The BEDTools manual

Aaron R. Quinlan and Ira M. Hall
University of Virginia

Contact: aaronquinlan@gmail.com

1. Overview

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. ***Most recently, we have added support for sequence alignments in BAM (http://samtools.sourceforge.net/) format, as well as for features in GFF and "blocked" BED format.*** The tools are quite fast and typically finish in a matter of a few seconds, even for large datasets.

This manual describes 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 (**Section 6** and **7**).

1.2 Summary of available tools

BEDTools support a wide range of operations for interrogating and manipulating genomic features. We have also defined a concise paired-end BED format (**BEDPE**, described in more detail in **Section 4.1.2**) to facilitate comparisons of discontinuous features (e.g., paired-end sequence reads) to each other (**pairToPair**), as well as to genomic features in traditional BED format (**pairToBed**). This functionality is crucial for interpreting structural variants detected by paired-end mapping, and for identifying fusion genes or alternative splicing patterns by RNA-seq. The table below summarizes the tools available in the suite (tools that support BAM file are indicated).

Utility	Description
intersectBed	Returns overlapping features between two BED/GFF files. *Supports BAM format as input and output.*
pairToBed	Returns overlaps between a BEDPE file and a regular BED/GFF file. *Supports BAM format as input and output.*
pairToPair	Returns overlaps between two BEDPE files.
bamToBed	Converts BAM alignments to BED and BEDPE formats. *Supports BAM format as input.*
windowBed	Returns overlapping features between two BED/GFF files within a "window".
closestBed	Returns the closest feature to each entry in a BED/GFF file.
subtractBed	Removes the portion of an interval that is overlapped by another feature.
mergeBed	Merges overlapping features into a single feature.
coverageBed	Summarizes the depth and breadth of coverage of features in one BED/GFF file (e.g., aligned reads) relative to another (e.g., user-defined windows).
genomeCoverageBed	Histogram or a "per base" report of genome coverage.
fastaFromBed	Creates FASTA sequences from BED/GFF intervals.
maskFastaFromBed	Masks a FASTA file based upon BED/GFF coordinates.
shuffleBed	Permutes the locations of features within a genome.
slopBed	Adjusts features by a requested number of base pairs.
sortBed	Sorts BED/GFF files in useful ways.
linksBed	Creates an HTML links from a BED/GFF file.
complementBed	Returns intervals not spanned by features in a BED/GFF file.

1.3 Fundamental concepts regarding BEDTools usage.

1.3.1 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.

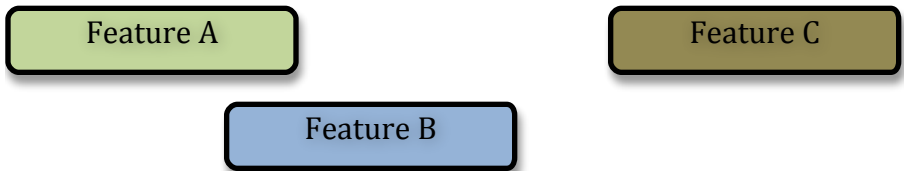
The two most widely used formats for representing genome features are the BED (**B**rowser **E**xtensible **D**ata) and GFF (**G**eneral **F**eature **F**ormat) formats. BEDTools was originally written to work exclusively with genome features described using the BED format, **but it has been recently extended to seamlessly work with both BED and GFF 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)

Section 4 of this manual describes BED and GFF formats in detail and illustrates how to define your own annotations.

1.3.2 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.



1.3.3 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 **intersectBed** in the following manner:

```
$ intersectBed -a snps.bed -b exons.bed
```

1.3.4 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.

1.3.5 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.

1.3.6 File B is loaded into memory.

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.

1.3.7 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.

1.3.8 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". For example:

```
$ cat snps.bed | intersectBed -a stdin -b exons.bed
```

1.3.9 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):

```
$ intersectBed -a snps.bed -b exons.bed
chr1  100100      100101      rs233454
```

```
chr1  200100      200101      rs446788
chr1  300100      300101      rs645678
```

Writing to a file:

```
$ intersectBed -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
```

1.3.10 What is a "genome" file?

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 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.

1.3.11 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. **See Section 4.1.2** for more details.

1.3.12 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.

1.3.13 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.

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

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

1.3.15 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 -
```

1.4 Implementation and algorithmic approach

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.

1.5 License and Availability

BEDTools is freely available under a GNU Public License (Version 2) at:
http://bedtools.googlecode.com

1.6 Discussion group

A discussion group for reporting bugs, asking questions of the developer and of the user community, as well as for requesting new features is available at:
http://groups.google.com/group/bedtools-discuss

2. Installation

BEDTools is intended to run in a "command line" environment on UNIX, LINUX and Apple OS X operating systems. Installing BEDTools involves downloading the latest source code archive followed by compiling the source code into binaries on your local system. The following commands will install BEDTools in a local directory on a *NIX or OS X machine. Note that the "***<version>***" refers to the latest posted version number on http://bedtools.googlecode.com/.

Note: The BEDTools "makefiles" use 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
make clean
make all
ls bin
```

At this point, one should copy the binaries in `BEDTools/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.

3. "Quick start" guide

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 clean
make all
sudo cp bin/* /usr/local/bin/
```

3.2 Use BEDTools

Below are examples of typical BEDTools usage. **Additional usage examples are described in section 6 of this manual.** Using the "-h" option with any BEDTools will report a list of all command line options.

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

```
$ intersectBed -a reads.bed -b genes.bed
```

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

```
$ intersectBed -a reads.bed -b genes.bed –v
```

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

```
$ intersectBed -a genes.bed -b LINES.bed | intersectBed -a stdin -b SINEs.bed –v
```

D. Find the closest ALU to each gene.

```
$ closestBed -a genes.bed -b ALUs.bed
```

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

```
$ mergeBed -i repeatMasker.bed -n
```

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

```
$ mergeBed -i repeatMasker.bed -d 1000
```

4. General usage information

4.1 Supported file formats

4.1.1 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), but the last six columns are not used for any comparisons by the BEDTools. In other words, BEDTools will not perform separate comparisons for each block in a BED12 feature. Instead, it 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:

(A) **BED3**: A BED file where each feature is described by **chrom**, **start,** and **end**.
For example: `chr1  11873  14409`

(B) **BED4**: A BED file where each feature is described by **chrom**, **start**, **end,** and **name**.
For example: `chr1  11873  14409  uc001aaa.3`

(C) **BED5**: A BED file where each feature is described by **chrom**, **start**, **end, name,** and **score**.
For example: `chr1  11873  14409  uc001aaa.3  0`

(D) **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  +`

(E) **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,
```

4.1.2 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
```

4.1.3 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][A-Za-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"
```

4.1.4 Genome files

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.

4.1.5 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). Currently, two BEDTools (**intersectBed** and **pairToBed**) support BAM input and output. These tools allow one to quickly:

(A) Find BAM alignments that overlap (or not) with BED annotation and report them in BED format.
(B) Create a new BAM file of BAM alignments that overlap (or not) with BED annotations. This serves as a powerful way to refine alignment datasets based on biological interest.

The details of how these tools work with BAM files are addressed in **Section 5** of this manual. In additions, **bamToBed** is a convenient utility that will convert BAM files to both BED and BEDPE formats for uses with all other BEDTools.

5. The BEDTools suite

This section covers the functionality and default / optional usage for each of the available BEDTools. Example "figures" are provided in some cases in an effort to convey the purpose of the tool. The behavior of each available parameter is discussed for each tool in abstract terms. More concrete usage examples are provided in **Section 6**.

5.1 intersectBed

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. **intersectBed** 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. **intersectBed** works with both BED and BAM files as input.

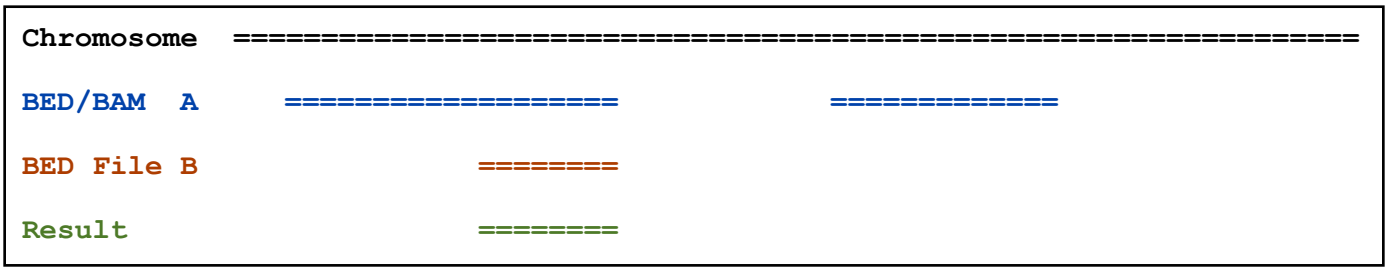
5.1.1 Usage and option summary

Usage: `$ intersectBed [OPTIONS] [-a <BED> || -abam <BAM>] -b <BED>`

Option	Description
-a	BED 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 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> | intersectBed -abam stdin -b genes.bed`
-bed	When using BAM input (-abam), write output as BED. The default is to write output in BAM when using -abam. For example: `intersectBed -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.**
-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.**
-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.**
-v	Only report those entries in A that have **no overlap** in B. **Restricted by -f.**
-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.

5.1.2 Default behavior

By default, if an overlap is found, **intersectBed** reports the shared interval between the two overlapping features.



For example:

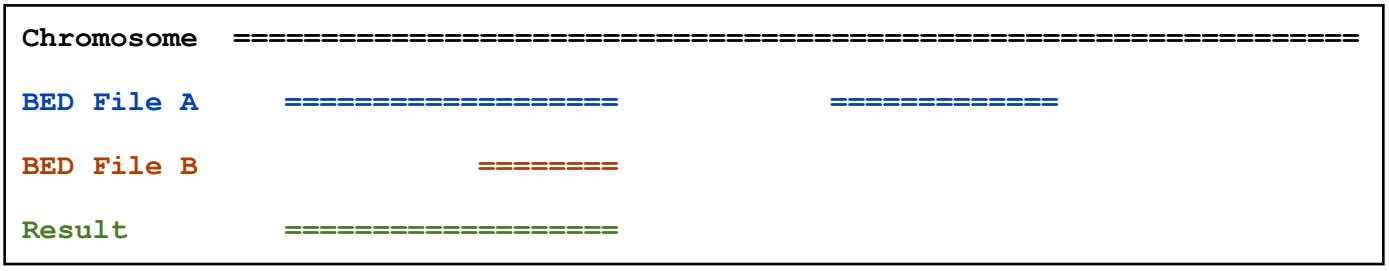
```
$ cat A.bed
chr1  100   200
chr1  1000  2000

$ cat B.bed
chr1  150   250

$ intersectBed -a A.bed -b B.bed
chr1  150   200
```

5.1.3 Reporting the original A feature (-wa)

Instead, one can force **intersectBed** 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 (compare with example from default behavior):

```
$ cat A.bed
chr1  100   200
chr1  1000  2000
```

```
$ cat B.bed
chr1  150   250

$ intersectBed -a A.bed -b B.bed -wa
chr1  100   200
```

5.1.4 Reporting the original B feature (-wb)

Similarly, one can force **intersectBed** 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  100   200
chr1  1000  2000

$ cat B.bed
chr1  150   250

$ intersectBed -a A.bed -b B.bed -wb
chr1  150   200   chr1  150   250
```

Now -wa *and* -wb:

```
$ cat A.bed
chr1  100   200
chr1  1000  2000

$ cat B.bed
chr1  150   250

$ intersectBed -a A.bed -b B.bed -wa -wb
chr1  100   200   chr1  150   250
```

5.1.5 Reporting the presence of *at least one* overlapping feature (-u)

Frequently a feature in "A" will overlap with multiple features in "B". By default, **intersectBed** will report each overlap as a separate output line. However, one may want to simply know that there is at least one overlap (or none). When one uses the –**u** option, "A" features that overlap with one or more "B" features are reported once. Those that overlap with no "B" features are not reported at all.

For example:

```
$ cat A.bed
```

```
chr1  100   200
chr1  1000  2000

$ cat B.bed
chr1  101   201
chr1  120   220

$ intersectBed -a A.bed -b B.bed -u
chr1  100   200
```

5.1.6 Reporting the number of overlapping features (-c)

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.*

For example:

```
$ cat A.bed
chr1  100   200
chr1  1000  2000

$ cat B.bed
chr1  101   201
chr1  120   220

$ intersectBed -a A.bed -b B.bed -c
chr1  100   200   2
chr1  1000  2000  0
```

5.1.7 Reporting the absence of any overlapping features (-v)

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".

For example:

```
$ cat A.bed
chr1  100   200
chr1  1000  2000

$ cat B.bed
chr1  101   201
chr1  120   220

$ intersectBed -a A.bed -b B.bed -v
chr1  1000  2000
```

5.1.8 Requiring a minimal overlap fraction (-f)

By default, **intersectBed** 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

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

5.1.9 Requiring reciprocal minimal overlap fraction (-r, combined with -f)

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

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

5.1.10 Enforcing "strandedness" (-s)

By default, **intersectBed** 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 second B entry is not reported):

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

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

$ intersectBed -a A.bed -b B.bed -wa -wb -s
chr1  100   200   a1    100   +     chr1  130   201   b2    100   +
```

5.1.11 Default behavior when using BAM input (-abam)

When comparing alignments in BAM format (**-abam**) to features in BED format (**-b**), **intersectBed** 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.

For example:

```
$ intersectBed -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 Y P W W _
\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 C A C T G T T A
       gffeaaddddgggggggedgcgeggdeggggggffcgggggggegdfggfgf   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_ggggggggggddgdggVg\gWdfgfgfff    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
```

5.1.12 Output BED format when using BAM input (-bed)

When comparing alignments in BAM format (**-abam**) to features in BED format (**-b**), **intersectBed** 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.

For example:

```
$ intersectBed -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    +
```

5.2 pairToBed

pairToBed compares each end of a BEDPE feature or a paired-end BAM alignment to a BED file in search of overlaps.

5.2.1 Usage and option summary

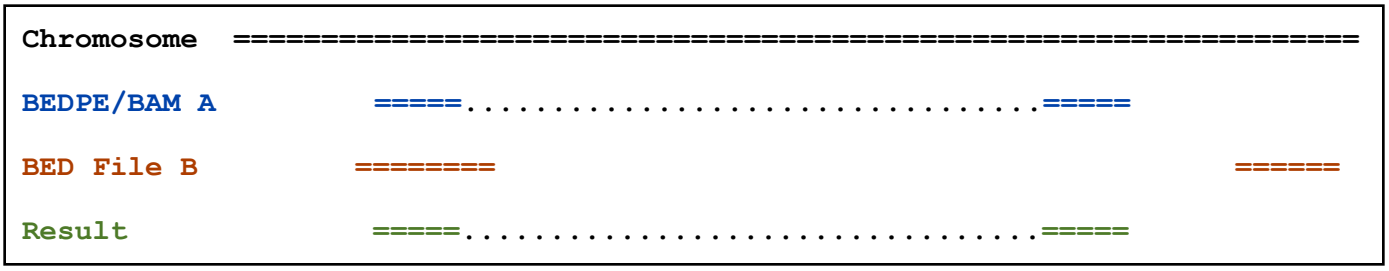
Usage: `$ pairToBed [OPTIONS] [-a <BEDPE> || -abam <BAM>] -b <BED>`

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. Output will be in BEDPE format.
-b	BED file B. Use "stdin" if passing B with a UNIX pipe.
-abam	BAM file A. Each end of 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> | pairToBed -abam stdin -b genes.bed | samtools view -`
-bedpe	When using BAM input (-abam), write output as BEDPE. The default is to write output in BAM when using -abam. For example: `pairToBed -abam reads.bam -b genes.bed -bedpe`
-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.
-type	Approach to reporting overlaps between BEDPE and BED. **either** Report overlaps if either end of A overlaps B. - **Default**. **neither** Report A if neither end of A overlaps B. **xor** Report overlaps if one and only one end of A overlaps B. **both** Report overlaps if both ends of A overlap B. **notboth** Report overlaps if neither end or one and only one end of A overlap B. **ispan** Report overlaps between [end1, start2] of A and B. - *Note: If chrom1 <> chrom2, entry is ignored.* **ospan** Report overlaps between [start1, end2] of A and B. - *Note: If chrom1 <> chrom2, entry is ignored.* **notispan** Report A if ispan of A doesn't overlap B. - *Note: If chrom1 <> chrom2, entry is ignored.* **notospan** Report A if ospan of A doesn't overlap B. - *Note: If chrom1 <> chrom2, entry is ignored.*

5.2.2 Default behavior

By default, a BEDPE / BAM feature will be reported if *either* end overlaps a feature in the BED file. In the example below, the left end of the pair overlaps B yet the right end does not. Thus, BEDPE/ BAM A is reported since the default is to report A if either end overlaps B.

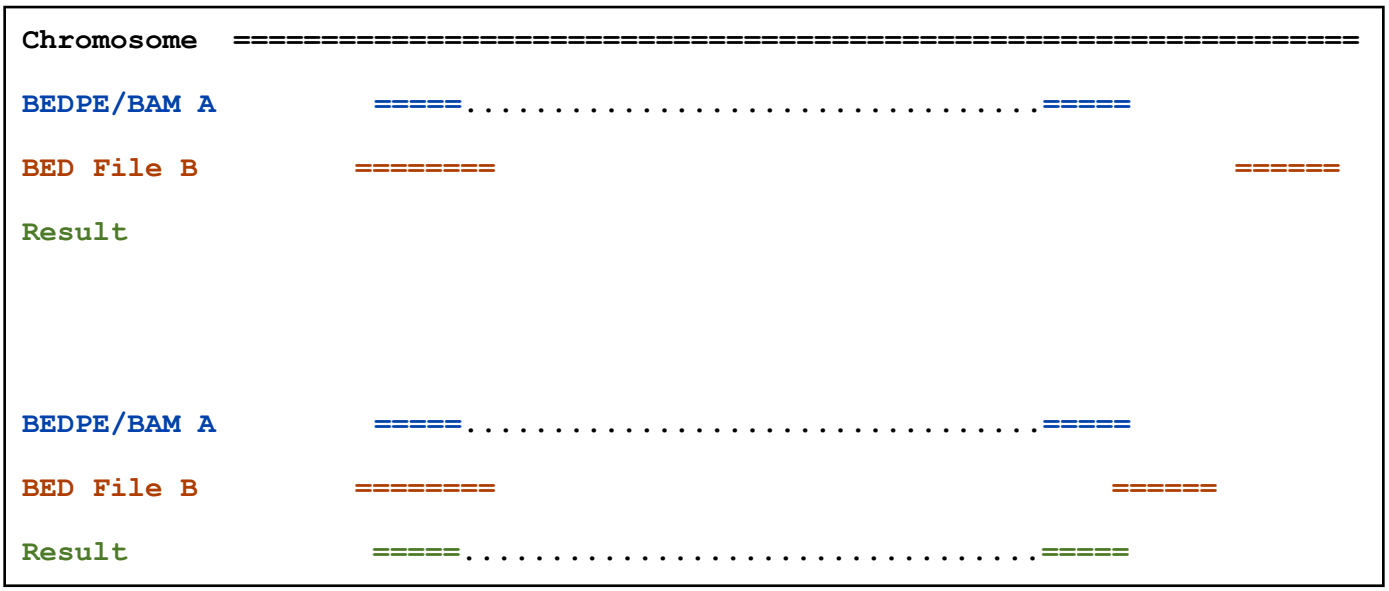
Default: Report A if *either* end overlaps B.



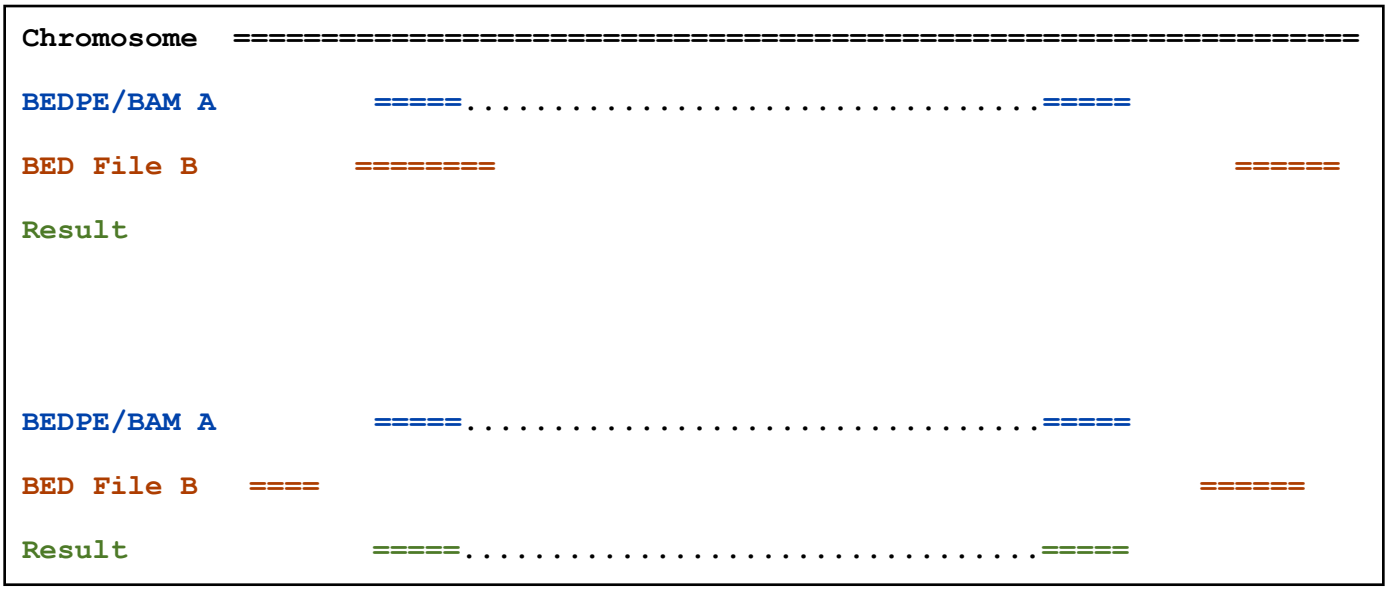
5.2.3 Optional overlap requirements (-type)

Using then **-type** option, **pairToBed** provides several other overlap requirements for controlling how overlaps between BEDPE/BAM A and BED B are reported. The examples below illustrate how each option behaves.

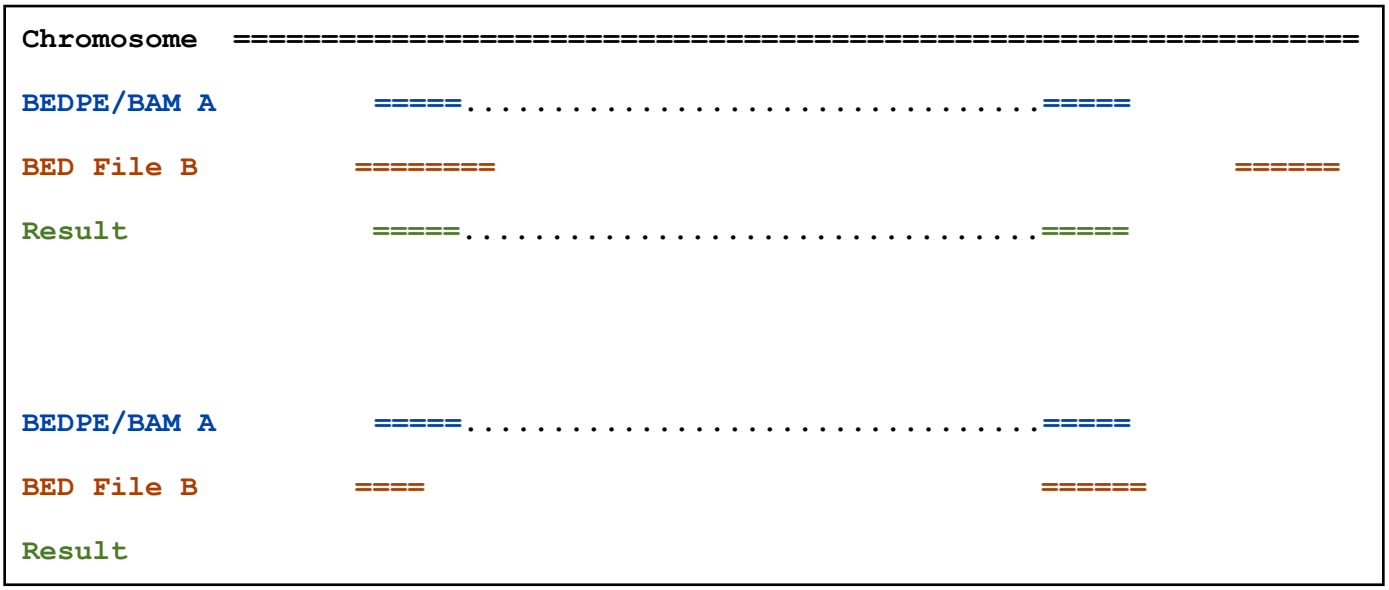
-type both: Report A only if *both* ends overlap B.



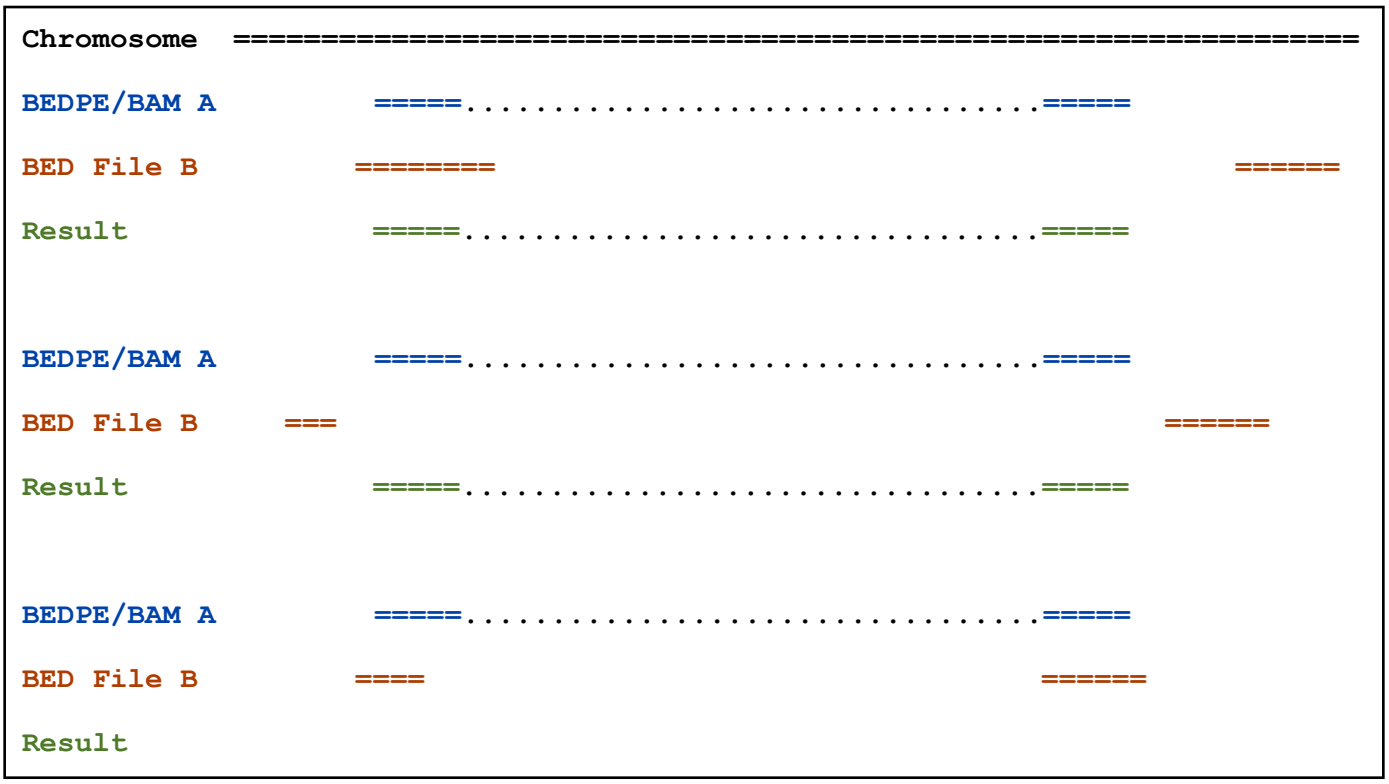
-type neither: Report A only if *neither* end overlaps B.



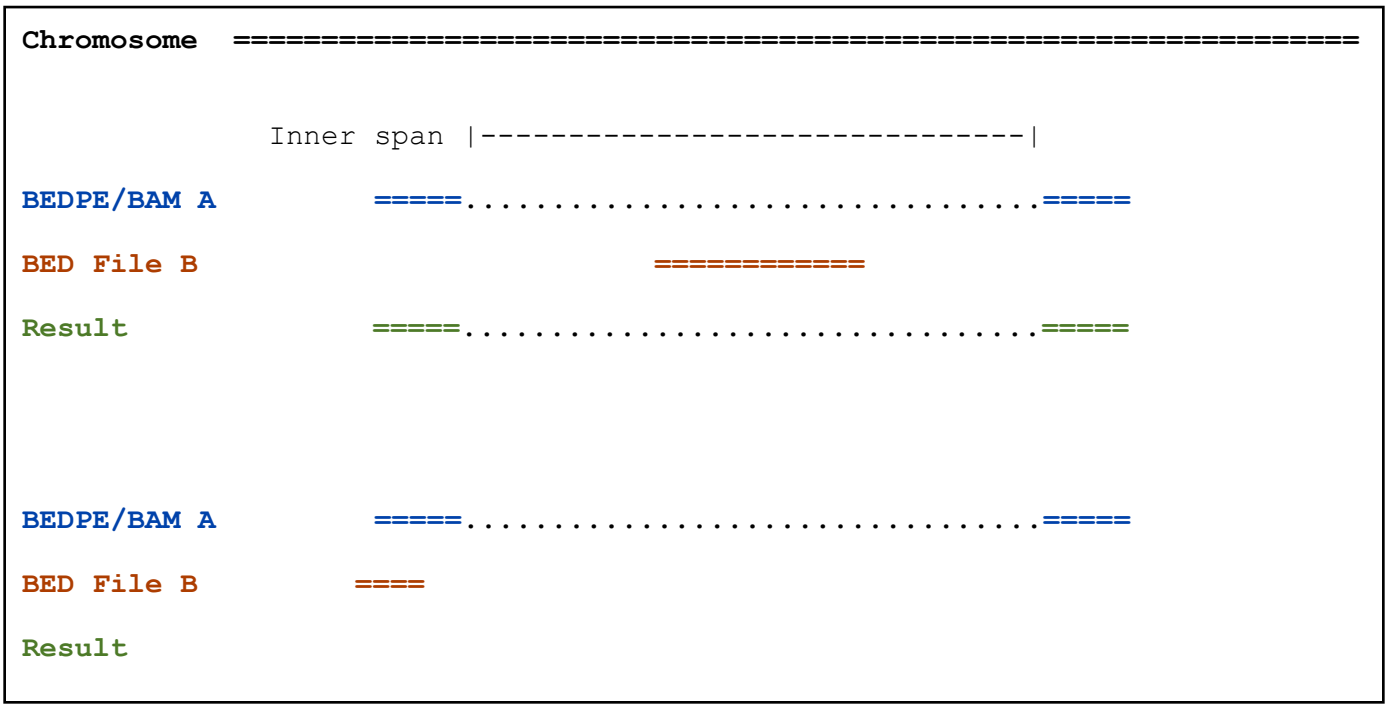
-type xor: Report A only if *one and only one* end overlaps B.



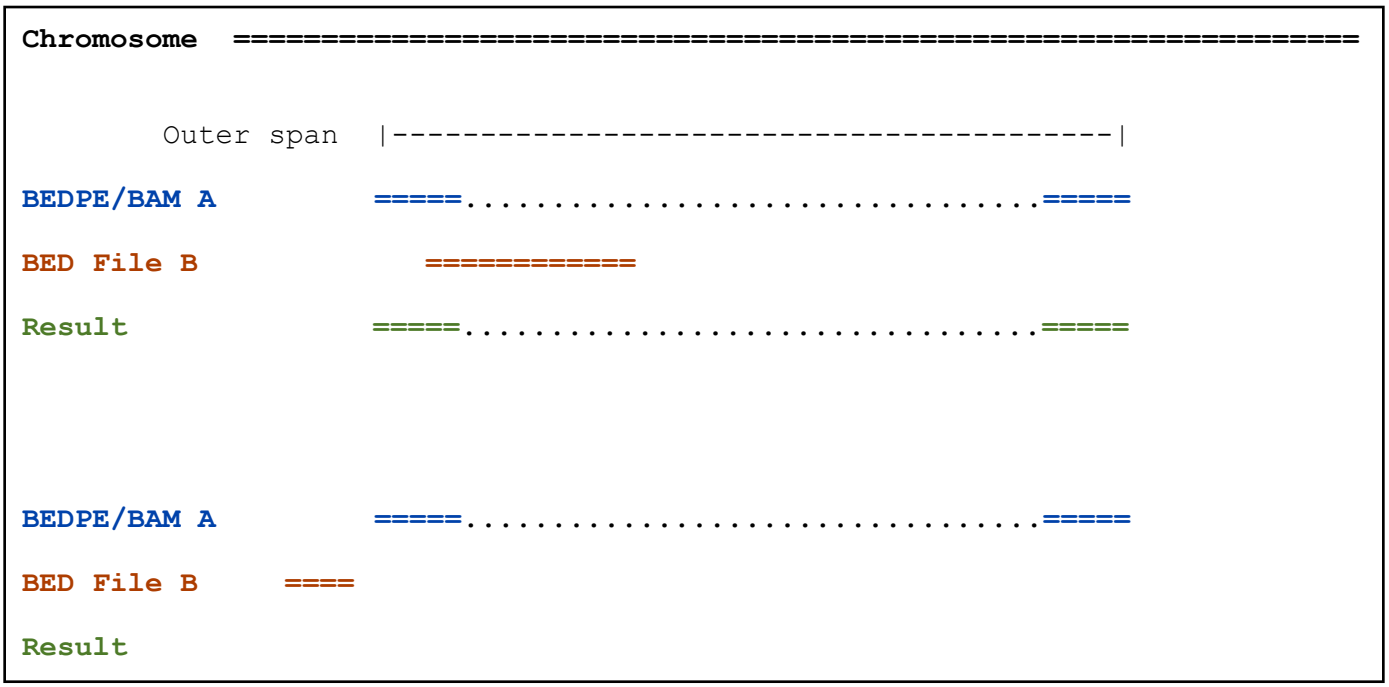
-type notboth: Report A only if *neither end* **or** *one and only one* end overlaps B. Thus "notboth" includes what would be reported by "neither" and by "xor".



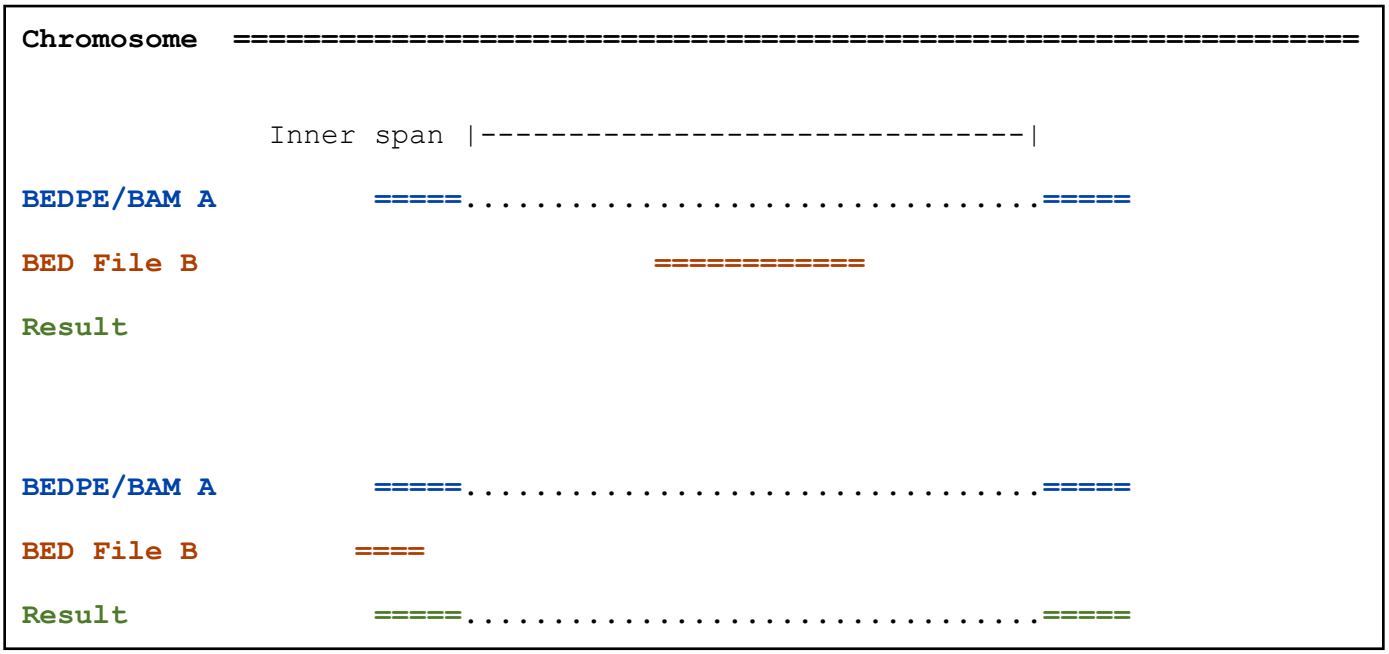
-type ispan: Report A if it's "*inner span*" overlaps B. Applicable only to intra-chromosomal features.



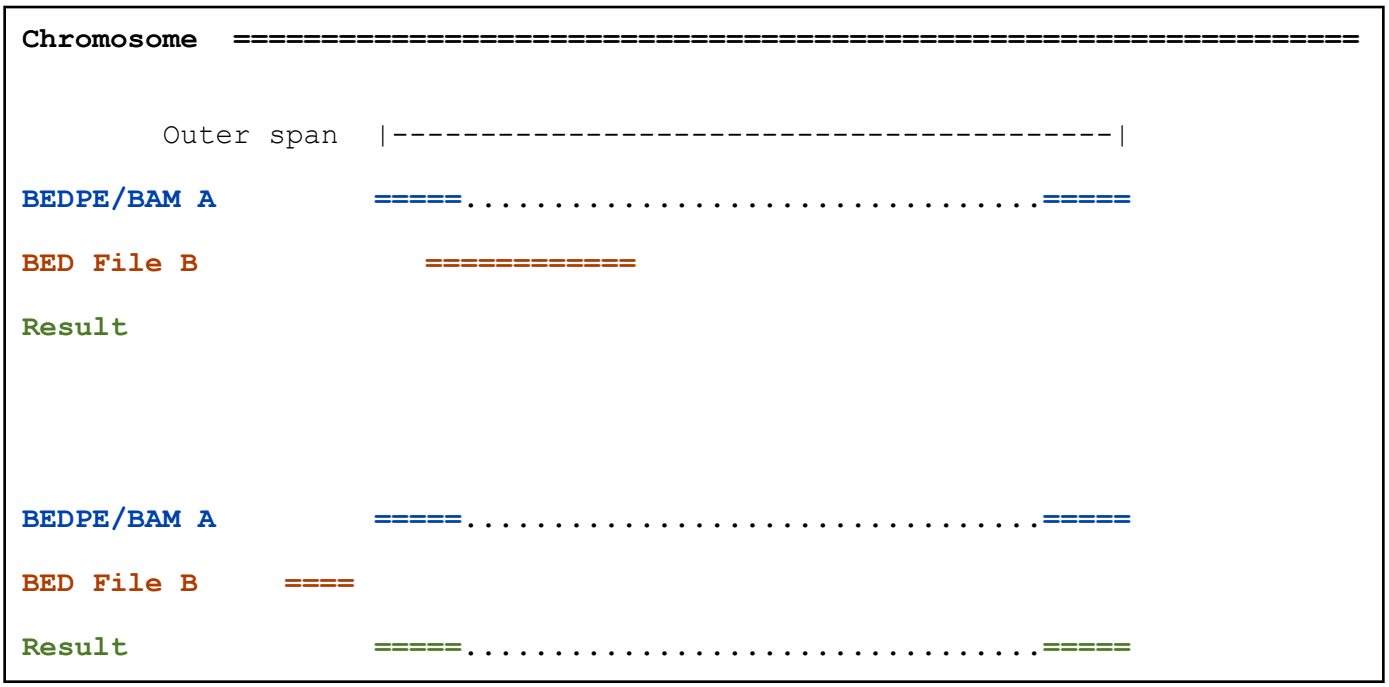
-type ospan: Report A if it's "*outer span*" overlaps B. Applicable only to intra-chromosomal features.



-type notispan: Report A only if it's "*inner span*" does not overlap B. Applicable only to intra-chromosomal features.



-type notospan: Report A if it's "*outer span*" overlaps B. Applicable only to intra-chromosomal features.

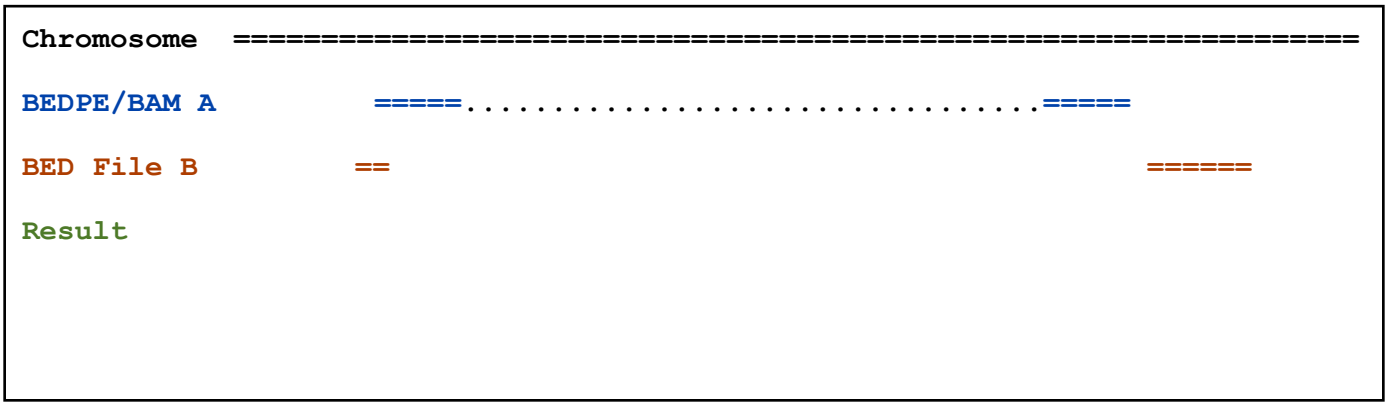


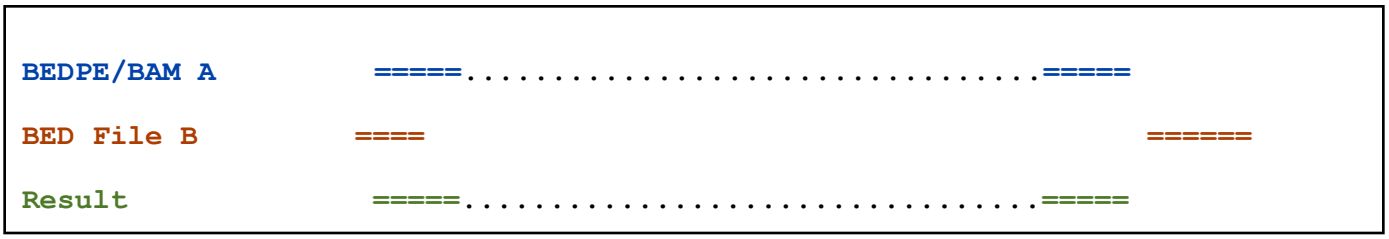
5.2.4 Requiring a minimum overlap fraction (-f)

By default, **pairToBed** will report an overlap between A and B so long as there is at least one base pair is overlapping on either end. 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 A. The –**f** option does exactly this. The **-f** option may also be combined with the -type option for additional control. For example, combining **-f 0.50** with **-type both** requires that both ends of A have at least 50% overlap with a feature in B.

For example, report A only at least 50% of one of the two ends is overlapped by B.

```
$ pairToBed -a A.bedpe -b B.bed -f 0.5
```



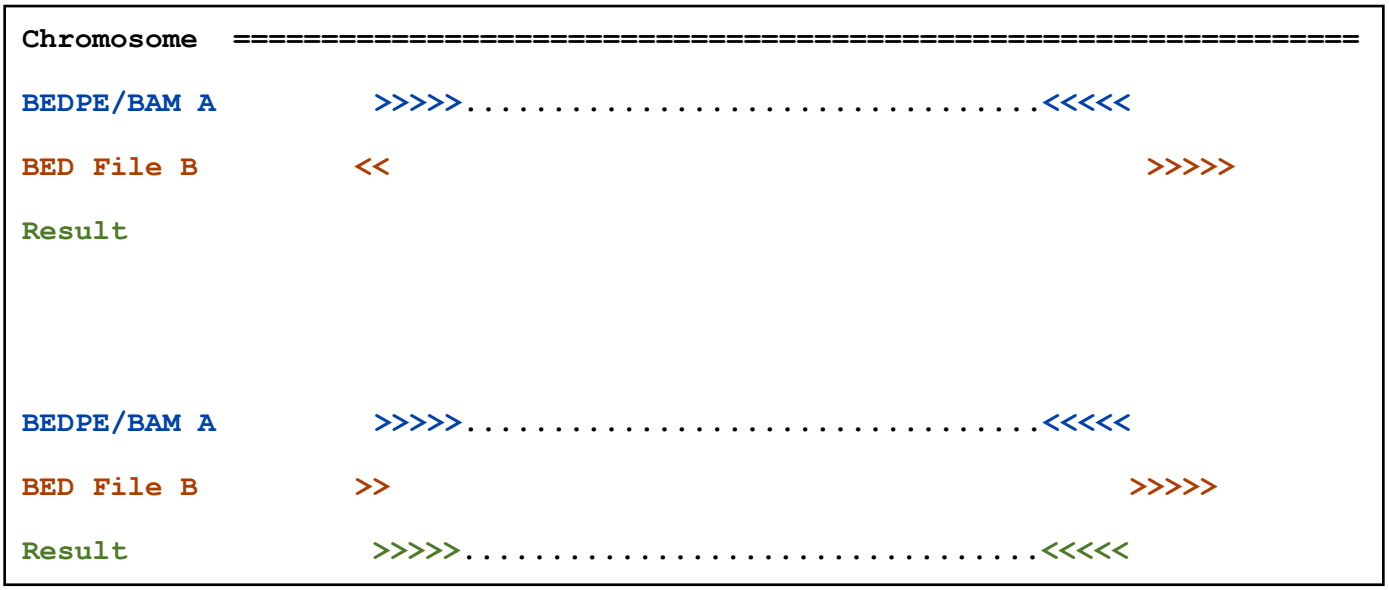


5.2.5 Enforcing "strandedness" (-s)

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

For example, report A only at least 50% of one of the two ends is overlapped by B.

```
$ pairToBed -a A.bedpe -b B.bed -s
```



5.2.6 Default is to write BAM output when using BAM input (-abam)

When comparing ***paired*** alignments in BAM format (**-abam**) to features in BED format (**-b**), **pairToBed** 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 both alignments for each aligned pair will be written to the BAM output.

For example:

```
$ pairToBed -abam pairedReads.bam -b simreps.bed | samtools view - | head -4
JOBU_0001:3:1:4:1060#0 99      chr10   42387928        29      50M     =       42393091        5   2   1   3
       A A A A A C G G A A T T A T C G A A T G G A A T C G A A G A G A A T C T T C G A A C G G A C C C G A
       dcgggggfbgfgdgggggggfdfgggcggggfcggcgggggggagfgbggc   XT:A:R  NM:i:5  SM:i:0  AM:i:0  X0:i:3  X1:i:
3      XM:i:5  XO:i:0  XG:i:0  MD:Z:0T0C33A5T4T3
JOBU_0001:3:1:4:1060#0 147     chr10   42393091        0       50M     =       42387928        -   5   2   1   3
       AAATGGAATCGAATGGAATCAACATCAAATGGAATCAAATGGAATCATTG   K   g   d   c   g   g   d   e   c   d   g
\d`ggfcgcggffcgggc^cgfgccgggfc^gcdgg\bg      XT:A:R  NM:i:2  SM:i:0  AM:i:0  X0:i:3  X1:i:13 XM:i:2  XO:i:
0      XG:i:0  MD:Z:21T14G13
JOBU_0001:3:1:8:446#0  99      chr10   42388091        9       50M     =       42392738        4   6   9   7
       GAATCGACTGGAATCATCATCGGATGGAAATGAATGGAATAATCATCGAA   f_Off`]IeYff`ffeddcfefcP`c_W\\R_]
_BBBBBBBBBBBBBBB      XT:A:U  NM:i:4  SM:i:0  AM:i:0  X0:i:1  X1:i:3  XM:i:4  XO:i:0  XG:i:0  MD:Z:
7A22C9C2T6
JOBU_0001:3:1:8:446#0  147     chr10   42392738        9       50M     =       42388091        -   4   6   9   7
       TTATCGAATGCAATCGAATGGAATTATCGAATGCAATCGAATAGAATCAT   df^ffec_JW[`MWceRec``fee`dcecfeeZae`c]
f^cNeecfccf^   XT:A:R  NM:i:1  SM:i:0  AM:i:0  X0:i:2  X1:i:2  XM:i:1  XO:i:0  XG:i:0  MD:Z:38A11
```

5.2.7 Output BEDPE format when using BAM input (-bedpe)

When comparing ***paired*** alignments in BAM format (**-abam**) to features in BED format (**-b**), **pairToBed** will optionally write the output in BEDPE format. That is, each alignment in the BAM file is converted to a 10 column BEDPE feature and if overlaps are found (or not) based on the user's criteria, the BAM alignment will be reported in BEDPE format. The BEDPE "name" field is comprised of the RNAME field in the BAM alignment. The "score" field is the mapping quality score from the BAM alignment.

For example:

```
$ pairToBed -abam pairedReads.bam -b simreps.bed -bedpe | head -5
chr10 42387927    42387977    chr10 42393090    42393140
      JOBU_0001:3:1:4:1060#0  29    +     -
chr10 42388090    42388140    chr10 42392737    42392787
      JOBU_0001:3:1:8:446#0   9     +     -
chr10 42390552    42390602    chr10 42396045    42396095
      JOBU_0001:3:1:10:1865#0 9     +     -
chrX  139153741   139153791   chrX  139159018   139159068
      JOBU_0001:3:1:14:225#0  37    +     -
chr4  9236903     9236953     chr4  9242032     9242082
      JOBU_0001:3:1:15:1362#0 0     +     -
```

5.3 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.

5.3.1 Usage and option summary

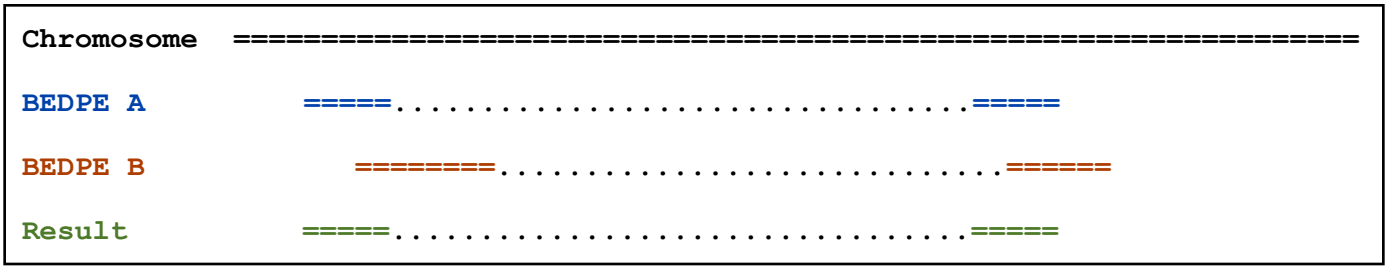
Usage: `$ pairToPair [OPTIONS] -a <BEDPE> -b <BEDPE>`

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. **neither** Report A if neither end of A overlaps B. **both** Report overlaps if both ends of A overlap B. *Default behavior.*

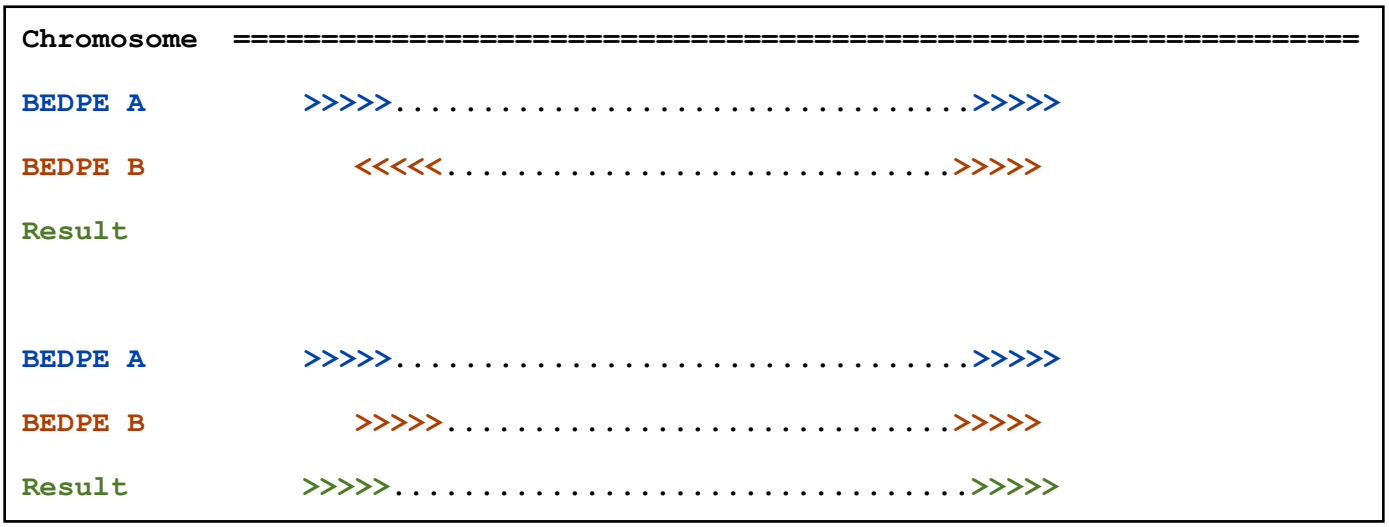
5.3.2 Default behavior

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.



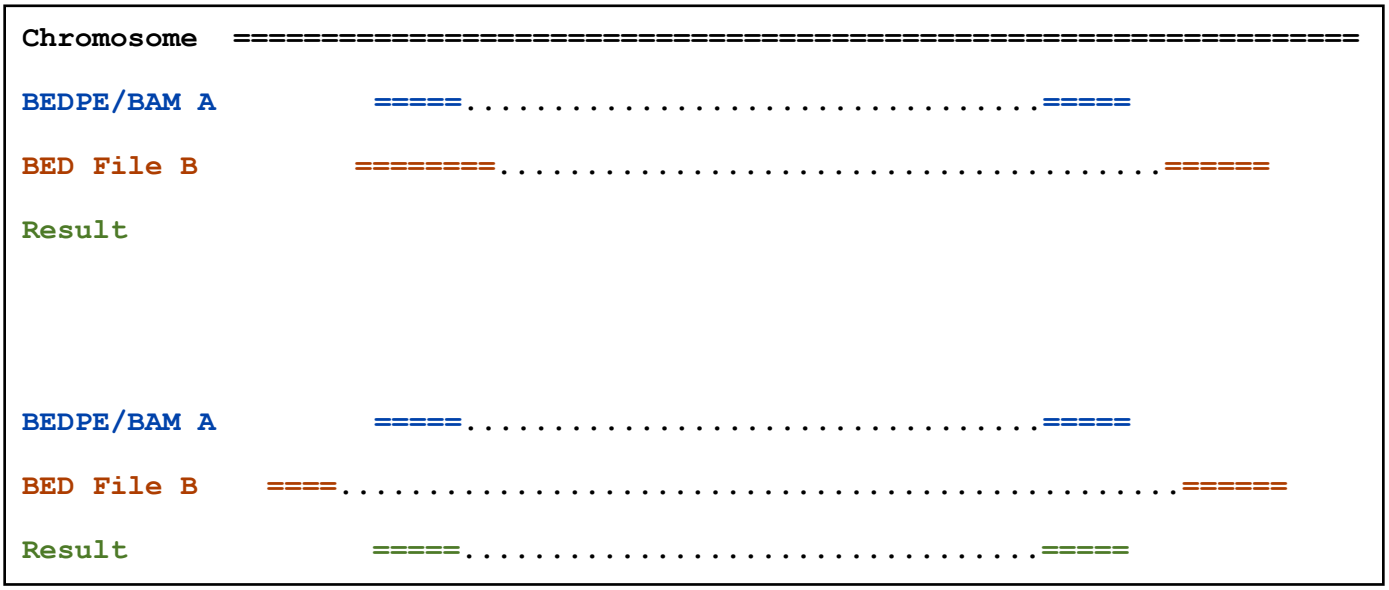
Default when strand information is present in both BEDPE files: Report A if *both* ends overlaps B ***on the same strands***.



5.3.3 Optional overlap requirements (-type neither)

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.



5.4 bamToBed

bamToBed is a general purpose tool that will convert sequence alignments in BAM format to either BED or BEDPE format. This enables one to convert BAM files for use with all of the other BEDTools.

5.4.1 Usage and option summary

Usage: `$ 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. Alignments with an insert size of 0 are ignored. When the mate alignments are ***inter***-chromosomal, the alignment where mate 1 is the primary alignment will be reported. Lastly, only pairs where both ends are mapped (i.e., not "orphans" or unmapped pairs) will be reported. *By default, this is disabled and the output will be reported in BED format.* **BAM files may be piped to bamToBed by specifying "–i stdin". See example below.**
-ed	Use the "edit distance" tag (NM) for the BED score field. Not available when writing BEDPE format. *By default, the score field will be the mapping quality.*

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. The "score" field is the mapping quality score from the BAM alignment, unless the –**ed** option is used.

Examples:

```
$ bamToBed -i reads.bam | head -5
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    -

$ bamToBed -i reads.bam -ed | head -5
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     -

$ bamToBed -i reads.bam -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 (BAM flag == 0x2) reads to BED format.

```
samtools view -bf 0x2 reads.bam  | 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    -
```

5.5 windowBed

Similar to **intersectBed, windowBed** searches for overlapping features in A and B. However, **windowBed** 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.

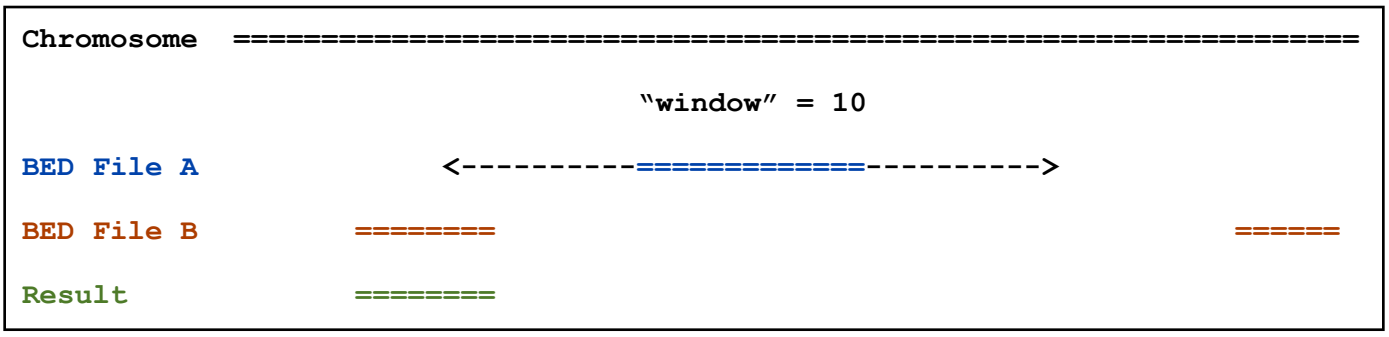
5.5.1 Usage and option summary

Usage: `$ windowBed [OPTIONS] -a <BED> -b <BED>`

Option	Description
-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.*
-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.

5.5.2 Default behavior

By default, **windowBed** 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. For example, in the figure below, feature B1 would be found, but B2 would not.



For example:

```
$ cat A.bed
chr1  100   200

$ cat B.bed
chr1  500   1000
chr1  1300  2000

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

5.5.3 Defining a custom window size (-w)

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

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

5.5.4 Defining assymteric windows (-l and -r)

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.

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

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

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

5.5.5 Defining assymteric windows based on strand (-sw)

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

$ windowBed -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
```

5.5.6 Enforcing "strandedness" (-sm)

This option behaves the same as the –s option for intersectBed while scanning for overlaps within the "window" surrounding A. See the discussion in the intersectBed section for details.

5.5.7 Reporting the presence of *at least one* overlapping feature (-u)

This option behaves the same as for intersectBed while scanning for overlaps within the "window" surrounding A. See the discussion in the intersectBed section for details.

5.5.8 Reporting the number of overlapping features (-c)

This option behaves the same as for intersectBed while scanning for overlaps within the "window" surrounding A. See the discussion in the intersectBed section for details.

5.5.9 Reporting the absence of any overlapping features (-v)

This option behaves the same as for intersectBed while scanning for overlaps within the "window" surrounding A. See the discussion in the intersectBed section for details.

5.6 closestBed

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.

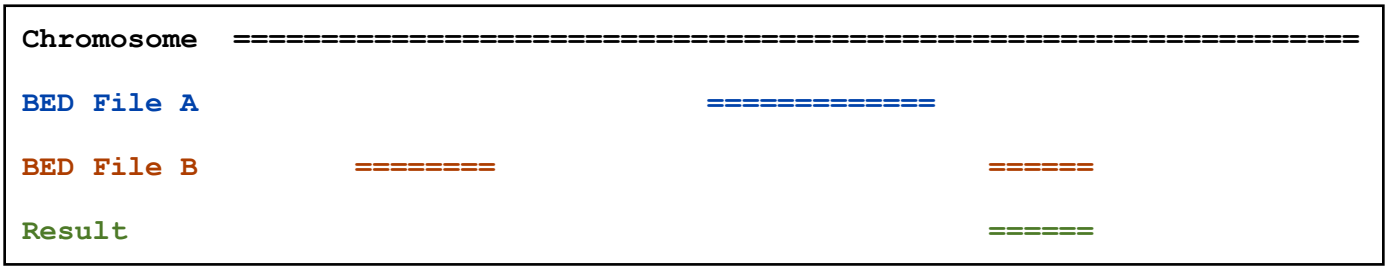
5.6.1 Usage and option summary

Usage: `$ closestBed [OPTIONS] -a <BED> -b <BED>`

Option	Description
-s	Force strandedness. That is, find the closest feature in B overlaps A on the same strand. *By default, this is disabled.*
-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.

5.6.2 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.



For example:

```
$ cat A.bed
chr1  100   200

$ cat B.bed
chr1  500   1000
chr1  1300  2000

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

5.6.3 Enforcing "strandedness" (-s)

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.

5.6.4 Controlling how ties for "closest" are broken (-t)

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   -
```

5.7 subtractBed

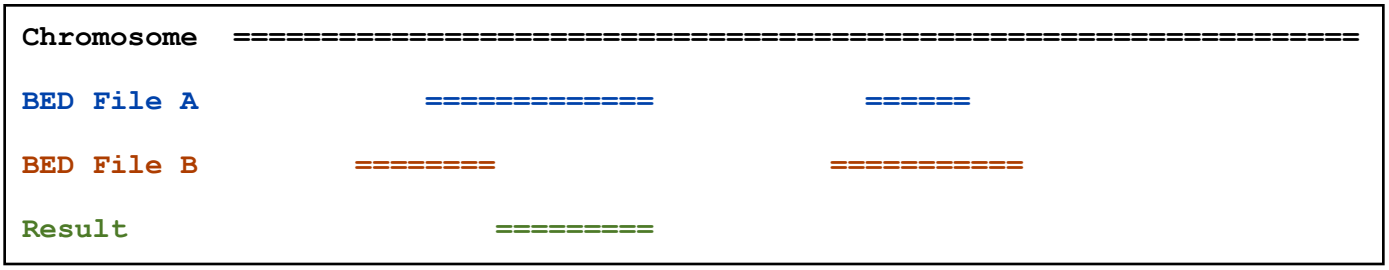
subtractBed searches for features in B that overlap A. If an overlapping feature is found in B, the overlapping portion is removed from A and the remaining portion of A is reported. If a feature in B overlaps all of a feature in A, the A feature will not be reported.

5.7.1 Usage and option summary

Usage: `$ subtractBed [OPTIONS] -a <BED> -b <BED>`

Option	Description
-f	Minimum overlap required as a fraction of A. Default is 1E-9 (i.e. 1bp).
-s	Force strandedness. That is, find the closest feature in B overlaps A on the same strand. *By default, this is disabled.*

5.7.2 Default behavior



For example:

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

$ cat B.bed
chr1  0     30
chr1  180   300

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

5.7.3 Requiring a minimal overlap fraction before subtracting (-f)

This option behaves the same as the –f option for intersectBed. In this case, subtractBed 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.

For example:

```
$ cat A.bed
chr1  100   200

$ cat B.bed
chr1  180   300

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

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

5.7.4 Enforcing "strandedness" (-s)

This option behaves the same as the –s option for intersectBed while scanning for features in B that should be subtracted from A. See the discussion in the intersectBed section for details.

5.8 mergeBed

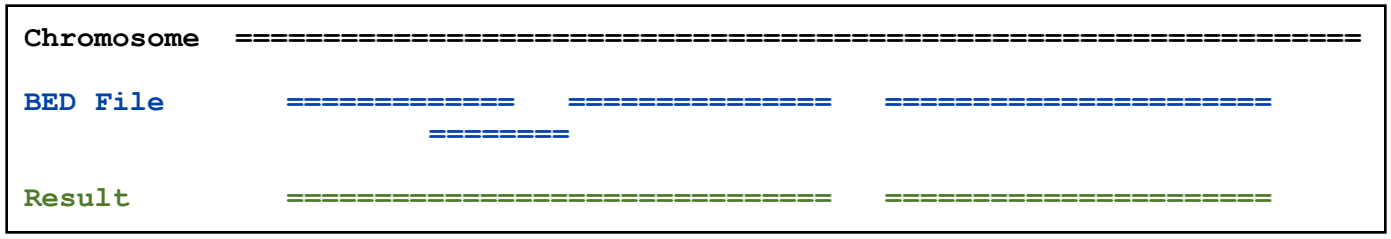
mergeBed combines overlapping or "book-ended" (that is, one base pair away) features in a BED file into a single feature which spans all of the combined features.

5.8.1 Usage and option summary

Usage: `$ mergeBed [OPTIONS] -i <BED>`

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.

5.8.2 Default behavior



For example:

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

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

5.8.3 Enforcing "strandedness" (-s)

This option behaves the same as the –s option for intersectBed while scanning for features that should be merged. Only features on the same strand will be merged. See the discussion in the intersectBed section for details.

5.8.4 Reporting the number of features that were merged (-n)

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.

For example:

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

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

5.8.5 Controlling how close two features must be in order to merge (-d)

By default, only overlapping or book-ended features are combined into a new feature. However, one can force mergeBed 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.

For example:

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

$ mergeBed -i A.bed
chr1  100   200
chr1  501   1000

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

5.8.6 Reporting the names of the features that were merged (-nms)

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 mergeBed output which lists (separated by semicolons) the names of the merged features.

For example:

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

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

5.9 coverageBed

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.

5.9.1 Usage and option summary

Usage: `$ coverageBed [OPTIONS] -a <BED> -b <BED>`

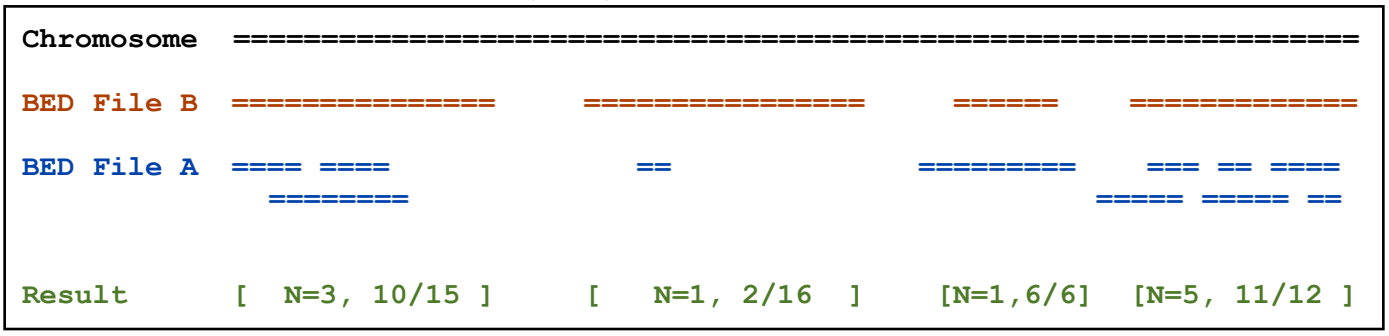
Option	Description
-s	Force strandedness. That is, only features in A are only counted towards coverage in B if they are the same strand. *By default, this is disabled and coverage is counted without respect to strand.*

5.9.2 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.



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
```

5.9.4 Calculating coverage by strand (-s)

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
```

5.10 genomeCoverageBed

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**).

5.10.1 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
-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.

5.10.2 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
```

5.10.3 Controlling the histogram's maximum depth (-max)

Using the –**max** option, **genomeCoverageBed** will "lump" all positions in the genome having feature coverage greather 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.

5.10.4 Reporting "per-base" genome coverage (-d)

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
```

5.11 fastaFromBed

fastaFromBed extracts sequences from a FASTA file for each of the intervals defined in a BED file. The headers in the input FASTA file must exactly match the chromosome column in the BED file.

5.11.1 Usage and option summary

Usage: $fastaFromBed **[OPTIONS]** -fi <input FASTA> -bed <BED> -fo <output FASTA>

Option	Description
-names	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.

5.11.2 Default behavior

fastaFromBed 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>".

For example:

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1  5     10

$ fastaFromBed -fi test.fa -bed test.bed -fo test.fa.out

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

5.11.3 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.

For example:

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1  5     10    myseq
```

```
$ fastaFromBed -fi test.fa -bed test.bed -fo test.fa.out -name
$ cat test.fa.out
>myseq
AAACC
```

5.11.4 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.

For example:

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1  5     10    myseq

$ fastaFromBed -fi test.fa -bed test.bed -fo test.fa.out.tab -name -tab
$ cat test.fa.out
myseq AAACC
```

5.12 maskFastaFromBed

maskFastaFromBed masks sequences in a FASTA file based on intervals defined in a BED file. The headers in the input FASTA file must exactly match the chromosome column in the BED 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.

5.12.1 Usage and option summary

Usage: `$maskFastaFromBed [OPTIONS] -fi <input FASTA> -bed <BED> -fo <output FASTA>`

NOTE: The input and output FASTA files must be different.

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.*

5.12.2 Default behavior

maskFastaFromBed 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.

For example:

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGG

$ cat test.bed
chr1  5     10

$ maskFastaFromBed -fi test.fa -bed test.bed -fo test.fa.out

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

5.12.3 Soft-masking the FASTA file.

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

For example:

```
$ cat test.fa
>chr1
AAAAAAAACCCCCCCCCCCCCGCTACTGGGGGGGGGGGGGGGGG
```

```
$ cat test.bed
chr1  5     10

$ maskFastaFromBed -fi test.fa -bed test.bed -fo test.fa.out -soft

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

5.13 shuffleBed

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

5.13.1 Usage and option summary

Usage: `$ shuffleBed [OPTIONS] -i <BED> -g <GENOME>`

Option	Description
-excl	A BED file of coordinates in which features from –i should ***not*** be placed (e.g., genome gaps).
-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.*

5.13.2 Default behavior

By default, **shuffleBed** 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

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

5.13.3 Requiring that features be shuffled on the *same* chromosome (-chrom)

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.

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

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

5.13.4 Excluding certain genome regions from shuffleBed

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. **shuffleBed** 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

$ shuffleBed -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

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

5.13.5 Defining a "seed" for the random replacement.

shuffleBed 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 **shuffleBed**. In turn, each execution of **shuffleBed** 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

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

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

. . .

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

5.14 slopBed

slopBed will increase the size of each feature in a BED 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*>}'**", **slopBed** will restrict the resizing to the size of the chromosome (i.e. no start $<$ 0 and no end $>$ chromosome size).

5.14.1 Usage and option summary

Usage: `$ slopBed [OPTIONS] -i <BED> -g <GENOME> [-b or (-l and -r)]`

Option	Description
-b	Increase the BED 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.

5.14.2 Default behavior

By default, **slopBed** will either add a fixed number of bases in each direction (**-b**) or an asymmetric number of bases in each direction (**-l** and **–r**).

For example:

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

$ cat my.genome
chr1  1000

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

$ slopBed -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, **slopBed** will "clip" the feature accordingly.

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

$ cat my.genome
chr1  1000

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

5.14.3 Resizing features according to strand

slopBed 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

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

5.15 sortBed

sortBed sorts a BED file by chromosome and other criteria.

5.15.1 Usage and option summary

Usage: `$ sortBed [OPTIONS] -i <BED>`

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).

5.15.2 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
```

5.15.3 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
```

5.16 linksBed

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

5.16.1 Usage and option summary

Usage: `$ linksBed [OPTIONS] -i <BED> > <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*

5.16.2 Default behavior

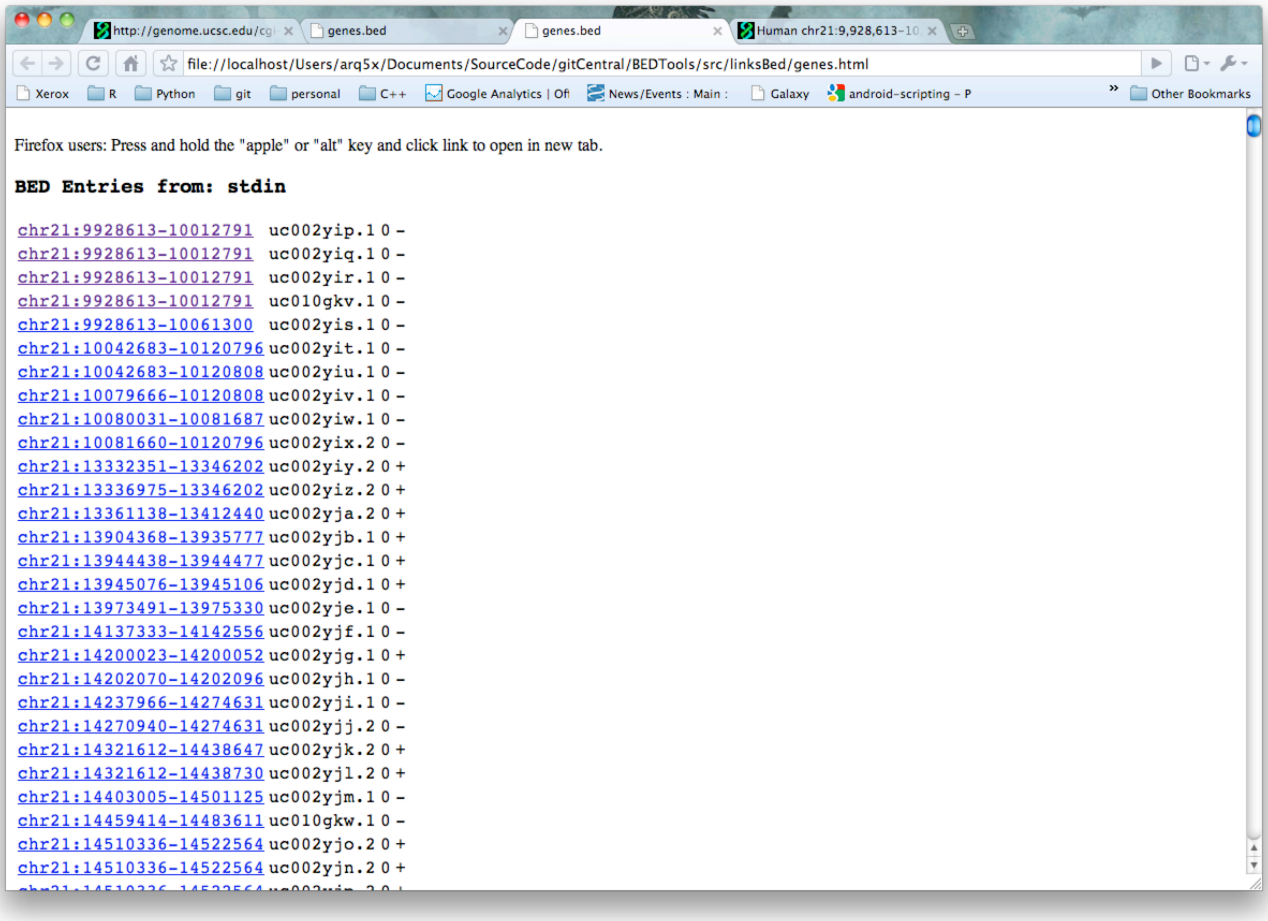
By default, **linksBed** creates links to the public UCSC Genome Browser.

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:



5.16.3 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
```

5.17 complementBed

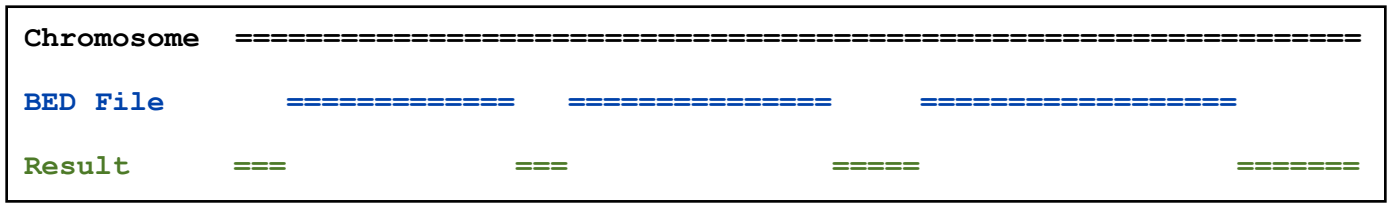
complementBed returns the intervals in a genome that are ***not*** by the features in a BED file. An example usage of this tool would be to return the intervals of the genome that are not annotated as a repeat.

5.17.1 Usage and option summary

Usage: `$ complementBed [OPTIONS] -i <BED> -g <GENOME>`

No additional options.

5.15.2 Default behavior



For example:

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

$ cat my.genome
chr1  1000

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

6. Example usage.

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

6.1 intersectBed

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

```
$ intersectBed -a reads.bed -b genes.bed
```

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

```
$ intersectBed -a reads.bed -b genes.bed -u
```

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

```
$ intersectBed -a reads.bed -b genes.bed -v
```

6.1.4 Report the number of genes that each alignment overlaps.

```
$ intersectBed -a reads.bed -b genes.bed -c
```

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

```
$ intersectBed -a reads.bed -b genes.bed -wa
```

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

```
$ intersectBed -a reads.bed -b genes.bed -wb
```

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

```
$ intersectBed -a reads.bed -b genes.bed -wa -wb
```

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

```
$ intersectBed -a exons.bed -b repeatMasker.bed -f 0.50
```

6.1.9 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.

```
$ intersectBed -a SV.bed -b segmentalDups.bed -f 0.50 -r
```

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

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

6.1.11 Retain only single-end BAM alignments that overlap exons.

```
$ intersectBed -abam reads.bam -b exons.bed > reads.touchingExons.bam
```

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

```
$ intersectBed -abam reads.bam -b SSRs.bed -v > reads.noSSRs.bam
```

6.2 pairToBed

6.2.1 Return all structural variants (in BEDPE format) that overlap with genes on either end.

```
$ pairToBed -a sv.bedpe -b genes > sv.genes
```

6.2.1 Return all structural variants (in BEDPE format) that overlap with genes on both end.

```
$ pairToBed -a sv.bedpe -b genes -type both > sv.genes
```

6.2.3 Retain only paired-end BAM alignments where neither end overlaps simple sequence repeats.

```
$ pairToBed -abam reads.bam -b SSRs.bed -type neither > reads.noSSRs.bam
```

6.2.4 Retain only paired-end BAM alignments where both ends overlap segmental duplications.

```
$ pairToBed -abam reads.bam -b segdups.bed -type both > reads.SSRs.bam
```

6.2.5 Retain only paired-end BAM alignments where neither or one and only one end overlaps segmental duplications.

```
$ pairToBed -abam reads.bam -b segdups.bed -type notboth > reads.notbothSSRs.bam
```

6.3 pairToPair

6.3.1 Find all SVs (in BEDPE format) in sample 1 that are also in sample 2.

```
$ pairToPair -a 1.sv.bedpe -b 2.sv.bedpe | cut -f 1-10 > 1.sv.in2.bedpe
```

6.3.2 Find all SVs (in BEDPE format) in sample 1 that are not in sample 2.

```
$ pairToPair -a 1.sv.bedpe -b 2.sv.bedpe -type neither | cut -f 1-10 >
1.sv.notin2.bedpe
```

6.4 bamToBed

6.4.1 Convert BAM alignments to BED format.

```
$ bamToBed -i reads.bam > reads.bed
```

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

```
$ bamToBed -i reads.bam -ed > reads.bed
```

6.4.2 Convert BAM alignments to BEDPE format.

```
$ bamToBed -i reads.bam -bedpe > reads.bedpe
```

6.5 windowBed

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

```
$ windowBed -a CNVs.bed -b genes.bed -w 10000
```

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

```
$ windowBed -a CNVs.bed -b genes.bed -l 10000 -r 5000
```

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

```
$ windowBed -a genes.bed -b snps.bed -l 5000 -r 1000 -sw
```

6.6 closestBed

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

6.6.1 Find the closest ALU to each gene.

```
$ closestBed -a genes.bed -b ALUs.bed
```

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

```
$ closestBed -a genes.bed -b ALUs.bed -t first
```

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

```
$ closestBed -a genes.bed -b ALUs.bed -t last
```

6.7 subtractBed

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

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

```
$ subtractBed -a genes.bed -b introns.bed
```

6.8 mergeBed

6.8.1 Merge overlapping repetitive elements into a single entry.

```
$ mergeBed -i repeatMasker.bed
```

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

```
$ mergeBed -i repeatMasker.bed -n
```

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

```
$ mergeBed -i repeatMasker.bed -d 1000
```

6.9 coverageBed

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

```
$ coverageBed -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
```

6.9.2 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.

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

6.9.3 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.

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

6.10 complementBed

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

```
$ complementBed -i repeatMasker.bed -g hg18.genome
```

6.11 shuffleBed

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

```
$ shuffleBed -i variants.bed -g hg18.genome -excl genome_gaps.bed
```

6.11.2 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.

```
$ shuffleBed -i variants.bed -g hg18.genome -excl genome_gaps.bed -chrom
```

7. Advanced usage.

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

```
# Add 500 bp up and downstream of each probe
$ slopBed -i probes.bed -b 500 > probes.500bp.bed

# Get a BED file of all regions not covered by the probes (+500 bp up/down)
$ complementBed -i probes.500bp.bed -g hg18.genome > probes.500bp.complement.bed

# Create a masked genome where all bases are masked except for the probes +500bp
$ maskFastaFromBed -in hg18.fa -bed probes.500bp.complement.bed -fo hg18.probe-
complement.masked.fa
```

7.2 Screening for novel SNPs.

```
# Find all SNPs that are not in dbSnp and not in the latest 1000 genomes calls
$ intersectBed -a snp.calls.bed -b dbSnp.bed -v | intersectBed -a stdin -b 1KG.bed
-v > snp.calls.novel.bed
```

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

```
# By default, coverageBed 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.
$ intersectBed -a features.bed -b windows.bed -f 1.0 | coverageBed -a stdin -b
windows.bed > windows.bed.coverage
```

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.
$ bamToBed -i reads.bam | coverageBed -a stdin -b exons.bed > exons.bed.coverage

# Take it a step further and require that coverage be from properly-paired reads.
$ samtools view -bf 0x2 reads.bam | bamToBed -i stdin | coverageBed -a stdin -b
exons.bed > exons.bed.proper.coverage
```

7.5 Computing coverage separately for each strand.

```
# Use grep to only look at forward strand features (i.e. those that end in "+").
$ bamToBed -i reads.bam | grep \+$ | coverageBed -a stdin -b genes.bed >
genes.bed.forward.coverage

# Use grep to only look at reverse strand features (i.e. those that end in "-").
$ bamToBed -i reads.bam | grep \-$ | coverageBed -a stdin -b genes.bed >
genes.bed.forward.coverage
```

7.6 Find structural variant calls that are private to one sample.

```
$ pairToPair -a sample1.sv.bedpe -b othersamples.sv.bedpe -type neither >
sample1.sv.private.bedpe
```

7.7 Exclude SV deletions that appear to be ALU insertions in the reference genome.

```
# We'll require that 90% of the inner span of the deletion be overlapped by a
recent ALU.
$ pairToBed -a deletions.sv.bedpe -b ALUs.recent.bed -type notispan -f 0.80 >
deletions.notALUsinRef.bedpe
```