1  +  0  0   100  0.0000000
chr1  100 200 b2  1  -  0  0   100  0.0000000
chr2  0   100 b3  1  +  0  0   100  0.0000000
```

`-hist` Creating a histogram of coverage for each feature in the B file

One should use the "**-hist**" option to create, for each interval in B, a histogram of coverage of the features in A across B.

In this case, each entire feature in B will be reported, followed by the depth of coverage, the number of bases at that depth, the size of the feature, and the fraction covered. After all of the features in B have been reported, a histogram summarizing the coverage among all features in B will be reported.

```
cat A.bed
chr1  10  20  a1  1  -
chr1  20  30  a2  1  -
chr1  30  40  a3  1  -
chr1  100 200 a4  1  +

cat B.bed
chr1  0   100 b1  1  +
chr1  100 200 b2  1  -
chr2  0   100 b3  1  +

coverageBed -a A.bed -b B.bed -hist
chr1  0   100 b1  1  +  0  70  100  0.7000000
chr1  0   100 b1  1  +  1  30  100  0.3000000
chr1  100 200 b2  1  -  1  100 100  1.0000000
chr2  0   100 b3  1  +  0  100 100  1.0000000
all   0   170 300 0.5666667
all   1   130 300 0.4333333
```

`-hist` Reporting the per-base of coverage for each feature in the B file

One should use the “**-d**” option to create, for each interval in B, a detailed list of coverage at each of the positions across each B interval.

The output will consist of a line for each one-based position in each B feature, followed by the coverage detected at that position.

```
cat A.bed
chr1  0  5
chr1  3  8
chr1  4  8
chr1  5  9

cat B.bed
chr1  0  10

coverageBed -a A.bed -b B.bed -d
chr1  0  10  B  1  1
chr1  0  10  B  2  1
chr1  0  10  B  3  1
chr1  0  10  B  4  2
chr1  0  10  B  5  3
chr1  0  10  B  6  3
chr1  0  10  B  7  3
chr1  0  10  B  8  3
chr1  0  10  B  9  1
chr1  0  10  B  10 0
```

`-split` Reporting coverage with spliced alignments or blocked BED features

As described in section 1.3.19, coverageBed will, by default, screen for overlaps against the entire span of a spliced/split BAM alignment or blocked BED12 feature. When dealing with RNA-seq reads, for example, one typically wants to only tabulate coverage for the portions of the reads that come from exons (and ignore the interstitial intron sequence). The **-split** command allows for such coverage to be performed.

expand

flank

genomecov

genomeCoverageBed computes a histogram of feature coverage (e.g., aligned sequences) for a given genome. Optionally, by using the **-d** option, it will report the depth of coverage at *each base* on each chromosome in the genome file (**-g**).

Usage and option summary

Usage:

```
genomeCoverageBed [OPTIONS] -i <BED> -g <GENOME>
```

NOTE: genomeCoverageBed requires that the input BED file be sorted by chromosome. A simple sort -k1,1 will suffice.

Option	Description
-ibam	BAM file as input for coverage. Each BAM alignment in A added to the total coverage for the genome. Use "stdin" if passing it with a UNIX pipe: For example: samtools view -b <BAM> | genomeCoverageBed -ibam stdin -g hg18.genome
-d	Report the depth at each genome position. *Default behavior is to report a histogram.*
-max	Combine all positions with a depth >= max into a single bin in the histogram.
-bg	Report depth in BedGraph format. For details, see: http://genome.ucsc.edu/goldenPath/help/bedgraph.html
-bga	Report depth in BedGraph format, as above (i.e., -bg). However with this option, regions with zero coverage are also reported. This allows one to quickly extract all regions of a genome with 0 coverage by applying: "grep -w 0$" to the output.
-split	Treat "split" BAM or BED12 entries as distinct BED intervals when computing coverage. For BAM files, this uses the CIGAR "N" and "D" operations to infer the blocks for computing coverage. For BED12 files, this uses the BlockCount, BlockStarts, and BlockEnds fields (i.e., columns 10,11,12).
-strand	Calculate coverage of intervals from a specific strand. With BED files, requires at least 6 columns (strand is column 6).

Default behavior

By default, **genomeCoverageBed** will compute a histogram of coverage for the genome file provided. The default output format is as follows: 1. chromosome (or entire genome) 2. depth of coverage from features in input file 3. number of bases on chromosome (or genome) with depth equal to column 2. 4. size of chromosome (or entire genome) in base pairs 5. fraction of bases on chromosome (or entire genome) with depth equal to column 2.

For example:

```
cat A.bed
chr1  10  20
chr1  20  30
chr2  0   500

cat my.genome
chr1  1000
chr2  500

genomeCoverageBed -i A.bed -g my.genome
chr1   0  980  1000  0.98
chr1   1  20   1000  0.02
chr2   1  500  500   1
genome 0  980  1500  0.653333
genome 1  520  1500  0.346667
```

`-max` Controlling the histogram's maximum depth

Using the **-max** option, **genomeCoverageBed** will "lump" all positions in the genome having feature coverage greater than or equal to **max** into the **max** histogram bin. For example, if one sets **-max** equal to 50, the max depth reported in the output will be 50 and all positions with a depth >= 50 will be represented in bin 50.

`-d` Reporting "per-base" genome coverage

Using the **-d** option, **genomeCoverageBed** will compute the depth of feature coverage for each base on each chromosome in genome file provided.

The "per-base" output format is as follows: 1. chromosome 2. chromosome position 3. depth (number) of features overlapping this chromosome position.

For example:

```
cat A.bed
chr1  10  20
chr1  20  30
chr2  0   500

cat my.genome
chr1  1000
chr2  500

genomeCoverageBed -i A.bed -g my.genome -d | head -15 | tail -n 10
chr1  6  0
chr1  7  0
chr1  8  0
chr1  9  0
chr1  10 0
```

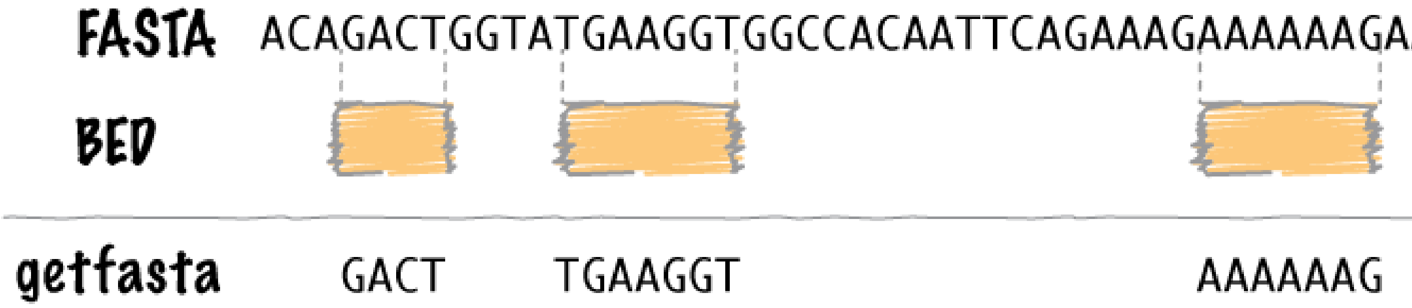
```
chr1   11 1
chr1   12 1
chr1   13 1
chr1   14 1
chr1   15 1
```

`-split` Reporting coverage with spliced alignments or blocked BED features

As described in section 1.3.19, genomeCoverageBed will, by default, screen for overlaps against the entire span of a spliced/split BAM alignment or blocked BED12 feature. When dealing with RNA-seq reads, for example, one typically wants to only screen for overlaps for the portions of the reads that come from exons (and ignore the interstitial intron sequence). The **-split** command allows for such overlaps to be performed.

For additional details, please visit the Usage From The Wild site and have a look at example 5, contributed by Assaf Gordon.

getfasta



`bedtools getfasta` extracts sequences from a FASTA file for each of the intervals defined in a BED/GFF/VCF file.

Tip: 1. The headers in the input FASTA file must *exactly* match the chromosome column in the BED file.

2. You can use the UNIX `fold` command to set the line width of the FASTA output. For example, `fold -w 60` will make each line of the FASTA file have at most 60 nucleotides for easy viewing.

See Also:

maskfasta

Usage and option summary

Usage

```
$ bedtools getfasta [OPTIONS] -fi <input FASTA> -bed <BED/GFF/VCF> -fo <output FASTA>
```

(or):

```
$ getFastaFromBed [OPTIONS] -fi <input FASTA> -bed <BED/GFF/VCF> -fo <output FASTA>
```

Option	Description
-name	Use the "name" column in the BED file for the FASTA headers in the output FASTA file.
-tab	Report extract sequences in a tab-delimited format instead of in FASTA format.
-s	Force strandedness. If the feature occupies the antisense strand, the sequence will be reverse complemented. *Default: strand information is ignored.*
-split	Given BED12 input, extract and concatenate the sequences from the BED "blocks" (e.g., exons)
-s	Force strandedness. If the feature occupies the antisense strand, the sequence will be reverse complemented. | By default, strand information is ignored.

Default behavior

`bedtools getfasta` will extract the sequence defined by the coordinates in a BED interval and create a new FASTA entry in the output file for each extracted sequence. By default, the FASTA header for each extracted sequence will be formatted as follows: "<chrom>:<start>-<end>".

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1 5 10

$ bedtools getfasta -fi test.fa -bed test.bed -fo test.fa.out

$ cat test.fa.out
>chr1:5-10
AAACC
```

`-name` Using the BED "name" column as a FASTA header.

Using the `-name` option, one can set the FASTA header for each extracted sequence to be the "name" columns from the BED feature.

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1 5 10 myseq

$ bedtools getfasta -fi test.fa -bed test.bed -fo test.fa.out -name

$ cat test.fa.out
```

```
>myseq
AAACC
```

`-tab` Creating a tab-delimited output file in lieu of FASTA output.

Using the `-tab` option, the `-fo` output file will be tab-delimited instead of in FASTA format.

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1 5 10 myseq

$ bedtools getfasta -fi test.fa -bed test.bed -fo test.fa.out.tab -name -tab

$ cat test.fa.out
myseq AAACC
```

`-s` Forcing the extracted sequence to reflect the requested strand

`bedtools getfasta` will extract the sequence in the orientation defined in the strand column when the "-s" option is used.

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1 20 25 forward 1 +
chr1 20 25 reverse 1 -

$ bedtools getfasta -fi test.fa -bed test.bed -s -name -fo test.fa.out

$ cat test.fa.out
>forward
CGCTA
>reverse
TAGCG
```

`-split` Extracting BED "blocks".

One can optionally request that FASTA records be extracting and concatenating each block in a BED12 record. For example, consider a BED12 record describing a transcript. By default, `getfasta` will extract the sequence representing the entire transcript (intons, exons, UTRs). Using the -split option, `getfasta` will instead produce separate a FASTA record representing a transcript that splices together each BED12 block (e.g., exons and UTRs in the case of genes described with BED12).

```
$ cat genes.bed12
chr1  164404  173864  ENST00000466557.1       0       -       173864  173864  0       6       387,59
chr1  235855  267253  ENST00000424587.1       0       -       267253  267253  0       4       2100,1
chr1  317810  328455  ENST00000426316.1       0       +       328455  328455  0       2       323,14

$ bedtools getfasta -fi chr1.fa -bed genes.bed12 -split -name -fo stdout
```

```
>ENST00000466557.1
gaggcgggaagatcacttgatatcaggagtcgaggcgggaagatcacttgacgtcaggagttcgagactggcccggccaacatggtgaaaccgcatctcc
>ENST00000424587.1
ccaggaagtgaaaatgacactttactgttttaatttgcatttctctgcttacaagtggattacacacattttcgtgtgctgttggctacttatTCATTCA
>ENST00000426316.1
AATGATCAAATTATGTTTCCCATGCATCAGGTGCAATGGGAAGCTCTTctggagagtgagagaagcttccagttaaggtgacattgaagccaagtcctga

# use the UNIX fold command to wrap the FASTA sequence such that each line
# has at most 60 characters
$ bedtools getfasta -fi chr1.fa -bed genes.bed12 -split -name -fo stdout | \
      fold -w 60
>ENST00000466557.1
gaggcgggaagatcacttgatatcaggagtcgaggcgggaagatcacttgacgtcaggag
ttcgagactggcccggccaacatggtgaaaccgcatctccactaaaaatacaaaaattag
cctggtatggtggtgggcacctgtaatcccagtgacttgggaggctaaggcaggagaatt
tcttgaacccaggaggcagaggttgcagtgaccagcaaggttgcgccattgcaccccagc
ctgggcgataagagtgaaaactccatctcaaaaaaaaaaaaaaaaaaaaaaTTCCTTTGG
GAAGGCCTTCTACATAAAAATCTTCAACATGAGACTGGAAAAAAGGGTATGGGATCATCA
CCGGACCTTTGGCTTTTACAGCTCGAGCTGACAAAGTTGATTTATCAAGTTGTAAATCTT
CACCTGTTGAATTCATAAGTTCATGTCATATTTTCTTTCAGACAATTCTTCAGTTTGTTT
ACGTAGATCAGCGATACGATGATTCCATTTCTtcggatccttgtaagagcagagcaggtg
atggagagggtgggaggtgtagtgacagaagcaggaaactccagtcattcgagacgggca
gcacaagctgcggagtgcaggccacctctacggccaggaaacggattctcccgcagagcc
tcggaagctaccgaccctgctcccaccttgactcagtaggacttactgtagaattctggc
cttcagacCTGAGCCTGGCAGCTCTCTCCAACTTTGGAAGCCCAGGGGCATGGCCCCTGT
CCACAGATGCACCTGGCATGAGGCGTGCCCAGAGGGACAGAGGCAGATGAGTttcgtctc
ctccactggattgtgagggcCAGAGTTGAACTCCCTCATTTTCCGTTCCCCAGCATTGGC
AGGTTCTGGGACTGGTGGCTGTGGTGGCTCGTTGGTCTTTGTCTCTTAGAAGGTGGGGAA
TAATCATCATCT
>ENST00000424587.1
ccaggaagtgaaaatgacactttactgttttaatttgcatttctctgcttacaagtggat
tacacacattttcgtgtgctgttggctacttatTCATTCAGAAAACATACTAAGTGCTGG
CTCTTTTTCATGTCCTTTATCAAGTTTGGATCATGTCATTTGCTATTTTCTTTCTGATGT
AAACTCTCAAAGTCTGAAGTGTATTGTCTTTTCCTGACACATATGTTGTAAATAATTTTC
TGGCTTACATTTTGACTTTTAATTTCATTCACGATGTTTTTAATGAATAATTTTAATTTT
TATGAATGCAAGTTAAAATAATTCTTTCATTGTGGTCTCTGACATGTCATGCCAATAAGG
GTCTTCTCCTCCAAGAGCACAGAAATATTTGCCAATACTGTCCTTAAAATCGGTCACAGT
TTCATTTTTTATATATGCATTTTACTTCAATTGGGGCTTCATTTTACTGAATGCCCTATT
TGAAGCAAGTTTCTCAGTTAATTCTTTTCTCAAAGGGCTAAGTATGGTAGATTGCAAACA
TAAGTGGCCACATAATGCTCTCACCTCctttgcctcctctcccaggaggagatagcgtcc
atctttccactccttaatctgggcttggccgtgtgacttgcactggccaatgggatatta
acaagtctgatgtgcacagaggctgtagaatgtgcacgggggcttggtctctcttgctgc
cctggagaccagctgccCCACGAAGGAACCAGAGCCAACCTGCTGCTTCCTGGAGGAAGA
CAGTCCCTCTGTCCCTCTGTCTCTGCCAACCAGTTAACCTGCTGCTTCCTGGAGGGAGAC
AGTCCCTCAGTCCCTCTGTCTCTGCCAACCAGTTAACCTGCTGCTTCCTGGAGGAAGACA
GTCACTCTGTCTCTGccaacccagttgaccgcagacatgcaggtctgctcaggtaagacc
agcacagtccctgccctgtgagccaaaccaaatggtccagccacagaatcgtgagcaaat
aagtgatgcttaagtcactaagatttgggCAAAAGCTGAGCATTTATCCCAATCCCAATA
CTGTTTGTCCTTCTGTTTATCTGTCTGTCCTTCCCTGCTCATTTAAAATGCCCCCACTGC
ATCTAGTACATTTTTATAGGATCAGGGATCTGCTCTTGGATTAATGTTGTGTTCCCACCT
CGAGGCAGCTTTGTAAGCTTCTGAGCACTTCCCAATTCCGGGTGACTTCAGGCACTGGGA
GGCCTGTGCATCAGCTGCTGCTGTCTGTAGCTGACTTCCTTCACCCCTCTGCTGTCCTCA
GCTCCTTCACCCCTGGGCCTCAGGAAATCAATGTCATGCTGACATCACTCTAGATCTAAA
AGTTGGGTTCTTGgaccaggcgtggtggctcacacctgtaatcccagcactttgggaggc
cgaggcgggtggatcacaaggtcaggagatcaagacgattctggctaacacggtgaaacc
ccgtctctactaaaaatacaaaaaaattagccgggtgtggtggcaggtgcctgtagcccc
agctacttgggaggctgaggcaggagaatggcttgaacctgggaggtggagcttgcagtg
agccaagatcacgccactgcactccagaatgggagagagagcgagactttctcaaaaaaa
```

```
aaaaaaaaaCTTAGGTTCTTGGATGTTCGGGAAAGGGGGTTATTATCTAGGATCCTTGAA
GCACCCCCAAGGGCATCTTCTCAAAGTTGGATGTGTGCATTTTCCTGAGAGGAAAGCTTT
CCCACATTATACAGCTTCTGAAAGGGTTGCTTGACCCACAGATGTGAAGCTGAGGCTGAA
GGAGACTGATGTGGTTTCTCCTCAGTTTCTCTGTGCAGCACCAGGTGGCAGCAGAGGTCA
GCAAGGCAAACCCGAGCCCGGGGATGCGGAGTGGGGGCAGCTACGTCCTCTCTTGAGCTA
CAGCAGATTCACTCTGTTCTGTTTCATTGTTGTTTAGTTTGCGTTGTGTTTCTCCAACTT
TGTGCCTCATCAGGAAAAGCTTTGGATCACAATTCCCAGtgctgaagaaaaggccaaact
cttggttgtgttctttgattAGTgcctgtgacgcagcttcaggaggtcctgagaacgtgt
gcacagtttagtcggcagaaacttagggaaatgtaagaccaccatcagcacataggagtt
ctgcattggtttggtctgcattggtttggtCTTTTCCTGGATACAGGTCTTGATAGGTCT
CTTGATGTCATTTCACTTCAGATTCTTCTTTAGAAAACTTGGACAATAGCATTTGCTGTC
TTGTCCAAATTGTTACTTCAAGTTTGCTCTTAGCAAGTAATTGTTTCAGTATCTATATCA
AAAATGGCTTAAGCCTGCAACATGTTTCTGAATGATTAACAAGGTGATAGTCAGTTCTTC
ATTGAATCCTGGATGCTTTATTTTTCTTAATAAGAGGAATTCATATGGATCAG
>ENST00000426316.1
AATGATCAAATTATGTTTCCCATGCATCAGGTGCAATGGGAAGCTCTTctggagagtgag
agaagcttccagttaaggtgacattgaagccaagtcctgaaagatgaggaagagttgtat
gagagtggggagggaaggggggaggtggagggaTGGGGAATGGGCCGGGATGGGATAGCGC
AAACTGCCCGGGAAGGGAAACCAGCACTGTACAGACCTGAACAACGAAGATGGCATATTT
TGTTCAGGGAATGGTGAATTAAGTGTGGCAGGAATGCTTTGTAGACACAGTAATTTGCTT
GTATGGAATTTTGCCTGAGAGACCTCATTCCTCACGTCGGCCATTCCAGGCCCCGTTTTT
CCCTTCCGGCAGCCTCTTGGCCTCTAATTTGTTTATCTTTTGTGTATAAATCCCAAAATA
TTGAATTTTGGAATATTTCCACCATTATGTAAATATTTTGATAGGTAA
```

groupby

`bedtools groupby` is a useful tool that mimics the "group by" clause in database systems. Given a file or stream that is sorted by the appropriate "grouping columns" (`-g`), `groupby` will compute summary statistics on another column (`-c`) in the file or stream. This will work with output from all BEDTools as well as any other tab-delimited file or stream. As such, this is a generally useful tool for all command-line analyses, not just genomics related research.

Note: When using `bedtools groupby`, the input data must be ordered by the same columns as specified with the `-grp` argument, which establish which columns should be used to define a group of similar data. For example, if `-grp` is 1,2,3, the data should be pre-grouped accordingly. When `bedtools groupby` detects changes in the group columns it then summarizes all lines with that group. For example, `sort -k1,1 -k2,2 -k3,3 data.txt | bedtools groupby -g 1,2,3 -c 4 -o mean`.

Usage and option summary

Usage

```
bedtools groupby [OPTIONS] -i <input> -g <group columns> -c <op. column> -o <operation>
```

or:

```
groupBy [OPTIONS] -i <input> -g <group columns> -c <op. column> -o <operation>
```

Option	Description
-i	The input file that should be grouped and summarized. *Use "stdin" when using piped input*. **Note: if -i is omitted, input is assumed to come from standard input (stdin)**
-g (-grp)	Specifies which column(s) (1-based) should be used to group the input. The columns must be comma-separated and each column must be explicitly listed. No ranges (e.g. 1-4) yet allowed. *Default: 1,2,3*
-c (-opCol)	Specify the column (1-based) that should be summarized. *Required*.
-o (-op)	Specify the operation that should be applied to **opCol**. Valid operations: **sum** - *numeric only* **count** - *numeric or text* **count_distinct** - *numeric or text* **min** - *numeric only* **max** - *numeric only* **mean** - *numeric only* **median** - *numeric only* **mode** - *numeric or text* **antimode** - *numeric or text* **stdev** - *numeric only* **sstdev** - *numeric only* **collapse** (i.e., print a comma separated list) - *numeric or text* **distinct** (i.e., print a comma separated list) - *numeric or text* **concat** (i.e., print a comma separated list) - *numeric or text* **freqasc** - *print a comma separated list of values observed and the number of times they were observed.* Reported in ascending order of frequency* **freqdesc** - *print a comma separated list of values observed and the number of times they were observed.* Reported in descending order of frequency* **first** - *numeric or text* **last** - *numeric or text* *Default: sum*

Default behavior.

Let's imagine we have three incredibly interesting genetic variants that we are studying and we are interested in what annotated repeats these variants overlap.

```
cat variants.bed
chr21  9719758 9729320 variant1
```

```
chr21  9729310 9757478 variant2
chr21  9795588 9796685 variant3

bedtools intersect -a variants.bed -b repeats.bed -wa -wb > variantsToRepeats.bed
cat variantsToRepeats.bed
chr21  9719758 9729320 variant1   chr21  9719768 9721892 ALR/Alpha   1004  +
chr21  9719758 9729320 variant1   chr21  9721905 9725582 ALR/Alpha   1010  +
chr21  9719758 9729320 variant1   chr21  9725582 9725977 L1PA3       3288  +
chr21  9719758 9729320 variant1   chr21  9726021 9729309 ALR/Alpha   1051  +
chr21  9729310 9757478 variant2   chr21  9729320 9729809 L1PA3       3897  -
chr21  9729310 9757478 variant2   chr21  9729809 9730866 L1P1        8367  +
chr21  9729310 9757478 variant2   chr21  9730866 9734026 ALR/Alpha   1036  -
chr21  9729310 9757478 variant2   chr21  9734037 9757471 ALR/Alpha   1182  -
chr21  9795588 9796685 variant3   chr21  9795589 9795713 (GAATG)n    308   +
chr21  9795588 9796685 variant3   chr21  9795736 9795894 (GAATG)n    683   +
chr21  9795588 9796685 variant3   chr21  9795911 9796007 (GAATG)n    345   +
chr21  9795588 9796685 variant3   chr21  9796028 9796187 (GAATG)n    756   +
chr21  9795588 9796685 variant3   chr21  9796202 9796615 (GAATG)n    891   +
chr21  9795588 9796685 variant3   chr21  9796637 9796824 (GAATG)n    621   +
```

We can see that variant1 overlaps with 3 repeats, variant2 with 4 and variant3 with 6. We can use bedtools groupby to summarize the hits for each variant in several useful ways. The default behavior is to compute the *sum* of the opCol.

```
bedtools groupby -i variantsToRepeats.bed -g 1,2,3 -c 9
chr21 9719758 9729320 6353
chr21 9729310 9757478 14482
chr21 9795588 9796685 3604
```

Computing the min and max.

Now let's find the *min* and *max* repeat score for each variant. We do this by "grouping" on the variant coordinate columns (i.e. cols. 1,2 and 3) and ask for the min and max of the repeat score column (i.e. col. 9).

```
bedtools groupby -i variantsToRepeats.bed -g 1,2,3 -c 9 -o min
chr21 9719758 9729320 1004
chr21 9729310 9757478 1036
chr21 9795588 9796685 308
```

We can also group on just the *name* column with similar effect.

```
bedtools groupby -i variantsToRepeats.bed -g 4 -c 9 -o min
variant1 1004
variant2 1036
variant3 308
```

What about the *max* score? Let's keep the coordinates and the name of the variants so that we stay in BED format.

```
bedtools groupby -i variantsToRepeats.bed -grp 1-4 -c 9 -o max
chr21 9719758 9729320 variant1 3288
chr21 9729310 9757478 variant2 8367
chr21 9795588 9796685 variant3 891
```

Computing the mean and median.

Now let's find the *mean* and *median* repeat score for each variant.

```
cat variantsToRepeats.bed | bedtools groupby -g 1-4 -c 9 -o mean
chr21 9719758 9729320 variant1 1588.25
chr21 9729310 9757478 variant2 3620.5
chr21 9795588 9796685 variant3 600.6667

bedtools groupby -i variantsToRepeats.bed -g 1-4 -c 9 -op median
chr21 9719758 9729320 variant1 1030.5
chr21 9729310 9757478 variant2 2539.5
chr21 9795588 9796685 variant3 652
```

Computing the mode and "antimode".

Now let's find the *mode* and *antimode* (i.e., the least frequent) repeat score for each variant (in this case they are identical).

```
bedtools groupby -i variantsToRepeats.bed -g 1-4 -c 9 -o mode
chr21 9719758 9729320 variant1 1004
chr21 9729310 9757478 variant2 1036
chr21 9795588 9796685 variant3 308

bedtools groupby -i variantsToRepeats.bed -g 1-4 -c 9 -o antimode
chr21 9719758 9729320 variant1 1004
chr21 9729310 9757478 variant2 1036
chr21 9795588 9796685 variant3 308
```

Computing the count of lines for a given group.

Figure:

```
bedtools groupby -i variantsToRepeats.bed -g 1-4 -c 9 -c count
chr21 9719758 9729320 variant1 4
chr21 9729310 9757478 variant2 4
chr21 9795588 9796685 variant3 6
```

Collapsing: listing all of the values in the opCol for a given group.

Now for something different. What if we wanted all of the names of the repeats listed on the same line as the variants? Use the collapse option. This "denormalizes" things. Now you have a list of all the repeats on a single line.

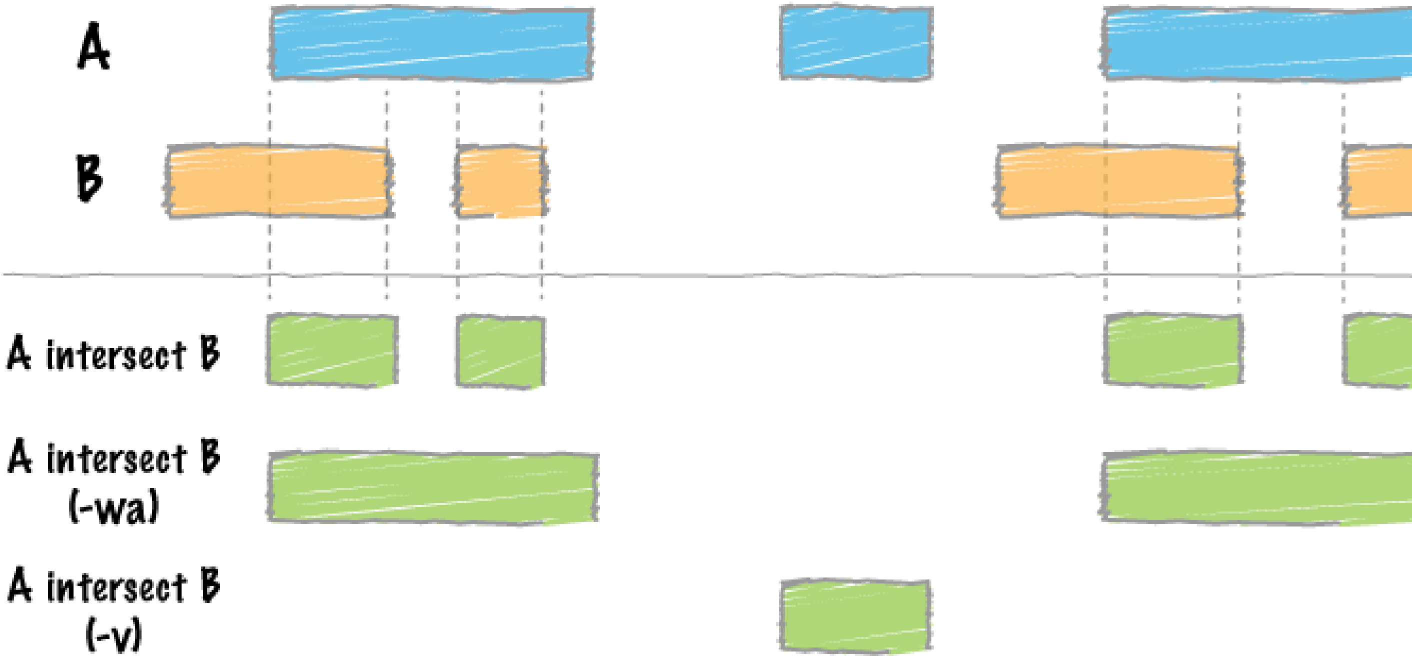
```
bedtools groupby -i variantsToRepeats.bed -grp 1-4 -c 9 -o collapse
chr21 9719758 9729320 variant1 ALR/Alpha,ALR/Alpha,L1PA3,ALR/Alpha,
chr21 9729310 9757478 variant2 L1PA3,L1P1,ALR/Alpha,ALR/Alpha,
chr21 9795588 9796685 variant3 (GAATG)n,(GAATG)n,(GAATG)n,(GAATG)n,(GAATG)n,(GAATG)n,
```

Computing frequencies: freqasc and freqdesc.

What if we want to report each distinct value along with its number of occurrence (much like uniq -c)? The `freqasc` and freqdesc'' operations handle this.

```
cat variantsToRepeats.bed | bedtools groupby -g 1 -c 8 -o freqdesc
chr21 (GAATG)n:6,ALR/Alpha:5,L1PA3:2,L1P1:1,
cat variantsToRepeats.bed | bedtools groupby -g 1 -c 8 -o freqasc
chr21 L1P1:1,L1PA3:2,ALR/Alpha:5,(GAATG)n:6,
```

intersect



By far, the most common question asked of two sets of genomic features is whether or not any of the features in the two sets "overlap" with one another. This is known as feature intersection. `bedtools intersect` allows one to screen for overlaps between two sets of genomic features. Moreover, it allows one to have fine control as to how the intersections are reported. `bedtools intersect` works with both BED/GFF/VCF and BAM files as input.

Note: If you are trying to intersect very large files and are having trouble with excessive memory usage, please presort your data by chromosome and then by start position (e.g., `sort k1,1 -k2,2n in.bed > in.sorted.bed` for BED files) and then use the `-sorted` option. This invokes a memory-efficient algorithm designed for large files.

See Also:

subtract window

Usage and option summary

Usage:

```
bedtools intersect [OPTIONS] [-a|-abam] -b <BED/GFF/VCF>
```

(or):

```
intersectBed [OPTIONS] [-a|-abam] -b <BED/GFF/VCF>
```

Option	Description
-a	BED/GFF/VCF file A. Each feature in A is compared to B in search of overlaps. Use "stdin" if passing A with a UNIX pipe.
-b	BED/GFF/VCF file B. Use "stdin" if passing B with a UNIX pipe.
-abam	BAM file A. Each BAM alignment in A is compared to B in search of overlaps. Use "stdin" if passing A with a UNIX pipe: For example: samtools view -b <BAM> | bedtools intersect -abam stdin -b genes.bed
-ubam	Write uncompressed BAM output. The default is write compressed BAM output.
-bed	When using BAM input (-abam), write output as BED. The default is to write output in BAM when using -abam. For example: `bedtools intersect -abam reads.bam -b genes.bed -bed`
-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.
-u	Write original A entry once if any overlaps found in B. In other words, just report the fact at least one overlap was found in B. Restricted by -f and -r.
-c	For each entry in A, report the number of hits in B while restricting to -f. Reports 0 for A entries that have no overlap with B. Restricted by -f and -r.
-v	Only report those entries in A that have no overlap in B. Restricted by -f and -r.
-f	Minimum overlap required as a fraction of A. 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.
-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.
-split	Treat "split" BAM (i.e., having an "N" CIGAR operation) or BED12 entries as distinct BED intervals.
-sorted	For very large B files, invoke a "sweeping" algorithm that requires position-sorted (e.g., `sort -k1,1 -k2,2n` for BED files) input. When using -sorted, memory usage remains low even for very large files.
-header	Print the header from the A file prior to results.

Default behavior

By default, if an overlap is found, `bedtools intersect` reports the shared interval between the two overlapping features.

```
$ cat A.bed
chr1  10  20
chr1  30  40

$ cat B.bed
chr1  15   20

$ bedtools intersect -a A.bed -b B.bed
chr1  15   20
```

`-wa` Reporting the original A feature

Instead, one can force `bedtools intersect` to report the *original* **"A"** feature when an overlap is found. As shown below, the entire "A" feature is reported, not just the portion that overlaps with the "B" feature.

For example:

```
$ cat A.bed
chr1  10  20
chr1  30   40

$ cat B.bed
chr1  15  20

$ bedtools intersect -a A.bed -b B.bed -wa
chr1  10   20
```

`-wb` Reporting the original B feature

Similarly, one can force `bedtools intersect` to report the *original* **"B"** feature when an overlap is found. If just -wb is used, the overlapping portion of A will be reported followed by the *original* **"B"**. If both -wa and -wb are used, the *originals* of both **"A"** and **"B"** will be reported.

For example (-wb alone):

```
$ cat A.bed
chr1  10  20
chr1  30  40

$ cat B.bed
chr1  15   20

$ bedtools intersect -a A.bed -b B.bed -wb
chr1  15  20  chr 15  20
```

Now -wa and -wb:

```
$ cat A.bed
chr1  10  20
chr1  30  40

$ cat B.bed
chr1  15   20

$ bedtools intersect -a A.bed -b B.bed -wa -wb
chr1  10  20  chr 15  20
```

`-loj` Left outer join. Report features in A with and without overlaps

By default, `bedtools intersect` will only report features in A that have an overlap in B. The `-loj` option will report every A feature no matter what. When there is an overlap (or more than 1), it will report A with its overlaps. Yet when there are no overlaps, an A feature will be reported with a NULL B feature to indicate that there were no overlaps

For example (*without* `-loj`):

```
$ cat A.bed
chr1  10  20
chr1  30  40

$ cat B.bed
chr1  15   20

$ bedtools intersect -a A.bed -b B.bed
chr1  10  20  chr 15  20
```

Now *with* `-loj`:

```
$ cat A.bed
chr1  10  20
chr1  30  40

$ cat B.bed
chr1  15   20

$ bedtools intersect -a A.bed -b B.bed -loj
chr1  10  20  chr 15  20
chr1  30  40  . -1  -1
```

`-wo` Write the *amount* of overlap between intersecting features

The `-wo` option reports a column after each combination of intersecting "A" and "B" features indicating the *amount* of overlap in bases pairs that is observed between the two features.

Note: When an interval in A does not intersect an interval in B, it will not be reported. If you would like to report such intervals with an overlap equal to 0, see the `-wao` option.

```
$ cat A.bed
chr1    10    20
chr1    30    40

$ cat B.bed
chr1    15  20
chr1    18  25

$ bedtools intersect -a A.bed -b B.bed -wo
chr1    10    20    chr1    15  20  5
chr1    10    20    chr1    18  25  2
```

`-wao` Write *amounts* of overlap for all features.

The `-wao` option extends upon the `-wo` option in that, unlike `-wo`, it reports an overlap of 0 for features in A that do not have an intersection in B.

```
$ cat A.bed
chr1    10    20
chr1    30    40

$ cat B.bed
chr1    15  20
chr1    18  25

$ bedtools intersect -a A.bed -b B.bed -wao
chr1    10    20    chr1    15  20  5
chr1    10    20    chr1    18  25  2
chr1    30    40    .       -1  -1  0
```

`-u` (unique) Reporting the mere presence of *any* overlapping features

Often you'd like to simply know a feature in "A" overlaps one or more features in B without reporting each and every intersection. The `-u` option will do exactly this: if an one or more overlaps exists, the A feature is reported. Otherwise, nothing is reported.

For example, without `-u`:

```
$ cat A.bed
chr1  10  20

$ cat B.bed
chr1  15  20
chr1  17  22

$ bedtools intersect -a A.bed -b B.bed
chr1  10   20
chr1  10   20
```

Now with `-u`:

```
$ cat A.bed
chr1  10  20

$ cat B.bed
chr1  15  20
chr1  17  22

$ bedtools intersect -a A.bed -b B.bed -u
chr1  10   20
```

`-c` Reporting the number of overlapping features

The -c option reports a column after each "A" feature indicating the *number* (0 or more) of overlapping features found in "B". Therefore, *each feature in A is reported once*.

```
$ cat A.bed
chr1    10    20
```

```
chr1    30    40

$ cat B.bed
chr1    15  20
chr1    18  25

$ bedtools intersect -a A.bed -b B.bed -c
chr1    10    20    2
chr1    30    40    0
```

`-v` Reporting the absence of any overlapping features

There will likely be cases where you'd like to know which "A" features do not overlap with any of the "B" features. Perhaps you'd like to know which SNPs don't overlap with any gene annotations. The `-v` (an homage to "grep -v") option will only report those "A" features that have no overlaps in "B".

```
$ cat A.bed
chr1  10  20
chr1  30  40

$ cat B.bed
chr1  15  20

$ bedtools intersect -a A.bed -b B.bed -v
chr1  30   40
```

`-f` Requiring a minimal overlap fraction

By default, `bedtools intersect` will report an overlap between A and B so long as there is at least one base pair is overlapping. Yet sometimes you may want to restrict reported overlaps between A and B to cases where the feature in B overlaps at least X% (e.g. 50%) of the A feature. The -f option does exactly this.

For example (note that the second B entry is not reported):

```
$ cat A.bed
chr1 100 200

$ cat B.bed
chr1 130 201
chr1 180 220

$ bedtools intersect -a A.bed -b B.bed -f 0.50 -wa -wb
chr1 100 200 chr1 130 201
```

`-r, and -f` Requiring reciprocal minimal overlap fraction

Similarly, you may want to require that a minimal fraction of both the A and the B features is overlapped. For example, if feature A is 1kb and feature B is 1Mb, you might not want to report the overlap as feature A can overlap at most 1% of feature B. If one set -f to say, 0.02, and one also enable the -r (reciprocal overlap fraction required), this overlap would not be reported.

For example (note that the second B entry is not reported):

```
$ cat A.bed
chr1 100 200

$ cat B.bed
chr1 130 201
chr1 130 200000

$ bedtools intersect -a A.bed -b B.bed -f 0.50 -r -wa -wb
chr1 100 200 chr1 130 201
```

`-s` Enforcing *same* strandedness

By default, `bedtools intersect` will report overlaps between features even if the features are on opposite strands. However, if strand information is present in both BED files and the "-s" option is used, overlaps will only be reported when features are on the same strand.

For example (note that the first B entry is not reported):

```
$ cat A.bed
chr1 100 200 a1 100 +

$ cat B.bed
chr1 130 201 b1 100 -
chr1 132 203 b2 100 +

$ bedtools intersect -a A.bed -b B.bed -wa -wb -s
chr1 100 200 a1 100 + chr1 132 203 b2 100 +
```

`-S` Enforcing *opposite* "strandedness"

The `-s` option enforces that overlaps be on the *same* strand. In some cases, you may want to enforce that overlaps be found on *opposite* strands. In this, case use the `-S` option.

For example:

```
$ cat A.bed
chr1 100 200 a1 100 +

$ cat B.bed
chr1 130 201 b1 100 -
chr1 132 203 b2 100 +

$ bedtools intersect -a A.bed -b B.bed -wa -wb -S
chr1 100 200 a1 100 + chr1 130 201 b1 100 -
```

`-abam` Default behavior when using BAM input

When comparing alignments in BAM format (**-abam**) to features in BED format (**-b**), `bedtools intersect` will, **by default**, write the output in BAM format. That is, each alignment in the BAM file that meets the user's criteria will be written (to standard output) in BAM format. This serves as a mechanism to create subsets of BAM alignments are of biological interest, etc. Note that only the mate in the BAM alignment is compared to the BED file. Thus, if only one end of a paired-end sequence overlaps with a feature in B, then that end will be written to the BAM output. By contrast, the other mate for the pair will not be written. One should use **pairToBed(Section 5.2)** if one wants each BAM alignment for a pair to be written to BAM output.

```
$ bedtools intersect -abam reads.unsorted.bam -b simreps.bed | \
       samtools view - | \
           head -3

BERTHA_0001:3:1:15:1362#0 99 chr4 9236904 0 50M = 9242033 5 1 7 9
AGACGTTAACTTTACACACCTCTGCCAAGGTCCTCATCCTTGTATTGAAG W c T U ] b \ g c e g X g f c b f c c b d d g g V
\c`dcdabdfW^a^gggfgd XT:A:R NM:i:0 SM:i:0 AM:i:0 X0:i:19 X1:i:2 XM:i:0 XO:i:0 XG:i:0 MD:Z:50
BERTHA _0001:3:1:16:994#0 83 chr6 114221672 37 25S6M1I11M7S =
114216196 -5493 G A A A G G C C A G A G T A T A G A A T A A A C A C A A C A A T G T C C A A G G T A
gffeaaddddggggggedgcgeggdegggggffcgggggggegdfggfgf XT:A:M NM:i:3 SM:i:37 AM:i:37 XM:i:2 X O : i :
1 XG:i:1 MD:Z:6A6T3
BERTHA _0001:3:1:16:594#0 147 chr8 43835330 0 50M =
43830893 -4487 CTTTGGGAGGGCTTTGTAGCCTATCTGGAAAAAGGAAATATCTTCCCATG U
\e^bgeTdg_Kgcg`ggeggg_ggggggggddgdggVg\gWdfgfgff XT:A:R NM:i:2 SM:i:0 AM:i:0 X0:i:10 X1:i:7 X M : i
2 XO:i:0 XG:i:0 MD:Z:1A2T45
```

`-ubam` Default behavior when using BAM input

The `-ubam` option writes *uncompressed* BAM output to stdout. This is useful for increasing the speed of pipelines that accept the output of `bedtools` intersect as input, since the receiving tool does not need to uncompress the data.

`-bed` Output BED format when using BAM input

When comparing alignments in BAM format (**-abam**) to features in BED format (**-b**), `bedtools intersect` will **optionally** write the output in BED format. That is, each alignment in the BAM file is converted to a 6 column BED feature and if overlaps are found (or not) based on the user's criteria, the BAM alignment will be reported in BED format. The BED "name" field is comprised of the RNAME field in the BAM alignment. If mate information is available, the mate (e.g., "/1" or "/2") field will be appended to the name. The "score" field is the mapping quality score from the BAM alignment.

```
$ bedtools intersect -abam reads.unsorted.bam -b simreps.bed -bed | head -20

chr4  9236903   9236953   BERTHA_0001:3:1:15:1362#0/1  0   +
chr6  114221671 114221721 BERTHA_0001:3:1:16:994#0/1   37  -
chr8  43835329  43835379  BERTHA_0001:3:1:16:594#0/2   0   -
chr4  49110668  49110718  BERTHA_0001:3:1:31:487#0/1   23  +
chr19 27732052  27732102  BERTHA_0001:3:1:32:890#0/2   46  +
chr19 27732012  27732062  BERTHA_0001:3:1:45:1135#0/1  37  +
chr10 117494252 117494302 BERTHA_0001:3:1:68:627#0/1   37  -
chr19 27731966  27732016  BERTHA_0001:3:1:83:931#0/2   9   +
chr8  48660075  48660125  BERTHA_0001:3:1:86:608#0/2   37  -
chr9  34986400  34986450  BERTHA_0001:3:1:113:183#0/2  37  -
chr10 42372771  42372821  BERTHA_0001:3:1:128:1932#0/1 3   -
chr19 27731954  27732004  BERTHA_0001:3:1:130:1402#0/2 0   +
chr10 42357337  42357387  BERTHA_0001:3:1:137:868#0/2  9   +
chr1  159720631 159720681 BERTHA_0001:3:1:147:380#0/2  37  -
chrX  58230155  58230205  BERTHA_0001:3:1:151:656#0/2  37  -
chr5  142612746 142612796 BERTHA_0001:3:1:152:1893#0/1 37  -
chr9  71795659  71795709  BERTHA_0001:3:1:177:387#0/1  37  +
chr1  106240854 106240904 BERTHA_0001:3:1:194:928#0/1  37  -
chr4  74128456  74128506  BERTHA_0001:3:1:221:724#0/1  37  -
chr8  42606164  42606214  BERTHA_0001:3:1:244:962#0/1  37  +
```

`-split` Reporting overlaps with spliced alignments or blocked BED features

As described in section 1.3.19, bedtools intersect will, by default, screen for overlaps against the entire span of a spliced/split BAM alignment or blocked BED12 feature. When dealing with RNA-seq reads, for example, one typically wants to only screen for overlaps for the portions of the reads that come from exons (and ignore the interstitial intron sequence). The **-split** command allows for such overlaps to be performed.

For example, the diagram below illustrates the *default* behavior. The blue dots represent the “split/ spliced” portion of the alignment (i.e., CIGAR “N” operation). In this case, the two exon annotations are reported as overlapping with the “split” BAM alignment, but in addition, a third feature that overlaps the “split” portion of the alignment is also reported.

```
Chromosome  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Exons       ---------------                                        ----------

BED/BAM  A     ************........................................****

BED File B  ^^^^^^^^^^^^^^^                      ^^^^^^^^          ^^^^^^^^^^

Result      ===============                      ========          ==========
```

In contrast, when using the **-split** option, only the exon overlaps are reported.

```
Chromosome  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Exons       ---------------                                        ----------

BED/BAM  A     ************........................................****

BED File B  ^^^^^^^^^^^^^^^                      ^^^^^^^^          ^^^^^^^^^^

Result      ===============                                        ==========
```

`-sorted` Invoke a memory-efficient algorithm for very large files.

The default algorithm for detecting overlaps loads the B file into an R-tree structure in memory. While fast, it can consume substantial memory for large files. For these reason, we provide an alternative, memory efficient algorithm that depends upon inout files that have been sorted by chromosome and then by start position. When both input files are position-sorted, the algorithm can “sweep” through the data and detect overlaps on the fly in a manner much like the way database systems join two tables. This option is invoked with the `-sorted` option.

For example:

```
$ sort -k1,1 -k2,2n big.bed > big.sorted.bed

$ sort -k1,1 -k2,2n huge.bed > huge.sorted.bed

$ bedtools intersect -a big.sorted.bed -b huge.sorted.bed -sorted
```

`-header` Print the header for the A file before reporting results.

By default, if your A file has a header, it is ignored when reporting results. This option will instead tell bedtools to first print the header for the A file prior to reporting results.

jaccard

links

Creates an HTML file with links to an instance of the UCSC Genome Browser for all features / intervals in a file. This is useful for cases when one wants to manually inspect through a large set of annotations or features.

Usage and option summary

Usage:

```
linksBed [OPTIONS] -i <BED/GFF/VCF> > <HTML file>
```

Option	Description
-base	The "basename" for the UCSC browser. *Default: http://genome.ucsc.edu*
-org	The organism (e.g. mouse, human). *Default: human*
-db	The genome build. *Default: hg18*

Default behavior

By default, **linksBed** creates links to the public UCSC Genome Browser.

For example:

```
head genes.bed
chr21 9928613  10012791  uc002yip.1 0  -
chr21 9928613  10012791  uc002yiq.1 0  -
chr21 9928613  10012791  uc002yir.1 0  -
chr21 9928613  10012791  uc010gkv.1 0  -
chr21 9928613  10061300  uc002yis.1 0  -
chr21 10042683 10120796  uc002yit.1 0  -
chr21 10042683 10120808  uc002yiu.1 0  -
chr21 10079666 10120808  uc002yiv.1 0  -
chr21 10080031 10081687  uc002yiw.1 0  -
chr21 10081660 10120796  uc002yix.2 0  -

linksBed -i genes.bed > genes.html
```

When genes.html is opened in a web browser, one should see something like the following, where each link on the page is built from the features in genes.bed:

Creating HTML links to a local UCSC Browser installation

Optionally, **linksBed** will create links to a local copy of the UCSC Genome Browser.

For example:

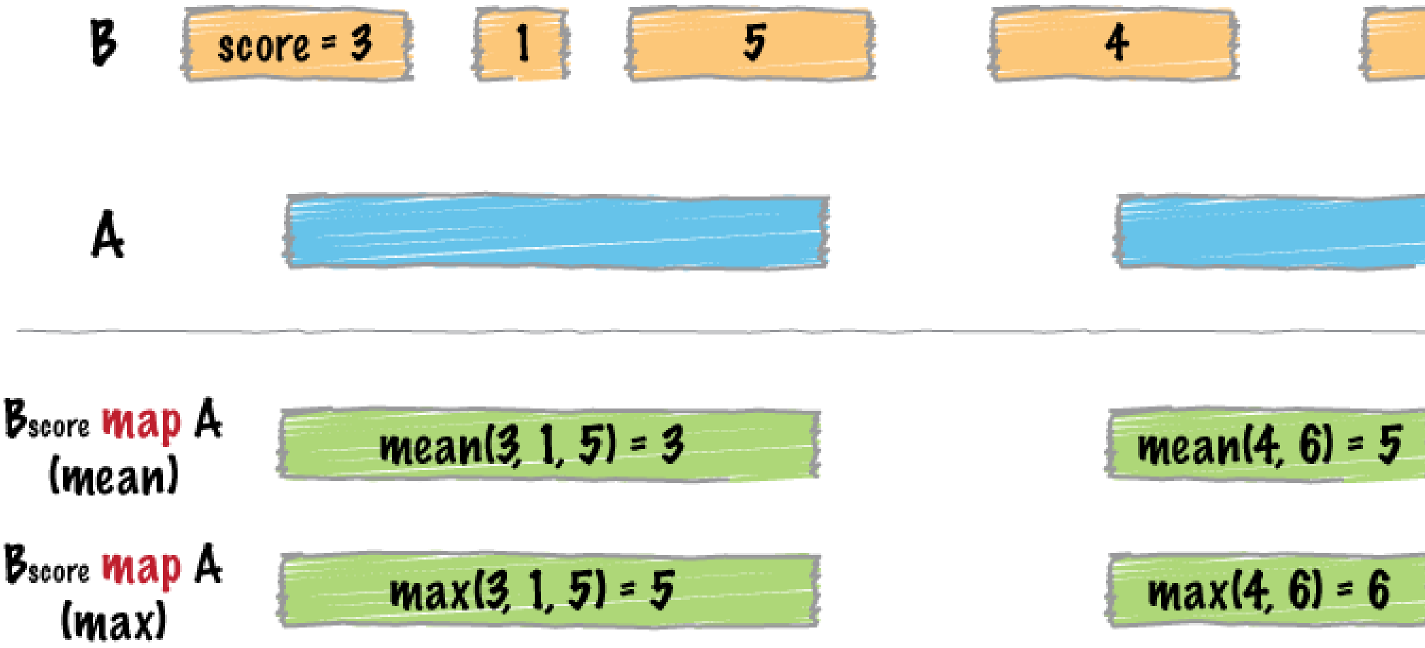
```
head -3 genes.bed
chr21 9928613 10012791 uc002yip.1 0 -
chr21 9928613 10012791 uc002yiq.1 0 -

linksBed -i genes.bed -base http://mirror.uni.edu > genes.html
```

One can point the links to the appropriate organism and genome build as well:

```
head -3 genes.bed
chr21 9928613 10012791 uc002yip.1 0 -
chr21 9928613 10012791 uc002yiq.1 0 -

linksBed -i genes.bed -base http://mirror.uni.edu -org mouse -db mm9 > genes.html
```

makewindows

map



`bedtools map` allows one to map overlapping features in a B file onto features in an A file and apply statistics and/or summary operations on those features.

For example, one could use `bedtools map` to compute the average score of BEDGRAPH records that overlap genes. Since the fourth column in BEDGRAPH is the score, the following command illustrates how this would be done:

```
$ bedtools map -a genes.bed -b peaks.bedgraph -c 4 -o mean
```

Another example is discussed in this Biostars post.

Note: `bedtools map` requires each input file to be sorted by genome coordinate. For BED files, this can be done with `sort -k1,1 -k2,2n`.

Usage and option summary

Usage:

```
bedtools map [OPTIONS] -a <bed/gff/vcf> -b <bed/gff/vcf>
```

(or):

```
mapBed [OPTIONS] -a <bed/gff/vcf> -b <bed/gff/vcf>
```

Option	Description
-c	Specify the column from the B file to map onto intervals in A. `Default: 5`
-o	Specify the operation that should be applied to `-c`. Valid operations: **sum** - *numeric only* **count** - *numeric or text* **count_distinct** - *numeric or text* **min** - *numeric only* **max** - *numeric only* **mean** - *numeric only* **median** - *numeric only* **antimode** - *numeric or text* **collapse** (i.e., print a comma separated list) - *numeric or text* **distinct** (i.e., print a comma separated list) - *numeric or text* **concat** (i.e., print a comma separated list) - *numeric or text* `Default: 5`
-f	Minimum overlap required as a fraction of A. 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.
-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.
-null	The value to print if no overlaps are found for an A interval. `Default: "."`
-header	Print the header from the A file prior to results.

Default behavior - compute the `sum` of the `score` column for all overlaps.

By default, `map` computes the sum of the 5th column (the `score` field for BED format) for all intervals in B that overlap each interval in A.

Tip: Records in A that have no overlap will, by default, return `.` for the computed value from B. This can be changed with the `-null` option.

```
$ cat a.bed
chr1        10      20      a1      1       +
chr1        50      60      a2      2       -
chr1        80      90      a3      3       -

$ cat b.bed
chr1        12      14      b1      2       +
chr1        13      15      b2      5       -
chr1        16      18      b3      5       +
chr1        82      85      b4      2       -
chr1        85      87      b5      3       +

$ bedtools map -a a.bed -b b.bed
chr1        10      20      a1      1       +       12
chr1        50      60      a2      2       -       .
chr1        80      90      a3      3       -       5
```

`mean` Compute the mean of a column from overlapping intervals

```
$ cat a.bed
chr1        10      20      a1      1       +
chr1        50      60      a2      2       -
chr1        80      90      a3      3       -

$ cat b.bed
chr1        12      14      b1      2       +
chr1        13      15      b2      5       -
chr1        16      18      b3      5       +
chr1        82      85      b4      2       -
chr1        85      87      b5      3       +

$ bedtools map -a a.bed -b b.bed -c 5 -o mean
chr1        10      20      a1      1       +       4
chr1        50      60      a2      2       -       .
chr1        80      90      a3      3       -       2.5
```

`collapse` List each value of a column from overlapping intervals

```
$ bedtools map -a a.bed -b b.bed -c 5 -o collapse
chr1        10      20      a1      1       +       2,5,5
chr1        50      60      a2      2       -       .
chr1        80      90      a3      3       -       2,3
```

`distinct` List each *unique* value of a column from overlapping intervals

```
$ bedtools map -a a.bed -b b.bed -c 5 -o distinct
chr1        10      20      a1      1       +       2,5
chr1        50      60      a2      2       -       .
chr1        80      90      a3      3       -       2,3
```

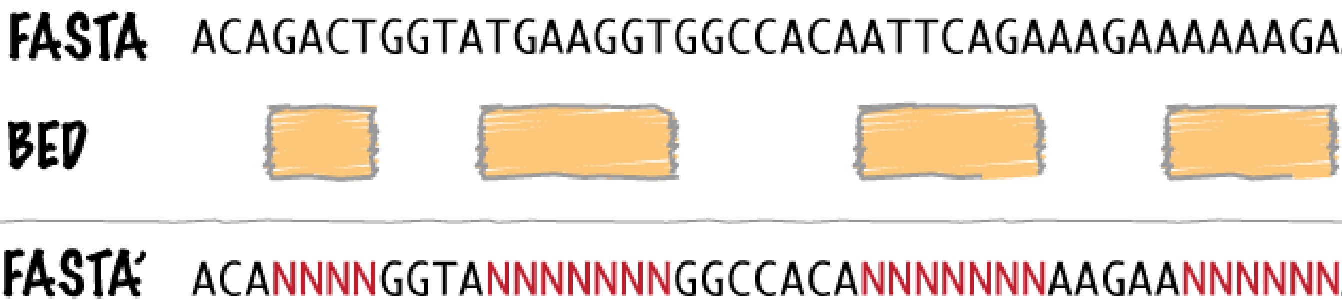
`-s` Only include intervals that overlap on the *same* strand.

```
$ bedtools map -a a.bed -b b.bed -c 5 -o collapse -s
chr1        10      20      a1      1       +       2,5
chr1        50      60      a2      2       -       .
chr1        80      90      a3      3       -       2
```

`-S` Only include intervals that overlap on the *opposite* strand.

```
$ bedtools map -a a.bed -b b.bed -c 5 -o collapse -S
chr1        10      20      a1      1       +       5
chr1        50      60      a2      2       -       .
chr1        80      90      a3      3       -       3
```

maskfasta


FASTA
ACAGACTGGTATGAAGGTGGCCACAATTCAGAAAGAAAAAGA
BED
FASTA'
ACANNNNGGTANNNNNNNGGCCACANNNNNNNAAGAANNNNNN

`bedtools maskfasta` masks sequences in a FASTA file based on intervals defined in a feature file. The headers in the input FASTA file must exactly match the chromosome column in the feature file. This may be useful fro creating your own masked genome file based on custom annotations or for masking all but your target regions when aligning sequence data from a targeted capture experiment.

Usage and option summary

Usage

```
$ bedtools maskfasta [OPTIONS] -fi <input FASTA> -bed <BED/GFF/VCF> -fo <output FASTA>
```

(or):

```
$ maskFastaFromBed [OPTIONS] -fi <input FASTA> -bed <BED/GFF/VCF> -fo <output FASTA>
```

Note: The input (`-fi`) and output (`-fo`) FASTA files must be different.

See Also:

getfasta

Option	Description
-soft	Soft-mask (that is, convert to lower-case bases) the FASTA sequence. *By default, hard-masking (that is, conversion to Ns) is performed.*
-mc	Replace masking character. That is, instead of masking with Ns, use another character.

Default behavior

bedtools maskfasta will mask a FASTA file based on the intervals in a BED file. The newly masked FASTA file is written to the output FASTA file.

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1 5 10

$ bedtools maskfasta -fi test.fa -bed test.bed -fo test.fa.out

$ cat test.fa.out
>chr1
AAAAANNNNNCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG
```

`-soft` Soft-masking the FASTA file.

Using the **-soft** option, one can optionally "soft-mask" the FASTA file.

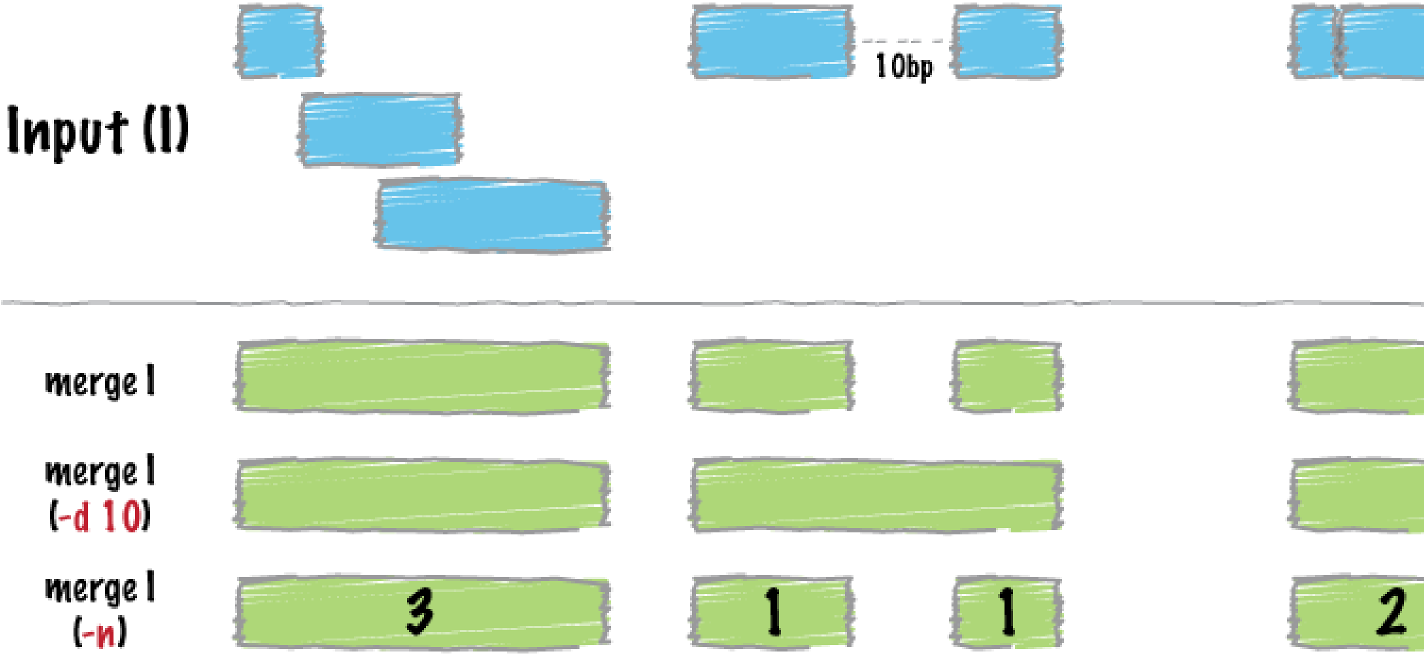
```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1 5 10

$ bedtools maskfasta -fi test.fa -bed test.bed -fo test.fa.out -soft

$ cat test.fa.out
>chr1
AAAAAaaaccCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG
```

`-mc` Specify a masking character.

Using the **-mc** option, one can optionally choose a masking character to each base that will be masked by the BED file.

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1 5 10

$ bedtools maskfasta -fi test.fa -bed test.bed -fo test.fa.out -mc X

$ cat test.fa.out
>chr1
AAAAAXXXXXCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG
```

merge



`bedtools merge` combines overlapping or "book-ended" features in an interval file into a single feature which spans all of the combined features.

Note: `bedtools merge` requires that you presort your data by chromosome and then by start position (e.g., `sort k1,1 -k2,2n in.bed > in.sorted.bed` for BED files).

See Also:

cluster complement

Usage and option summary

Usage:

```
bedtools merge [OPTIONS] -i <BED/GFF/VCF>
```

(or):

```
mergeBed [OPTIONS] -i <BED/GFF/VCF>
```

Option	Description
-s	Force strandedness. That is, only merge features that are the same strand. *By default, this is disabled.*
-n	Report the number of BED entries that were merged. *1 is reported if no merging occurred.*
-d	Maximum distance between features allowed for features to be merged. *Default is 0. That is, overlapping and/or book-ended features are merged.*
-nms	Report the names of the merged features separated by semicolons.
-scores	Report the scores of the merged features. Specify one of | the following options for reporting scores: | sum, min, max, | mean, median, mode, antimode, | collapse (i.e., print a semicolon-separated list)

Default behavior

By default, `bedtools merge` combines overlapping (by at least 1 bp) and/or bookended intervals into a single, “flattened” or “merged” interval.

```
$ cat A.bed
chr1  100  200
chr1  180  250
chr1  250  500
chr1  501  1000

$ bedtools merge -i A.bed
chr1  100  500
chr1  501  1000
```

`-s` Enforcing “strandedness”

The `-s` option will only merge intervals that are overlapping/bookended *and* are on the same strand.

```
$ cat A.bed
chr1  100  200   a1  1 +
chr1  180  250   a2  2 +
chr1  250  500   a3  3 -
chr1  501  1000  a4  4 +

$ bedtools merge -i A.bed -s
chr1  100  250    +
chr1  501  1000   +
chr1  250  500    -
```

`-n` Reporting the number of features that were merged

The -n option will report the number of features that were combined from the original file in order to make the newly merged feature. If a feature in the original file was not merged with any other features, a "1" is reported.

```
$ cat A.bed
chr1  100  200
chr1  180  250
chr1  250  500
chr1  501  1000

$ bedtools merge -i A.bed -n
chr1  100  500  3
chr1  501  1000 1
```

`-d` Controlling how close two features must be in order to merge

By default, only overlapping or book-ended features are combined into a new feature. However, one can force `merge` to combine more distant features with the `-d` option. For example, were one to set `-d` to 1000, any features that overlap or are within 1000 base pairs of one another will be combined.

```
$ cat A.bed
chr1  100  200
chr1  501  1000

$ bedtools merge -i A.bed
chr1  100  200
chr1  501  1000

$ bedtools merge -i A.bed -d 1000
chr1  100  200  1000
```

`-nms` Reporting the names of the features that were merged

Occasionally, one might like to know that names of the features that were merged into a new feature. The `-nms` option will add an extra column to the `merge` output which lists (separated by semicolons) the names of the merged features.

```
$ cat A.bed
chr1  100  200  A1
chr1  150  300  A2
chr1  250  500  A3
```

```
$ bedtools merge -i A.bed -nms
chr1  100  500  A1;A2;A3
```

`-scores` Reporting the scores of the features that were merged

Similarly, we might like to know that scores of the features that were merged into a new feature. Enter the `-scores` option. One can specify how the scores from each overlapping interval should be reported.

```
$ cat A.bed
chr1  100  200  A1 1
chr1  150  300  A2 2
chr1  250  500  A3 3

$ bedtools merge -i A.bed -scores mean
chr1  100  500  2

$ bedtools merge -i A.bed -scores max
chr1  100  500  3

$ bedtools merge -i A.bed -scores collapse
chr1  100  500  1,2,3
```

multicov

`bedtools multicov`, reports the count of alignments from multiple position-sorted and indexed BAM files that overlap intervals in a BED file. Specifically, for each BED interval provided, it reports a separate count of overlapping alignments from each BAM file.

Note: `bedtools multicov` depends upon index BAM files in order to count the number of overlaps in each BAM file. As such, each BAM file should be position sorted (`samtool sort aln.bam aln.sort`) and indexed (`samtools index aln.sort.bam`) with either samtools or bamtools.

Usage and option summary

Usage:

```
bedtools multicov [OPTIONS] -bams BAM1 BAM2 BAM3 ... BAMn -bed  <BED/GFF/VCF>
```

(or):

```
multiBamCov [OPTIONS] -bams BAM1 BAM2 BAM3 ... BAMn -bed  <BED/GFF/VCF>
```

Option	Description
-split	Treat “split” BAM or BED12 entries as distinct BED intervals.
-s	Require same 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.
-f	Minimum overlap required as a fraction of each A. Default is 1E-9 (i.e., 1bp).
-r	Require that the fraction overlap be reciprocal for A and B. In other words, if -f is 0.90 and -r is used, this requires that B overlap 90% of A and A _also_ overlaps 90% of B.
-q	Minimum mapping quality (MAPQ) allowed. Default is 0.
-D	Include duplicate reads. Default counts non-duplicates only
-F	Include failed-QC reads. Default counts pass-QC reads only
-p	Only count proper pairs. Default counts all alignments with `MAPQ > -q` argument, regardless of the BAM FLAG field.

Default behavior.

By default, `multicov` will report the count of alignments in each input BAM file that overlap.

```
$ cat ivls-of-interest.bed
chr1 0   10000   ivl1
chr1 10000   20000   ivl2
chr1 20000   30000   ivl3
chr1 30000   40000   ivl4

$ bedtools multicov -bams aln1.bam aln2.bam aln3.bam -bed ivls-of-interest.bed
chr1 0       10000   ivl1    100 2234    0
chr1 10000   20000   ivl2    123 3245    1000
chr1 20000   30000   ivl3    213 2332    2034
chr1 30000   40000   ivl4    335 7654    0
```

The output of `multicov` reflects a distinct report of the overlapping alignments for each record in the `-bed` file. In the example above, each line of the output reflects **a)** the original line from the `-bed` file followed by **b)** the count of alignments that overlap the `-bed` interval from each input `-bam` file. In the example above, the output consists of 7 columns: the first four of which are the columns from the `-bed` file and the last 3 are the count of overlapping alignments from the 3 input `-bam` files. The order of the counts reflects the order of the files given on the command line.

Note: `bedtools multicov` will work with a single BAM as well.

```
$ bedtools multicov -bams aln1.bam -bed ivls-of-interest.bed
chr1 0       10000   ivl1    100
chr1 10000   20000   ivl2    123
chr1 20000   30000   ivl3    213
chr1 30000   40000   ivl4    335
```

multiinter

nuc

overlap

overlap computes the amount of overlap (in the case of positive values) or distance (in the case of negative values) between feature coordinates occurring on the same input line and reports the result at the end of the same line. In this way, it is a useful method for computing custom overlap scores from the output of other BEDTools.

Usage and option summary

Usage:

```
overlap [OPTIONS] -i <input> -cols s1,e1,s2,e2
```

Option	Description
-i	Input file. Use "stdin" for pipes.
-cols	Specify the columns (1-based) for the starts and ends of the features for which you'd like to compute the overlap/distance. The columns must be listed in the following order: *start1,end1,start2,end2*

Default behavior

The default behavior is to compute the amount of overlap between the features you specify based on the start and end coordinates. For example:

```
windowBed -a A.bed -b B.bed -w 10
chr1  10  20  A  chr1  15  25  B
chr1  10  20  C  chr1  25  35  D
```

# Now let's say we want to compute the number of base pairs of overlap # between the overlapping features from the output of windowBed.

```
windowBed -a A.bed -b B.bed -w 10 | overlap -i stdin -cols 2,3,6,7
chr1  10  20  A  chr1  15  25  B  5
chr1  10  20  C  chr1  25  35  D  -5
```

pairtobed

pairtopair

pairToPair compares two BEDPE files in search of overlaps where each end of a BEDPE feature in A overlaps with the ends of a feature in B. For example, using pairToPair, one could screen for the exact same discordant paired-end alignment in two files. This could suggest (among other things) that the discordant pair suggests the same structural variation in each file/sample.

Usage and option summary

Usage:

```
pairToPair [OPTIONS] -a <BEDPE> -b <BEDPE>
```

Option	Description
-a	BEDPE file A. Each feature in A is compared to B in search of overlaps. Use “stdin” if passing A with a UNIX pipe.
-b	BEDPE file B. Use “stdin” if passing B with a UNIX pipe.
-f	Minimum overlap required as a fraction of A. Default is 1E-9 (i.e. 1bp).
-is	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.
-type	Approach to reporting overlaps between BEDPE and BED. **either** Report overlaps if either ends of A overlap B. **neither** Report A if neither end of A overlaps B. **both** Report overlaps if both ends of A overlap B. -*Default behavior.*

Default behavior

By default, a BEDPE feature from A will be reported if *both* ends overlap a feature in the BEDPE B file. If strand information is present for the two BEDPE files, it will be further required that the overlaps on each end be on the same strand. This way, an otherwise overlapping (in terms of genomic locations) F/R alignment will not be matched with a R/R alignment.

Default: Report A if *both* ends overlaps B.

```
Chromosome  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

BEDPE/BAM A         *****.................................*****

BED File B         ^^^^^^^^                                          ^^^^^^

Result              =====.................................=====
```

Default when strand information is present in both BEDPE files: Report A if *both* ends overlaps B *on the same strands*.

```
Chromosome  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

BEDPE A         >>>>>.................................>>>>>

BEDPE B            <<<<<.............................>>>>>

Result



BEDPE A         >>>>>.................................>>>>>
```

```
BEDPE B            >>>>>.............................>>>>>

Result          >>>>>.................................>>>>>
```

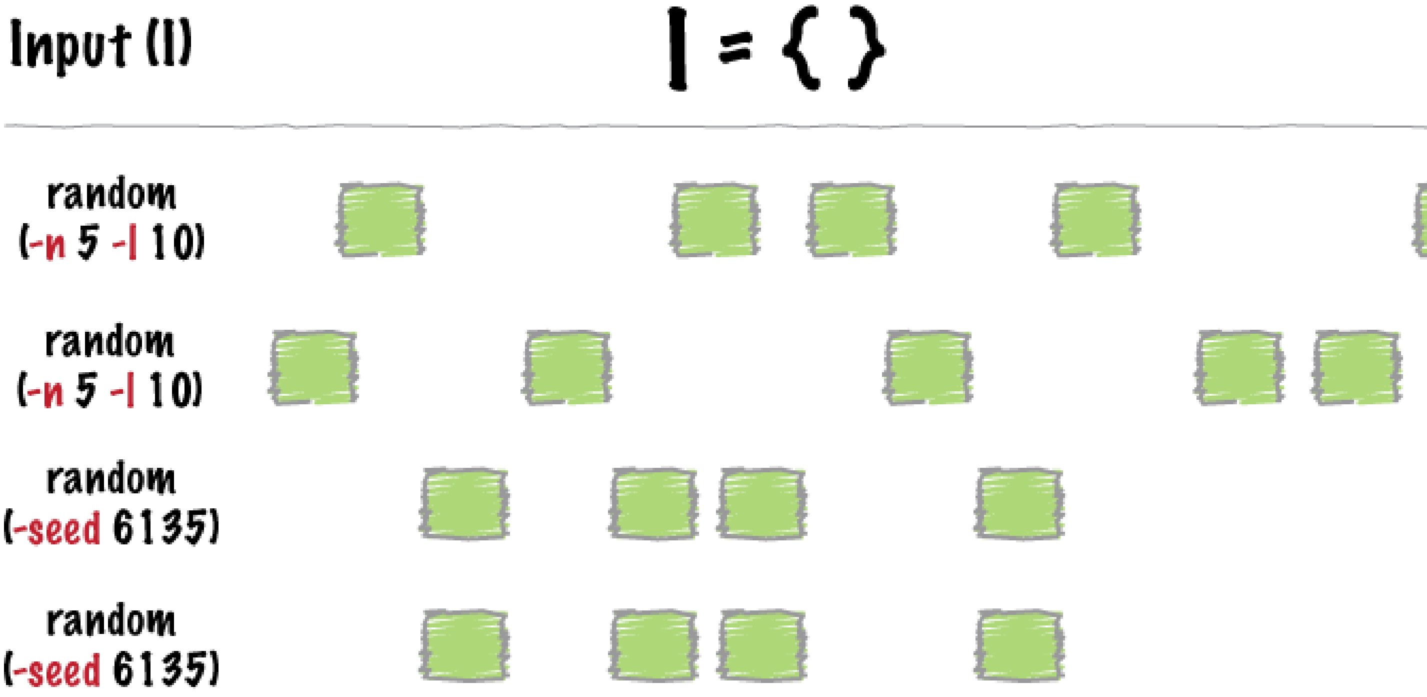
`-type neither` Optional overlap requirements

Using then **-type neither, pairToPair** will only report A if *neither* end overlaps with a BEDPE feature in B.

-type neither: Report A only if *neither* end overlaps B.

```
Chromosome  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

BEDPE/BAM A         *****.................................*****

BED File B         ^^^^^^^^......................................^^^^^^

Result


BEDPE/BAM A         *****.................................*****

BED File B    ^^^^................................................^^^^^^

Result              =====.................................=====
```

random



bedtools random will generate a random set of intervals in BED6 format. One can specify both the number (`-n`) and the size (`-l`) of the intervals that should be generated.

See Also:

shuffle jaccard

Usage and option summary

Usage:

```
bedtools random [OPTIONS] -g <GENOME>
```

(or):

```
randomBed [OPTIONS] -g <GENOME>
```

Option	Description
-l	The length of the intervals to generate. `Default = 100`
-n	The number of intervals to generate. `Default = 1,000,000`
-seed	Supply an integer seed for the shuffling. This will allow feature shuffling experiments to be recreated exactly as the seed for the pseudo-random number generation will be constant. *By default, the seed is chosen automatically*.

Default behavior

By default, *bedtools random* generate 1 million intervals of length 100 placed randomly in the genome specificed with `-g`.

```
$ bedtools random -g hg19.genome
chr2  87536758        87536858        1       100     -
chrX  46051735        46051835        2       100     +
chr18 5237041 5237141 3       100     -
chr12 45809998        45810098        4       100     +
chrX  42034890        42034990        5       100     -
chr10 77510935        77511035        6       100     -
chr3  39844278        39844378        7       100     -
chr6  101012700       101012800       8       100     +
chr12 38123482        38123582        9       100     +
chr7  88508598        88508698        10      100     -

$ bedtools random -g hg19.genome
chr3  141987850       141987950       1       100     +
chr5  137643331       137643431       2       100     +
chr2  155523858       155523958       3       100     -
chr5  147874094       147874194       4       100     +
chr1  71838335        71838435        5       100     -
chr8  71154323        71154423        6       100     -
chr2  133240474       133240574       7       100     +
chr9  131495427       131495527       8       100     +
chrX  125952943       125953043       9       100     +
chr3  59685545        59685645        10      100     +
```

`-n` Specify the *number* of intervals to generate.

The *-n* option allows one to override the default of generating 1 million intervals.

```
$ bedtools random -g hg19.genome -n 3
chr20 47975280        47975380        1       100     -
chr16 23381222        23381322        2       100     +
chr3  104913816       104913916       3       100     -
```

`-l` Specify the *length* of intervals to generate.

The *-l* option allows one to override the default interval length of 100bp.

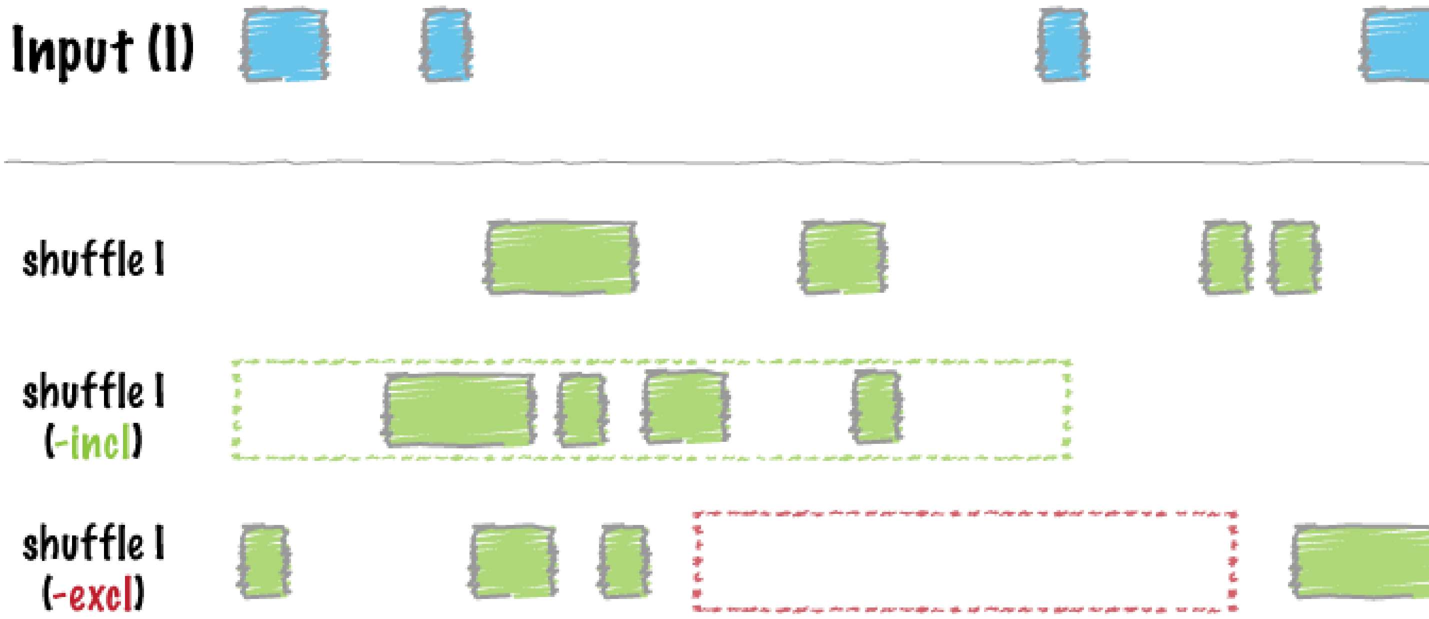
```
$ bedtools random -g hg19.genome -l 5
chr9  54133731        54133736        1       5       +
chr1  235288830       235288835       2       5       -
chr8  26744718        26744723        3       5       +
chr3  187313616       187313621       4       5       -
chr11 88996846        88996851        5       5       -
chr13 84714855        84714860        6       5       -
chr13 10759738        10759743        7       5       -
chr6  122569739       122569744       8       5       +
chr17 50884025        50884030        9       5       -
chr11 38576901        38576906        10      5       +
```

`-seed` Defining a "seed" for the random interval creation.

bedtools random uses a pseudo-random number generator to permute the locations of BED features. Therefore, each run should produce a different result. This can be problematic if one wants to exactly recreate an experiment. By using the *seed* option, one can supply a custom integer seed for *bedtools random*. In turn, each execution of *bedtools random* with the same seed and input files should produce identical results.

```
$ bedtools random -g hg19.genome -seed 71346
chrY  23380696        23380796        1       100     -
chr14 94368315        94368415        2       100     +
chr14 45353323        45353423        3       100     -
chr14 100546766       100546866       4       100     -
chr12 43294368        43294468        5       100     -
chr1  141470585       141470685       6       100     -
chr10 31273665        31273765        7       100     +
chr5  19102979        19103079        8       100     +
chr3  116730634       116730734       9       100     -
chr3  101222965       101223065       10      100     -

# (same seed, thus same as above)
$ bedtools random -g hg19.genome -seed 71346
chrY  23380696        23380796        1       100     -
chr14 94368315        94368415        2       100     +
chr14 45353323        45353423        3       100     -
chr14 100546766       100546866       4       100     -
chr12 43294368        43294468        5       100     -
chr1  141470585       141470685       6       100     -
chr10 31273665        31273765        7       100     +
chr5  19102979        19103079        8       100     +
chr3  116730634       116730734       9       100     -
chr3  101222965       101223065       10      100     -
```

shuffle



bedtools shuffle will randomly permute the genomic locations of a feature file among a genome defined in a genome file. One can also provide an "exclusions" BED/GFF/VCF file that lists regions where you do not want the permuted features to be placed. For example, one might want to prevent features from being placed in known genome gaps. *shuffle* is useful as a *null* basis against which to test the significance of associations of one feature with another.

See Also:

random jaccard

Usage and option summary

Usage:

```
bedtools shuffle [OPTIONS] -i <BED/GFF/VCF> -g <GENOME>
```

(or):

```
shuffleBed [OPTIONS] -i <BED/GFF/VCF> -g <GENOME>
```

Option	Description
-excl	A BED file of coordinates in which features from -i should *not* be placed (e.g., genome gaps).
-incl	A BED file of coordinates in which features from -i *should* be placed.
-chrom	Keep features in -i on the same chromosome. Solely permute their location on the chromosome. *By default, both the chromosome and position are randomly chosen.*
-seed	Supply an integer seed for the shuffling. This will allow feature shuffling experiments to be recreated exactly as the seed for the pseudo-random number generation will be constant. *By default, the seed is chosen automatically.*
-f	Maximum overlap (as a fraction of the `-i` feature) with an `-excl` feature that is tolerated before searching for a new, randomized locus.
-chromFirst	Instead of choosing a position randomly among the entire genome (the default), first choose a chrom randomly, and then choose a random start coordinate on that chrom. This leads to features being ~uniformly distributed among the chroms, as opposed to features being distribute as a function of chrom size.
-bedpe	Indicate that the A file is in BEDPE format.
-maxTries	Max. number of attempts to find a home for a shuffled interval in the presence of -incl or -excl. *Default = 1000.*

Default behavior

By default, *bedtools shuffle* will reposition each feature in the input BED file on a random chromosome at a random position. The size and strand of each feature are preserved.

For example:

```
$ cat A.bed
chr1  0  100  a1  1  +
chr1  0  1000 a2  2  -

$ cat my.genome
chr1  10000
chr2  8000
chr3  5000
chr4  2000

$ bedtools shuffle -i A.bed -g my.genome
chr4  1498  1598  a1  1  +
chr3  2156  3156  a2  2  -
```

`-chrom` Requiring that features be shuffled on the same chromosome

The *-chrom* option behaves the same as the default behavior except that features are randomly placed on the same chromosome as defined in the BED file.

```
$ cat A.bed
chr1  0  100  a1  1  +
chr1  0  1000 a2  2  -

$ cat my.genome
chr1  10000
chr2  8000
chr3  5000
chr4  2000
```

```
$ bedtools shuffle -i A.bed -g my.genome -chrom
chr1  9560  9660  a1  1  +
chr1  7258  8258  a2  2  -
```

`-excl` Excluding certain genome regions from `bedtools shuffle`

One may want to prevent BED features from being placed in certain regions of the genome. For example, one may want to exclude genome gaps from permutation experiment. The *excl* option defines a BED file of regions that should be excluded. `bedtools shuffle` will attempt to permute the locations of all features while adhering to the exclusion rules. However it will stop looking for an appropriate location if it cannot find a valid spot for a feature after 1,000,000 tries.

For example (*note that the exclude file excludes all but 100 base pairs of the chromosome*):

```
$ cat A.bed
chr1  0  100   a1  1  +
chr1  0  1000  a2  2  -

$ cat my.genome
chr1  10000

$ cat exclude.bed
chr1  100  10000

$ bedtools shuffle -i A.bed -g my.genome -excl exclude.bed
chr1  0  100  a1  1  +
Error, line 2: tried 1000000 potential loci for entry, but could not avoid excluded
regions. Ignoring entry and moving on.
```

For example (*now the exclusion file only excludes the first 100 bases of the chromosome*):

```
$ cat A.bed
chr1  0  100  a1  1  +
chr1  0  1000 a2  2  -

$ cat my.genome
chr1  10000

$ cat exclude.bed
chr1  0  100

$ bedtools shuffle -i A.bed -g my.genome -excl exclude.bed
chr1  147  247  a1  1  +
chr1  2441 3441 a2  2  -
```

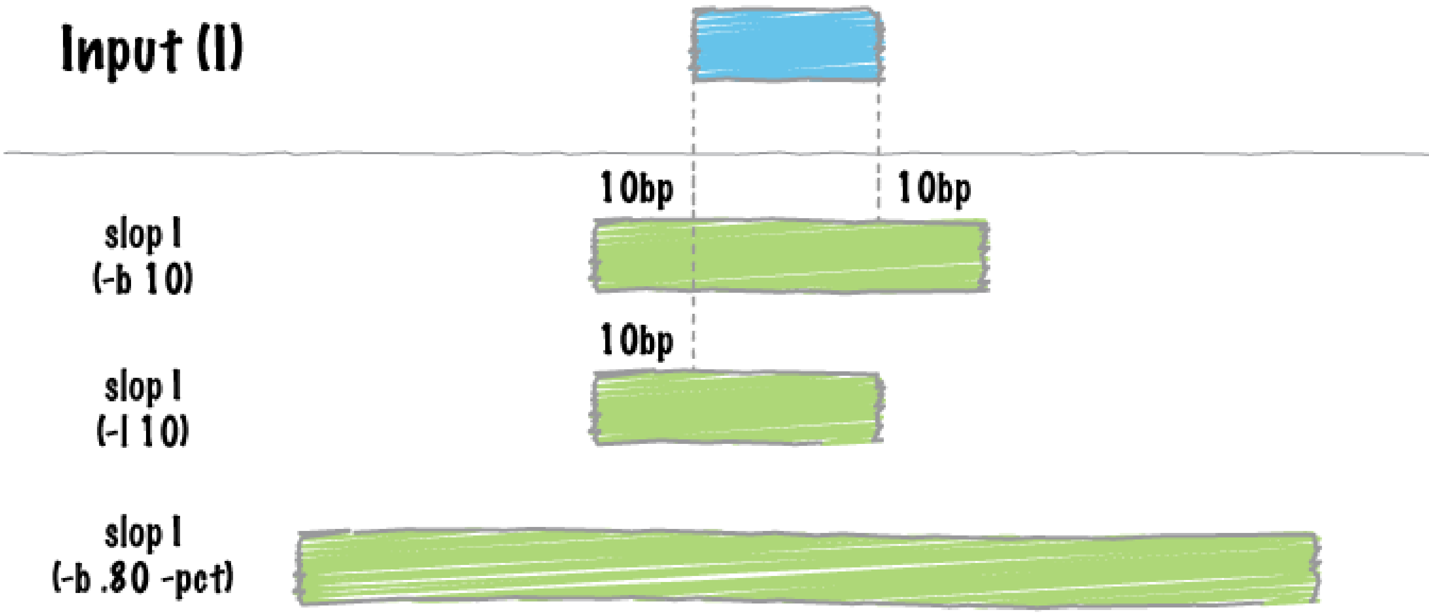
`-seed` Defining a "seed" for the random replacement.

bedtools shuffle uses a pseudo-random number generator to permute the locations of BED features. Therefore, each run should produce a different result. This can be problematic if one wants to exactly recreate an experiment. By using the *seed* option, one can supply a custom integer seed for *bedtools shuffle*. In turn, each execution of *bedtools shuffle* with the same seed and input files should produce identical results.

For example (*note that the exclude file below excludes all but 100 base pairs of the chromosome*):

```
$ cat A.bed
chr1 0 100 a1 1 +
chr1 0 1000 a2 2 -

$ cat my.genome
chr1 10000

$ bedtools shuffle -i A.bed -g my.genome -seed 927442958
chr1 6177 6277 a1 1 +
chr1 8119 9119 a2 2 -

$ bedtools shuffle -i A.bed -g my.genome -seed 927442958
chr1 6177 6277 a1 1 +
chr1 8119 9119 a2 2 -

. . .

$ bedtools shuffle -i A.bed -g my.genome -seed 927442958
chr1 6177 6277 a1 1 +
chr1 8119 9119 a2 2 -
```

slop



`bedtools slop` will increase the size of each feature in a feature file be a user-defined number of bases. While something like this could be done with an `awk '{OFS="\t" print $1,$2-<slop>,$3+<slop>}'`, `bedtools slop` will restrict the resizing to the size of the chromosome (i.e. no start < 0 and no end > chromosome size).

Note: In order to prevent the extension of intervals beyond chromosome boundaries, `bedtools slop` requires a *genome* file defining the length of each chromosome or contig.

Usage and option summary

Usage:

```
bedtools slop [OPTIONS] -i <BED/GFF/VCF> -g <GENOME> [-b or (-l and -r)]
```

(or):

```
bedtools slop [OPTIONS] -i <BED/GFF/VCF> -g <GENOME> [-b or (-l and -r)]
```

Option	Description
-b	Increase the BED/GFF/VCF entry by the same number base pairs in each direction. *Integer*.
-l	The number of base pairs to subtract from the start coordinate. *Integer*.
-r	The number of base pairs to add to the end coordinate. *Integer*.
-s	Define -l and -r based on strand. For example. if used, -l 500 for a negative-stranded feature, it will add 500 bp to the *end* coordinate.
-pct	Define -l and -r as a fraction of the feature's length. E.g. if used on a 1000bp feature, -l 0.50, will add 500 bp "upstream". Default = false.
-header	Print the header from the input file prior to results.

Default behavior

By default, `bedtools slop` will either add a fixed number of bases in each direction (`-b`) or an asymmetric number of bases in each direction with `-l` and `-r`.

```
$ cat A.bed
chr1 5 100
chr1 800 980

$ cat my.genome
chr1 1000

$ bedtools slop -i A.bed -g my.genome -b 5
chr1 0 105
chr1 795 985

$ bedtools slop -i A.bed -g my.genome -l 2 -r 3
chr1 3 103
chr1 798 983
```

However, if the requested number of bases exceeds the boundaries of the chromosome, `bedtools slop` will "clip" the feature accordingly.

```
$ cat A.bed
chr1  5   100
chr1  800 980

$ cat my.genome
chr1  1000

$ bedtools slop -i A.bed -g my.genome -b 5000
chr1  0   1000
chr1  0   1000
```

`-s` Resizing features according to strand

bedtools slop will optionally increase the size of a feature based on strand.

For example:

```
$ cat A.bed
chr1 100 200 a1 1 +
chr1 100 200 a2 2 -

$ cat my.genome
chr1 1000

$ bedtools slop  -i A.bed -g my.genome -l 50 -r 80 -s
chr1 50  280 a1 1 +
chr1 20  250 a2 2 -
```

`-pct` Resizing features by a given fraction

bedtools slop will optionally increase the size of a feature by a user-specific fraction.

For example:

```
$ cat A.bed
chr1 100 200 a1 1 +

$ bedtools slop -i A.bed -g my.genome -b 0.5 -pct
chr1 50  250 a1 1 +

$ bedtools slop -i a.bed -l 0.5 -r 0.0 -pct -g my.genome
chr1  50      200     a1      1       +
```

`-header` Print the header for the A file before reporting results.

By default, if your A file has a header, it is ignored when reporting results. This option will instead tell bedtools to first print the header for the A file prior to reporting results.

sort

sortBed sorts a feature file by chromosome and other criteria.

Usage and option summary

Usage:

```
sortBed [OPTIONS] -i <BED/GFF/VCF>
```

Option	Description
-sizeA	Sort by feature size in ascending order.
-sizeD	Sort by feature size in descending order.
-chrThenSizeA	Sort by chromosome, then by feature size (asc).
-chrThenSizeD	Sort by chromosome, then by feature size (desc).
-chrThenScoreA	Sort by chromosome, then by score (asc).
-chrThenScoreD	Sort by chromosome, then by score (desc).

Default behavior

By default, **sortBed** sorts a BED file by chromosome and then by start position in ascending order.

For example:

```
cat A.bed
chr1 800 1000
chr1 80  180
chr1 1   10
chr1 750 10000

sortBed -i A.bed
chr1 1   10
chr1 80  180
chr1 750 10000
chr1 800 1000
```

Optional sorting behavior

sortBed will also sorts a BED file by chromosome and then by other criteria.

For example, to sort by chromosome and then by feature size (in descending order):

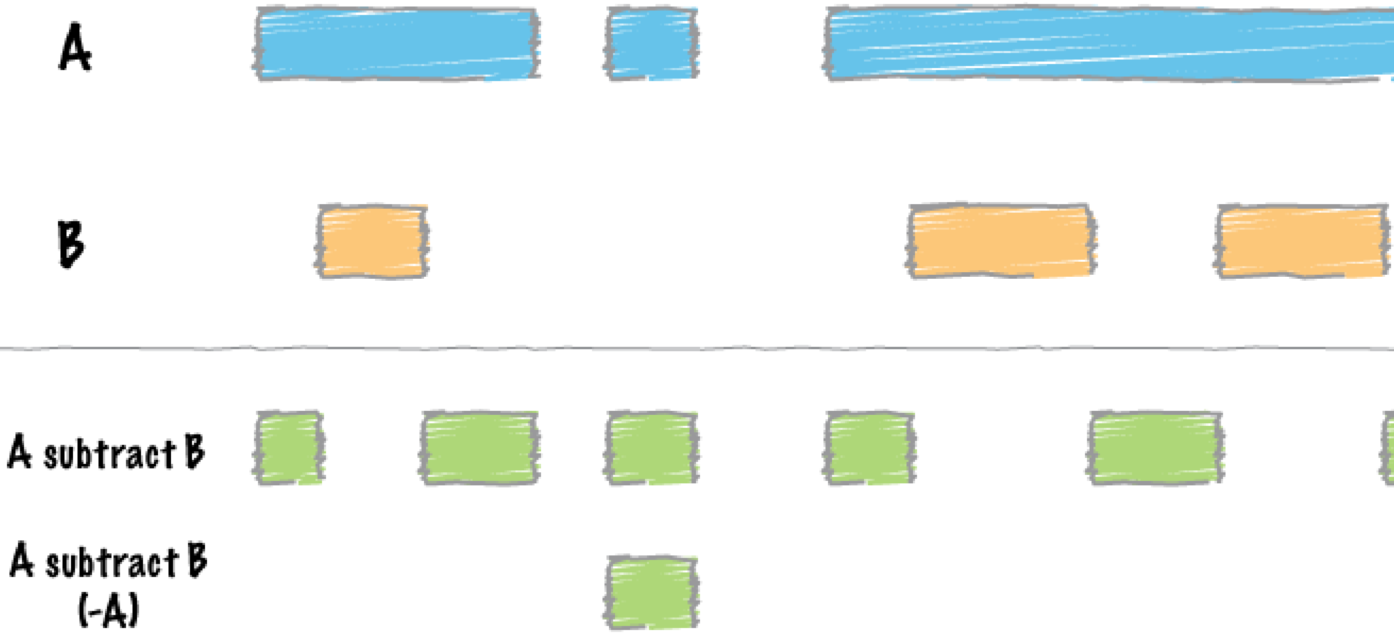
```
cat A.bed
chr1 800 1000
chr1 80  180
chr1 1   10
chr1 750 10000

sortBed -i A.bed -sizeD
chr1 750 10000
chr1 800 1000
chr1 80  180
chr1 1   10
```

Disclaimer: it should be noted that **sortBed** is merely a convenience utility, as the UNIX sort utility will sort BED files more quickly while using less memory. For example, UNIX sort will sort a BED file by chromosome then by start position in the following manner:

```
sort -k 1,1 -k2,2 -n a.bed
chr1 1   10
```

```
chr1 80  180
chr1 750 10000
chr1 800 1000
```

subtract



`bedtools subtract` searches for features in B that overlap A. If an overlapping feature is found in B, 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, the A feature will not be reported.

Usage and option summary

Usage:

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

(or):

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

Option	Description
-f	Minimum overlap required as a fraction of A. Default is 1E-9 (i.e. 1bp).
-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.

Default behavior

By default, `bedtools subtracts` removes each overlapping interval in B from A. If a feature in B *completely* overlaps a feature in A, the A feature is removed.

```
$ cat A.bed
chr1  10   20
chr1  100  200

$ cat B.bed
chr1  0    30
chr1  180  300

$ bedtools subtract -a A.bed -b B.bed
chr1  100  180
```

`-f` Requiring a minimal overlap fraction before subtracting

This option behaves the same as the `-f` option for `bedtools intersect`. In this case, `subtract` will only subtract an overlap with B if it covers at least the fraction of A defined by `-f`. If an overlap is found, but it does not meet the overlap fraction, the original A feature is reported without subtraction.

```
$ cat A.bed
chr1  100  200

$ cat B.bed
chr1  180  300

$ bedtools subtract -a A.bed -b B.bed -f 0.10
chr1  100  180

$ bedtools subtract -a A.bed -b B.bed -f 0.80
chr1  100  200
```

`-s` Enforcing same "strandedness"

This option behaves the same as the `-s` option for `bedtools intersect` while scanning for features in B that should be subtracted from A.

```
$ cat A.bed
chr1  100  200    a1  1   +

$ cat B.bed
chr1  80   120    b1  1   +
chr1  180  300    b2  1   -

$ bedtools subtract -a A.bed -b B.bed -s
chr1  120  120    a1  1   +
```

`-S` Enforcing opposite "strandedness"

This option behaves the same as the `-s` option for `bedtools intersect` while scanning for features in B that should be subtracted from A.

```
$ cat A.bed
chr1  100  200    a1  1   +

$ cat B.bed
chr1  80   120    b1  1   +
chr1  180  300    b2  1   -

$ bedtools subtract -a A.bed -b B.bed -S
chr1  100  180    a1  1   +
```

`-A` Remove features with any overlap

Unlike the default behavior, the `-A` option will completely remove a feature from A if it has even 1bp of overlap with a feature in B.

```
$ cat A.bed
chr1  100  200

$ cat B.bed
chr1  180  300

$ bedtools subtract -a A.bed -b B.bed
chr1  100  180

$ bedtools subtract -a A.bed -b B.bed -A
```

tag

unionbedg

unionBedGraphs combines multiple BEDGRAPH files into a single file such that one can directly compare coverage (and other text-values such as genotypes) across multiple sample

Usage and option summary

Usage:

```
unionBedGraphs [OPTIONS] -i FILE1 FILE2 FILE3 ... FILEn
```

Option	Description
-header	Print a header line, consisting of chrom, start, end followed by the names of each input BEDGRAPH file.
-names	A list of names (one per file) to describe each file in -i. These names will be printed in the header line.
-empty	Report empty regions (i.e., start/end intervals w/o values in all files). *Requires the '-g FILE' parameter (see below).*
-g	The genome file to be used to calculate empty regions.
-filler TEXT	Use TEXT when representing intervals having no value. Default is '0', but you can use 'N/A' or any other text.
-examples	Show detailed usage examples.

Default behavior

Figure:

```
cat 1.bg
chr1 1000 1500 10
chr1 2000 2100 20

cat 2.bg
chr1 900 1600 60
chr1 1700 2050 50

cat 3.bg
chr1 1980 2070 80
chr1 2090 2100 20

cat sizes.txt
chr1 5000

unionBedGraphs -i 1.bg 2.bg 3.bg
chr1 900  1000 0  60 0
chr1 1000 1500 10 60 0
chr1 1500 1600 0  60 0
chr1 1700 1980 0  50 0
chr1 1980 2000 0  50 80
chr1 2000 2050 20 50 80
chr1 2050 2070 20 0  80
chr1 2070 2090 20 0  0
chr1 2090 2100 20 0  20
```

`-header` Add a header line to the output

Figure:

```
unionBedGraphs -i 1.bg 2.bg 3.bg -header
chrom  start  end  1  2  3
chr1   900    1000 0  60 0
chr1   1000   1500 10 60 0
chr1   1500   1600 0  60 0
chr1   1700   1980 0  50 0
chr1   1980   2000 0  50 80
chr1   2000   2050 20 50 80
chr1   2050   2070 20 0  80
```

```
chr1   2070   2090 20 0  0
chr1   2090   2100 20 0  20
```

`-names` Add a header line with custom file names to the output

Figure:

```
unionBedGraphs -i 1.bg 2.bg 3.bg -header -names WT-1 WT-2 KO-1
chrom  start  end   WT-1  WT-2  KO-1
chr1   900    1000  0     60    0
chr1   1000   1500  10    60    0
chr1   1500   1600  0     60    0
chr1   1700   1980  0     50    0
chr1   1980   2000  0     50    80
chr1   2000   2050  20    50    80
chr1   2050   2070  20    0     80
chr1   2070   2090  20    0     0
chr1   2090   2100  20    0     20
```

`-empty` Include regions that have zero coverage in all BEDGRAPH files.

Figure:

```
unionBedGraphs -i 1.bg 2.bg 3.bg -empty -g sizes.txt -header
chrom  start  end  WT-1  WT-2  KO-1
chrom  start  end  1     2     3
chr1   0      900  0     0     0
chr1   900    1000 0     60    0
chr1   1000   1500 10    60    0
chr1   1500   1600 0     60    0
chr1   1600   1700 0     0     0
chr1   1700   1980 0     50    0
chr1   1980   2000 0     50    80
chr1   2000   2050 20    50    80
chr1   2050   2070 20    0     80
chr1   2070   2090 20    0     0
chr1   2090   2100 20    0     20
chr1   2100   5000 0     0     0
```

`-filler` Use a custom value for missing values.

Figure:

```
unionBedGraphs -i 1.bg 2.bg 3.bg -empty -g sizes.txt -header -filler N/A
chrom start end  WT-1  WT-2  KO-1
chrom start end  1     2     3
chr1  0     900  N/A   N/A   N/A
chr1  900   1000 N/A   60    N/A
chr1  1000  1500 10    60    N/A
chr1  1500  1600 N/A   60    N/A
chr1  1600  1700 N/A   N/A   N/A
chr1  1700  1980 N/A   50    N/A
chr1  1980  2000 N/A   50    80
chr1  2000  2050 20    50    80
```

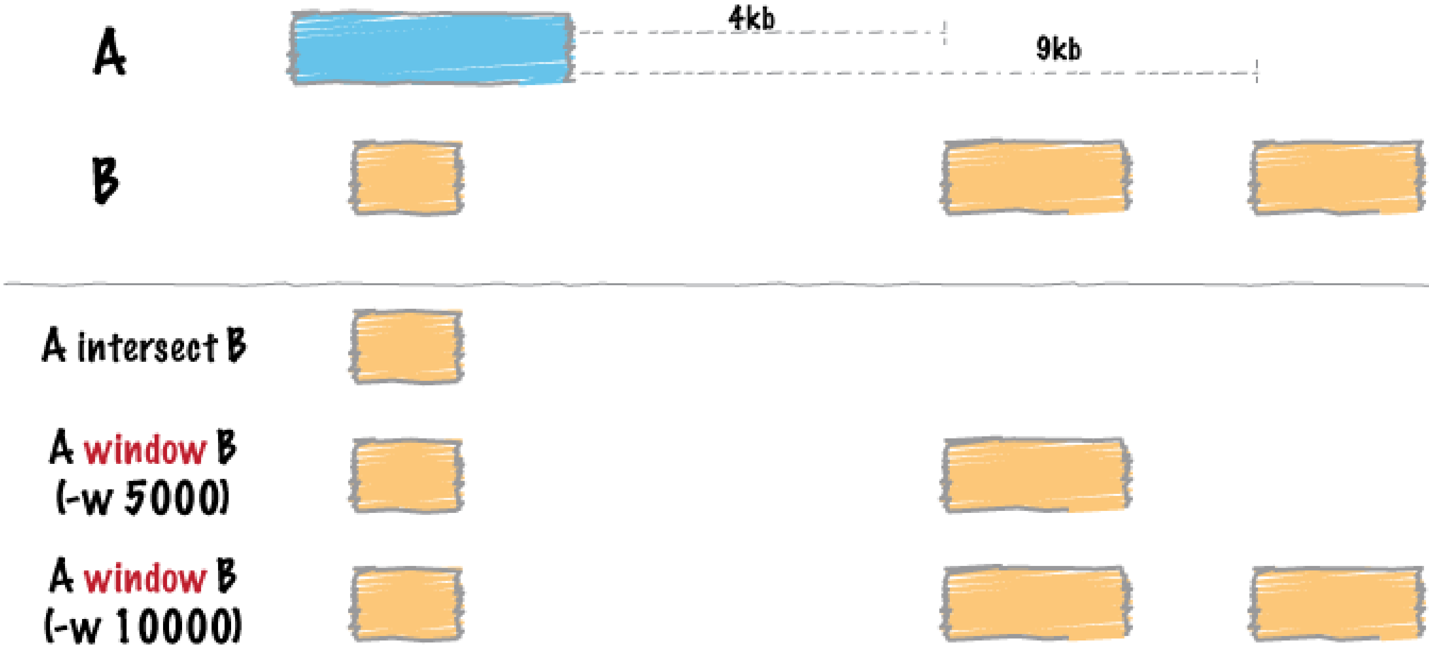
```
chr1  2050  2070 20    N/A   80
chr1  2070  2090 20    N/A   N/A
chr1  2090  2100 20    N/A   20
chr1  2100  5000 N/A   N/A   N/A
```

Use BEDGRAPH files with non-numeric values.

Figure:

```
cat 1.snp.bg
chr1 0 1 A/G
chr1 5 6 C/T

cat 2.snp.bg
chr1 0 1 C/C
chr1 7 8 T/T

cat 3.snp.bg
chr1 0 1 A/G
chr1 5 6 C/T

unionBedGraphs -i 1.snp.bg 2.snp.bg 3.snp.bg -filler -/-
chr1 0 1 A/G C/C A/G
chr1 5 6 C/T -/- C/T
chr1 7 8 -/- T/T -/-
```

window



Similar to `bedtools intersect`, `window` searches for overlapping features in A and B. However, `window` adds a specified number (1000, by default) of base pairs upstream and downstream of each feature in A. In effect, this allows features in B that are "near" features in A to be detected.

Usage and option summary

Usage:

```
bedtools window [OPTIONS] [-a|-abam] -b <BED/GFF/VCF>
```

(or):

```
bedtools window [OPTIONS] [-a|-abam] -b <BED/GFF/VCF>
```

Option	Description
-abam	BAM file A. Each BAM alignment in A is compared to B in search of overlaps. Use “stdin” if passing A with a UNIX pipe: For example: samtools view -b <BAM> | bedtools window -abam stdin -b genes.bed
-ubam	Write uncompressed BAM output. The default is write compressed BAM output.
-bed	When using BAM input (-abam), write output as BED. The default is to write output in BAM when using -abam. For example: bedtools window -abam reads.bam -b genes.bed -bed
-w	Base pairs added upstream and downstream of each entry in A when searching for overlaps in B. *Default is 1000 bp*.
-l	Base pairs added upstream (left of) of each entry in A when searching for overlaps in B. *Allows one to create assymetrical “windows”. Default is 1000bp*.
-r	Base pairs added downstream (right of) of each entry in A when searching for overlaps in B. *Allows one to create assymetrical “windows”. Default is 1000bp*.
-sw	Define -l and -r based on strand. For example if used, -l 500 for a negative-stranded feature will add 500 bp downstream. *By default, this is disabled*.
-sm	Only report hits in B that overlap A on the same strand. *By default, overlaps are reported without respect to strand*.
-Sm	Only report hits in B that overlap A on the same strand. *By default, overlaps are reported without respect to strand*.
-u	Write original A entry once if any overlaps found in B. In other words, just report the fact at least one overlap was found in B.
-c	For each entry in A, report the number of hits in B while restricting to -f. Reports 0 for A entries that have no overlap with B.
-v	Only report those entries in A that have *no overlaps* with B.
-header	Print the header from the A file prior to results.

Default behavior

By default, `bedtools window` adds 1000 bp upstream and downstream of each A feature and searches for features in B that overlap this “window”. If an overlap is found in B, both the *original* A feature and the *original* B feature are reported.

```
$ cat A.bed
chr1  100  200

$ cat B.bed
chr1  500  1000
chr1  1300 2000

$ bedtools window -a A.bed -b B.bed
chr1  100  200  chr1  500  1000
```

`-w` Defining a custom window size

Instead of using the default window size of 1000bp, one can define a custom, *symmetric* window around each feature in A using the **-w** option. One should specify the window size in base pairs. For example, a window of 5kb should be defined as `-w 5000`.

For example (note that in contrast to the default behavior, the second B entry is reported):

```
$ cat A.bed
chr1  100  200

$ cat B.bed
chr1  500  1000
chr1  1300 2000

$ bedtools window -a A.bed -b B.bed -w 5000
chr1  100  200  chr1  500   1000
chr1  100  200  chr1  1300  2000
```

`-l and -r` Defining *assymteric* windows

One can also define asymmetric windows where a differing number of bases are added upstream and downstream of each feature using the `-l` (upstream) and `-r` (downstream)** options.

Note: By default, the `-l` and `-r` options ignore. If you want to define *upstream* and *downstream* based on strand, use the `-sw` option (below) with the `-l` and `-r` options.

For example (note the difference between -l 200 and -l 300):

```
$ cat A.bed
chr1  1000  2000

$ cat B.bed
chr1  500   800
chr1  10000 20000

$ bedtools window -a A.bed -b B.bed -l 200 -r 20000
chr1  100   200  chr1  10000  20000

$ bedtools window -a A.bed -b B.bed -l 300 -r 20000
chr1  100   200  chr1  500    800
chr1  100   200  chr1  10000  20000
```

`-sw` Defining assymteric windows based on strand

Especially when dealing with gene annotations or RNA-seq experiments, you may want to define asymmetric windows based on "strand". For example, you may want to screen for overlaps that occur within 5000 bp upstream of a gene (e.g. a promoter region) while screening only 1000 bp downstream of the gene. By enabling the `-sw` ("stranded" windows) option, the windows are added upstream or downstream according to strand. For example, imagine one specifies `-l 5000`, `-r 1000` as well as the `-sw` option. In this case, forward stranded ("+") features will screen 5000 bp to the *left* (that is, *lower* genomic coordinates) and 1000 bp to the *right* (that is, *higher* genomic coordinates). By contrast, reverse stranded ("-") features will screen 5000 bp to the *right* (that is, *higher* genomic coordinates) and 1000 bp to the *left* (that is, *lower* genomic coordinates).

For example (note the difference between `-l 200` and `-l 300`):

```
$ cat A.bed
chr1  10000  20000  A.forward  1  +
chr1  10000  20000  A.reverse  1  -

$ cat B.bed
chr1  1000   8000   B1
```

```
chr1  24000  32000  B2

$ bedtools window -a A.bed -b B.bed -l 5000 -r 1000 -sw
chr1  10000  20000  A.forward  1  +  chr1  1000   8000   B1
chr1  10000  20000  A.reverse  1  -  chr1  24000  32000  B2
```

`-sm` Enforcing matches with the *same* "strandedness"

This option behaves the same as the `-s` option for `bedtools intersect` while scanning for overlaps within the "window" surrounding A. That is, overlaps in B will only be included if the B interval is on the *same* strand as the A interval.

`-Sm` Enforcing matches with the *same* "strandedness"

This option behaves the same as the `-S` option for `bedtools intersect` while scanning for overlaps within the "window" surrounding A. That is, overlaps in B will only be included if the B interval is on the *opposite* strand as the A interval.

`-u` Reporting the presence/absence of at least one overlapping feature

This option behaves the same as for `bedtools intersect`. That is, even if multiple overlaps exist, each A interval will only be reported once.

`-c` Reporting the number of overlapping features

This option behaves the same as for `bedtools intersect`. That is, it will report the *count* of intervals in B that overlap each A interval.

`-v` Reporting the absence of any overlapping features

This option behaves the same as for `bedtools intersect`. That is, it will only report those intervals in A that have have *zero* overlaps in B.

`-header` Print the header for the A file before reporting results.

By default, if your A file has a header, it is ignored when reporting results. This option will instead tell bedtools to first print the header for the A file prior to reporting results.

1.6 Example usage

Below are several examples of basic bedtools usage. Example BED files are provided in the /data directory of the bedtools distribution.

1.6.1 bedtools intersect

Report the base-pair overlap between sequence alignments and genes.

```
bedtools intersect -a reads.bed -b genes.bed
```

Report whether each alignment overlaps one or more genes. If not, the alignment is not reported.

```
bedtools intersect -a reads.bed -b genes.bed -u
```

Report those alignments that overlap NO genes. Like "grep -v"

```
bedtools intersect -a reads.bed -b genes.bed -v
```

Report the number of genes that each alignment overlaps.

```
bedtools intersect -a reads.bed -b genes.bed -c
```

Report the entire, original alignment entry for each overlap with a gene.

```
bedtools intersect -a reads.bed -b genes.bed -wa
```

Report the entire, original gene entry for each overlap with a gene.

```
bedtools intersect -a reads.bed -b genes.bed -wb
```

Report the entire, original alignment and gene entries for each overlap.

```
bedtools intersect -a reads.bed -b genes.bed -wa -wb
```

Only report an overlap with a repeat if it spans at least 50% of the exon.

```
bedtools intersect -a exons.bed -b repeatMasker.bed -f 0.50
```

Only report an overlap if comprises 50% of the structural variant and 50% of the segmental duplication. Thus, it is reciprocally at least a 50% overlap.

```
bedtools intersect -a SV.bed -b segmentalDups.bed -f 0.50 -r
```

Read BED A from stdin. For example, find genes that overlap LINEs but not SINEs.

```
bedtools intersect -a genes.bed -b LINES.bed | intersectBed -a stdin -b SINEs.bed -v
```

Retain only single-end BAM alignments that overlap exons.

```
bedtools intersect -abam reads.bam -b exons.bed > reads.touchingExons.bam
```

Retain only single-end BAM alignments that do not overlap simple sequence repeats.

```
bedtools intersect -abam reads.bam -b SSRs.bed -v > reads.noSSRs.bam
```

1.6.2 bedtools bamtobed

Convert BAM alignments to BED format.

```
bedtools bamtobed -i reads.bam > reads.bed
```

Convert BAM alignments to BED format using the BAM edit distance (NM) as the BED "score".

```
bedtools bamtobed -i reads.bam -ed > reads.bed
```

Convert BAM alignments to BEDPE format.

```
bedtools bamtobed -i reads.bam -bedpe > reads.bedpe
```

1.6.3 bedtools window

Report all genes that are within 10000 bp upstream or downstream of CNVs.

```
bedtools window -a CNVs.bed -b genes.bed -w 10000
```

Report all genes that are within 10000 bp upstream or 5000 bp downstream of CNVs.

```
bedtools window -a CNVs.bed -b genes.bed -l 10000 -r 5000
```

Report all SNPs that are within 5000 bp upstream or 1000 bp downstream of genes. Define upstream and downstream based on strand.

```
bedtools window -a genes.bed -b snps.bed -l 5000 -r 1000 -sw
```

1.6.4 bedtools closest

Note: By default, if there is a tie for closest, all ties will be reported. **closestBed** allows overlapping features to be the closest.

Find the closest ALU to each gene.

```
bedtools closest -a genes.bed -b ALUs.bed
```

Find the closest ALU to each gene, choosing the first ALU in the file if there is a tie.

```
bedtools closest -a genes.bed -b ALUs.bed -t first
```

Find the closest ALU to each gene, choosing the last ALU in the file if there is a tie.

```
bedtools closest -a genes.bed -b ALUs.bed -t last
```

1.6.5 bedtools subtract

Note: If a feature in A is entirely "spanned" by any feature in B, it will not be reported.

Remove introns from gene features. Exons will (should) be reported.

```
bedtools subtract -a genes.bed -b introns.bed
```

1.6.6 bedtools merge

Note: `merge` requires that the input is sorted by chromosome and then by start coordinate. For example, for BED files, one would first sort the input as follows: `sort -k1,1 -k2,2n input.bed > input.sorted.bed`

Merge overlapping repetitive elements into a single entry.

```
bedtools merge -i repeatMasker.bed
```

Merge overlapping repetitive elements into a single entry, returning the number of entries merged.

```
bedtools merge -i repeatMasker.bed -n
```

Merge nearby (within 1000 bp) repetitive elements into a single entry.

```
bedtools merge -i repeatMasker.bed -d 1000
```

1.6.7 bedtools coverage

Compute the coverage of aligned sequences on 10 kilobase "windows" spanning the genome.

```
bedtools coverage -a reads.bed -b windows10kb.bed | head
chr1 0     10000 0  10000 0.00
chr1 10001 20000 33 10000 0.21
chr1 20001 30000 42 10000 0.29
chr1 30001 40000 71 10000 0.36
```

Compute the coverage of aligned sequences on 10 kilobase "windows" spanning the genome and created a BEDGRAPH of the number of aligned reads in each window for display on the UCSC browser.

```
bedtools coverage -a reads.bed -b windows10kb.bed | cut -f 1-4 > windows10kb.cov.bedg
```

Compute the coverage of aligned sequences on 10 kilobase "windows" spanning the genome and created a BEDGRAPH of the fraction of each window covered by at least one aligned read for display on the UCSC browser.

```
bedtools coverage -a reads.bed -b windows10kb.bed | \
   awk '{OFS="\t"; print $1,$2,$3,$6}' \
   > windows10kb.pctcov.bedg
```

1.6.8 bedtools complement

Report all intervals in the human genome that are not covered by repetitive elements.

```
bedtools complement -i repeatMasker.bed -g hg18.genome
```

1.6.9 bedtools shuffle

Randomly place all discovered variants in the genome. However, prevent them from being placed in know genome gaps.

```
bedtools shuffle -i variants.bed -g hg18.genome -excl genome_gaps.bed
```

Randomly place all discovered variants in the genome. However, prevent them from being placed in know genome gaps and require that the variants be randomly placed on the same chromosome.

```
bedtools shuffle -i variants.bed -g hg18.genome -excl genome_gaps.bed -chrom
```

1.7 Advanced usage

1.7.1 Mask all regions in a genome except for targeted capture regions.

Step 1. Add 500 bp up and downstream of each probe

```
bedtools slop -i probes.bed -b 500 > probes.500bp.bed
```

Step 2. Get a BED file of all regions not covered by the probes (+500 bp up/down)

```
bedtools complement -i probes.500bp.bed -g hg18.genome > probes.500bp.complement.bed
```

Step 3. Create a masked genome where all bases are masked except for the probes +500bp

```
bedtools maskfasta -in hg18.fa -bed probes.500bp.complement.bed -fo \
> hg18.probecomplement.masked.fa
```

1.7.2 Screening for novel SNPs.

Find all SNPs that are not in dbSnp and not in the latest 1000 genomes calls

```
bedtools intersect -a snp.calls.bed -b dbSnp.bed -v | \
bedtools intersect -a - -b 1KG.bed -v | \
> snp.calls.novel.bed
```

1.7.3 Computing the coverage of features that align entirely within an interval.

By default, bedtools `coverage` counts any feature in A that overlaps B by >= 1 bp. If you want to require that a feature align entirely within B for it to be counted, you can first use intersectBed with the “-f 1.0” option.

```
bedtools intersect -a features.bed -b windows.bed -f 1.0 | \
bedtools coverage -a - -b \
> windows.bed.coverage
```

1.7.4 Computing the coverage of BAM alignments on exons.

One can combine `samtools` with `bedtools` to compute coverage directly from the BAM data by using `bamtobed`.

```
bedtools bamtobed -i reads.bam | \
bedtools coverage -a - -b exons.bed \
> exons.bed.coverage
```

Take it a step further and require that coverage be from properly-paired reads.

```
samtools view -uf 0x2 reads.bam | \
coverageBed -abam - -b exons.bed \
> exons.bed.proper.coverage
```

1.7.5 Computing coverage separately for each strand.

Use grep to only look at forward strand features (i.e. those that end in “+”).

```
bedtools bamtobed -i reads.bam | \
grep \+$  | \
bedtools coverage -a - -b genes.bed \
> genes.bed.forward.coverage
```

Use grep to only look at reverse strand features (i.e. those that end in "-").

```
bedtools bamtobed -i reads.bam | \
grep \-$ | \
bedtools coverage -a - -b genes.bed \
> genes.bed.reverse.coverage
```

1.8 Tips and Tricks

1.8.1 The `-sorted` option

1.9 FAQ

1.9.1 Installation issues

Why am I getting all of these zlib errors?

On certain operating systems (especially free Linux distributions) the complete zlib libraries are not installed. Bedtools depends upon zlib in order to decompress gzipped files.

```
- Building main bedtools binary.
obj/gzstream.o: In function gzstreambuf::open(char const*, int):
gzstream.C:(.text+0x2a5): undefined reference to gzopen64'
collect2: ld returned 1 exit status
make: *** [all] Error 1
```

If you see an error such as the above, it suggests you need to install the `zlib` and `zlib1g-dev` libraries. This is typically straightforward using package managers. For example, on Debian/Ubuntu this would be:

```
apt-get install zlib
apt-get install zlib1g-dev
```

and on Fedora/Centos this would be:

```
yum install zlib
yum install zlib1g-dev
```

1.9.2 General questions

How do I know what version of bedtools I am using?

Use the –version option.

```
$ bedtools --version
bedtools v2.17.0
```

How do I bring up the help/usage menu?

To receive a high level list of available tools in bedtools, use ``-h``:

```
$ bedtools -h
bedtools: flexible tools for genome arithmetic and DNA sequence analysis.
usage:    bedtools <subcommand> [options]

The bedtools sub-commands include:

[ Genome arithmetic ]
    intersect     Find overlapping intervals in various ways.
    window        Find overlapping intervals within a window around an interval.
    closest       Find the closest, potentially non-overlapping interval.
    coverage      Compute the coverage over defined intervals.
    map           Apply a function to a column for each overlapping interval.
    genomecov     Compute the coverage over an entire genome.
    merge         Combine overlapping/nearby intervals into a single interval.
    cluster       Cluster (but don't merge) overlapping/nearby intervals.
    complement    Extract intervals _not_ represented by an interval file.
...
```

To display the help for a specific tool (e.g., `bedtools shuffle`), use:

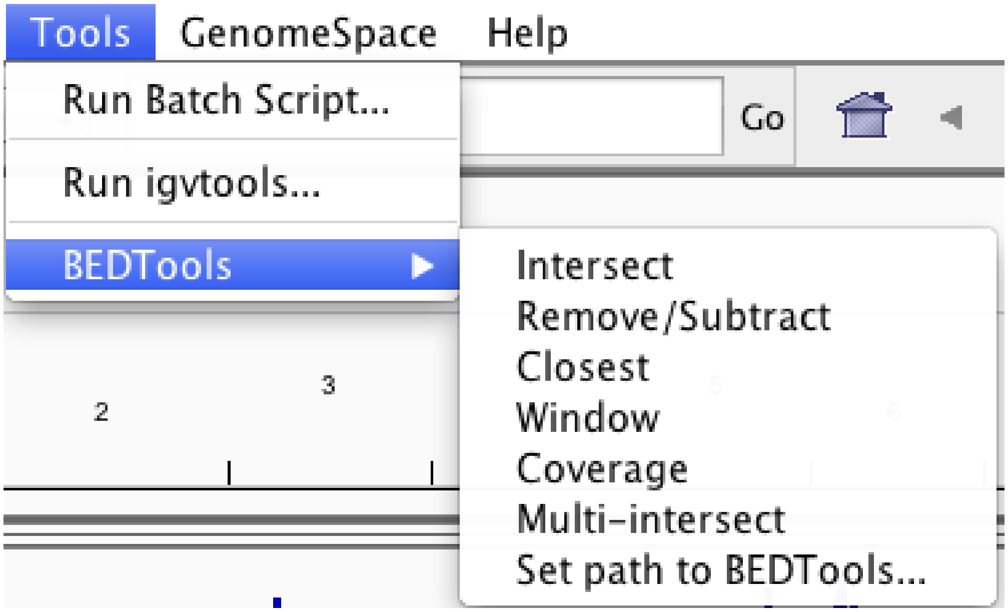
```
$ bedtools merge -h

Tool:    bedtools merge (aka mergeBed)
Version: v2.17.0
Summary: Merges overlapping BED/GFF/VCF entries into a single interval.

Usage:   bedtools merge [OPTIONS] -i <bed/gff/vcf>

Options:
    -s      Force strandedness.  That is, only merge features
            that are the same strand.
            - By default, merging is done without respect to strand.

    -n      Report the number of BED entries that were merged.
            - Note: "1" is reported if no merging occurred.
```

1.10 Related software

Bedtools has been used as an engine behind other genomics software and has been integrated into widely used tools such as Galaxy and IGV. Below is a likely incomplete list. If you know of others, please let us know, or better yet, edit the document on GitHub and send us a pull request. You can do this by clicking on the “Edit and improve this document” link in the lower lefthand corner.

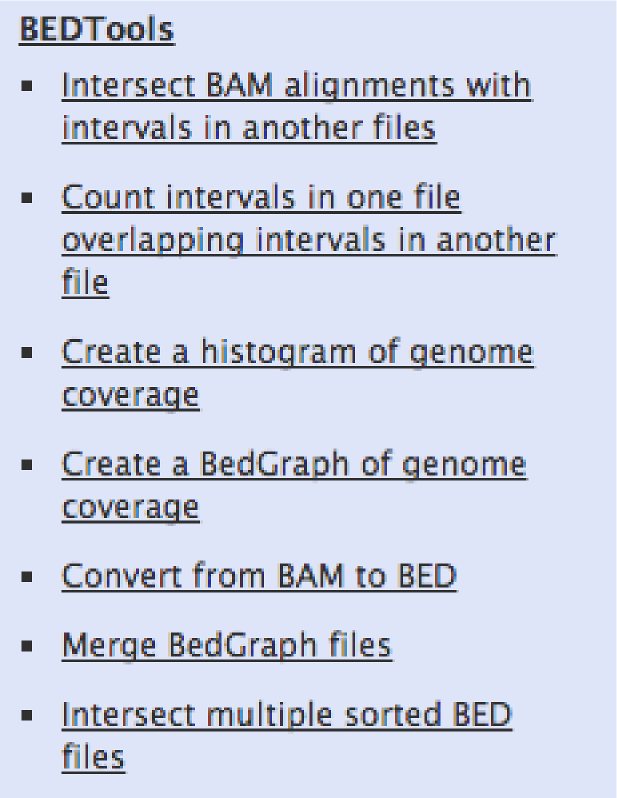
1.10.1 IGV

Bedtools is now integrated into the IGV genome viewer as of IGV version 2.2. We are actively working withe IGV development team to improve and expand this integration. See here and here for details.



1.10.2 Galaxy

Galaxy has its own tools for working with genomic intervals under the "Operate on Genomic Intervals" section. A subset of complementary Bedtools utilities have also been made available on Galaxy in an effort to provide functionality that isn't available with the native Galaxy tools.



BEDTools

- Intersect BAM alignments with intervals in another files
- Count intervals in one file overlapping intervals in another file
- Create a histogram of genome coverage
- Create a BedGraph of genome coverage
- Convert from BAM to BED
- Merge BedGraph files
- Intersect multiple sorted BED files

1.10.3 Pybedtools

Pybedtools is a really fantastic Python library that wraps (and extends upon) the bedtools utilities and exposes them for easy use and new tool development using Python. Pybedtools is actively maintained by Ryan Dale.

1.10.4 MISO

MISO is "a probabilistic framework that quantitates the expression level of alternatively spliced genes from RNA-Seq data, and identifies differentially regulated isoforms or exons across samples." A subset of the functionality in MISO depends upon `bedtools`. MISO is developed by Yarden Katz.

1.10.5 RetroSeq

RetroSeq is "a tool for discovery and genotyping of transposable element variants (TEVs) (also known as mobile element insertions) from next-gen sequencing reads aligned to a reference genome in BAM format". RetroSeq is developed by Thomas Keane. Source code can be obtained on GitHub.

CHAPTER

TWO

BRIEF EXAMPLE

Let's imagine you have a BED file of ChiP-seq peaks from two different experiments. You want to identify peaks that were observed in *both* experiments (requiring 50% reciprocal overlap) and for those peaks, you want to find to find the closest, non-overlapping gene. Such an analysis could be conducted with two, relatively simple bedtools commands.

```
# intersect the peaks from both experiments.
# -f 0.50 combined with -r requires 50% reciprocal overlap between the
# peaks from each experiment.
$ bedtools intersect -a exp1.bed -b exp2.bed -f 0.50 -r > both.bed

# find the closest, non-overlapping gene for each interval where
# both experiments had a peak
# -io ignores overlapping intervals and returns only the closest,
# non-overlapping interval (in this case, genes)
$ bedtools closest -a both.bed -b genes.bed -io > both.nearest.genes.txt
```

CHAPTER
THREE

LICENSE

bedtools is freely available under a GNU Public License (Version 2).

CHAPTER

FOUR

ACKNOWLEDGMENTS

To do.

CHAPTER

FIVE

MAILING LIST

If you have questions, requests, or bugs to report, please email the bedtools mailing list