pybedtools Documentation

Release 0.2.0dev

Ryan Dale

January 11, 2011

CONTENTS

Contents:

CHAPTER

ONE

PYBEDTOOLS OVERVIEW AND EXAMPLES

Python wrapper for Aaron Quinlan's BEDtools (http://code.google.com/p/bedtools/).

pybedtools.bedtool

1.1 Installation

Use pip to install the latest from the Github repo:

```
pip install pybedtools
```

or easy_install:

```
easy_install pybedtools
```

1.2 Quick examples for the impatient

Get the sequences of the 100 bp on either side of features (with automatic download of chromSizes from UCSC for dm3 genome). This assumes you have a local copy of the entire dm3 genome as dm3.fa:

```
import pybedtools
pybedtools.bedtool('in.bed').slop(genome='dm3',l=100,r=100).subtract('in.bed')
flanking_features.sequence(fi='dm3.fa').save_seqs('flanking.fa')
```

Or, get values for a 3-way Venn diagram of overlaps:

```
import pybedtools
a = pybedtools.bedtool('a.bed')
b = pybedtools.bedtool('b.bed')
c = pybedtools.bedtool('c.bed')

(a-b-c).count()  # unique to a
(a+b-c).count()  # in a and b, not c
(a+b+c).count()  # common to all
# ... and so on, for all the combinations.
```

Intersections, plus adding track names:

```
import pybedtools
a = pybedtools.bedtool('a.bed')
a.intersect('b.bed').saveas('a-and-b.bed', trackline="track name='a and b'")
```

Creating a `pybedtools.bedtool`:

```
>>> import pybedtools
>>> a = pybedtools.bedtool('chrX 1 100', from_string=True).saveas('a.bed')

>>> b = pybedtools.bedtool('a.bed')
>>> print b
chrX     1       100
```

1.3 Why use `pybedtools`?

Using this module allows you to use BEDtools directly from your Python code without awkward system calls. The provided `pybedtools.bedtool` class wraps the BEDtools command line programs in an intuitive and easy-to-use interface. As a quick illustration of the streamlining possible, here's how to get the number of features shared between a.bed and b.bed, those unique to a.bed, and those unique to b.bed:

```
from pybedtools import bedtool
a = bedtool('a.bed')
b = bedtool('b.bed')
(a+b).count()     # shared in a and b
(a-b).count()     # unique to a
(b-a).count()     # unique to b
```

In contrast, here's how you'd do the same from the command line:

```
intersectBed -a a.bed -b b.bed -u | wc -l   # shared in a and b
intersectBed -a a.bed -b b.bed -u | wc -l   # unique to a
intersectBed -a b.bed -b a.bed -u | wc -l   # unique to b
```

To do the same thing in Python, *each* of these lines would have to be wrapped in awkward, piped `subprocess.Popen` calls:

```
p1 = subprocess.Popen(['intersectBed','-a','a.bed','-b','b.bed','-u'], stdout=subprocess.PIPE)
p2 = subprocess.Popen(['wc','-l'], stdin=subprocess.PIPE)
results = p2.communicate()[0]
count = results.split()[-1]
```

See, `a+b` is much easier!

1.4 Translation between BEDTools programs and `pybedtools.bedtool` methods

This table summarizes the translation between BEDTools program names `bedtool` method names.

BEDTools program	pybedtools.bedtool method
intersectBed	bedtool.intersect()
subtractBed	bedtool.subtract()
fastaFromBed	bedtool.sequence()
slopBed	bedtool.slop()
mergeBed	bedtool.merge()
closestBed	bedtool.closest()
windowBed	bedtool.window()
groupBy	bedtool.groupBy()
shuffleBed	bedtool.shuffle()
sortBed	bedtool.sort()

There are also methods unique to pybedtools that provide additional usefulness:

- bedtool.size_filter()
- bedtool.random_subset()
- bedtool.saveas()
- bedtool.cat()
- bedtool.randomstats()
- bedtool.get_genome()
- bedtool.save_seqs()
- bedtool.count()
- bedtool.features()
- bedtool.lengths()

1.5 General usage

```
>>> from pybedtools import bedtool
>>> a = '''
...         chrX 1   100
...         chrX 200 500
...         chrY 499 600
... '''
>>> b = '''
...         chrX 10  60
...         chrY 200 500
... '''
>>> a = bedtool(a, from_string=True).saveas('a.bed')
>>> b = bedtool(b, from_string=True).saveas('b.bed')
```

1.5.1 Arguments are the same as BEDtools command line programs

The methods in the bedtool class mimic the command line usage of BEDtools as closely as possible. Any flag that can be passed to the BEDtools programs can be passed as a keyword argument in the corresponding pybedtools.bedtool method.

On/off switches (e.g., -c, -u, or -v for intersectBed) are called with a boolean kwarg; others (like -f for intersectBed) are passed in like a normal kwarg with appropriate values. For example:

```
a = bedtool('in.bed')
a.intersect('other.bed', v=True, f=0.5)
```

Typically, for convenience `-i`, `-a`, and `-b` are already passed for you although you can override this by passing these keyword arguments explicitly. The second line above could have equivalently been called as:

```
a.intersect(a='in.bed', b='other.bed', v=True, f=0.5)
```

Conveniently, the docstring for a method automatically includes the help text of the original `BEDtools` program, so you can check which kwargs you can use directly from the interpreter.

1.5.2 Typical workflow includes temporary files

Typical workflow is to set up a bedtool object with a bed file you already have:

```
a = bedtool('in.bed')
```

a now references the file `in.bed` on disk.

Using BEDtools from the command line, we might use the following in order to get a new bed file of the intersection of this file with another bed file, `other.bed` but only returning uniquely intersecting features from `in.bed`:

```
intersectBed -a in.bed -b other.bed -u > intersection.bed
```

Using `pybedtools`:

```
b = a.intersect('other.bed', u=True)
```

This creates a new temp file in `/tmp` by default, but you can change where temp files are saved using `pybedtools.set_tempdir()`. To save a file explicitly and optionally add a trackline that will label it in a genome browser, use `saveas()`:

```
b.saveas('intersection.bed',trackline='track name="intersection"')
```

1.5.3 Cleaning up temporary files

When you're done, it's a good idea to clean up temporary files. Temp files are never deleted until you explicitly say so:

```
pybedtools.cleanup()
```

If you forget to call `pybedtools.cleanup()`, you can always manually delete the files from your temp dir (typically `/tmp`), though you can specify this with `pybedtools.set_tempdir()`. They are the files that follow the pattern `pybedtools.*.tmp`.

1.5.4 Chaining together commands

Most methods return new `bedtool` objects, allowing you to chain things together. This means that you can chain commands together, just like piping things together on the command line. To give you a flavor of this, here's how you would get 10 random centers of features that are unique to the file `other.bed`:

```
b = a.intersect('other.bed',v=True).feature_centers(100).random_subset(10)
```

1.6 Individual methods

1.6.1 bedtool.intersect()

Examples of how to use `bedtool.intersect()`.

First, we set up the files to use:

```
>>> # Set up the bedtools
>>> a = bedtool('a.bed')
>>> b = bedtool('b.bed')
```

Here's what they look like:

```
>>> print a
chrX         1       100
chrX         200     500
chrY     499 600

>>> print b
chrX         10      60
chrY         200     500
```

Default is to only report the part that intersects:

```
>>> print a.intersect(b)
chrX         10      60
chrY     499 500
```

Use the *-u* flag of `intersectBed` to report full features in `a.bed` that intersected `b.bed`:

```
>>> print a.intersect(b, u=True)
chrX   1    100
chrY   499  600
```

Or the opposite – features in `a.bed` that are not in `b.bed`:

```
>>> print a.intersect(b, v=True)
chrX         200     500
```

Report features in `a.bed`, attaching an additional column indicating how many features in `b.bed` intersected:

```
>>> print a.intersect(b, c=True)
chrX         1       100     1
chrX         200     500     0
chrY         499     600     1
```

You can retrieve these counts later using the `bedtool.counts()` method:

```
>>> result = a.intersect(b, c=True)
>>> print result.counts()
[1, 0, 1]
```

1.7 Example: Flanking seqs

The `slop()` method (which calls `slopBed`) needs a chromosome size file. If you specify a genome name to the `slop()` method, it will retrieve this file for you automatically from the UCSC Genome Browser MySQL database.

```
import pybedtools
a = pybedtools.bedtool('in.bed')
extended = a.slop(genome='dm3',l=100,r=100)
flanking = extended.subtract(a).saveas('flanking.bed')
flanking.sequence(fi='dm3.fa')
flanking.save_seqs('flanking.fa')
```

Or, as a one-liner:

```
pybedtools.bedtool('in.bed').slop(genome='dm3',l=100,r=100).subtract(a).sequence(fi='dm3.fa').save_se
```

Don't forget to clean up!:

```
pybedtools.cleanup()
```

1.8 Example: Region centers that are fully intergenic

Useful for, e.g., motif searching:

```
a = pybedtools.bedtool('in.bed')

# Sort by score
a = a.sorted(col=5,reverse=True)

# Exclude some regions
a = a.subtract('regions-to-exclude.bed')

# Get 100 bp on either side of center
a = a.peak_centers(100).saveas('200-bp-peak-centers.bed')
```

1.9 Example: Histogram of feature lengths

Note that you need matplotlib installed to plot the histogram.

```
import pylab as p
a = pybedtools.bedtool('in.bed')
p.hist(a.lengths(),bins=50)
p.show()
```

1.10 Selected examples from http://code.google.com/p/bedtools/wiki/UsageAdv

Command line:

```
intersectBed -a snp.calls.bed -b dbSnp.bed -v | intersectBed -a stdin -b 1KG.bed -v > snp.calls.novel
```

Same thing, in `pybedtools`: a = pybedtools.bedtool('snp.calls.bed') a.intersect('dbSnp.bed',v=True).intersect('1KG.bed',v=True).s

CHAPTER

TWO

AUTODOC

class pybedtools.**bedtool**(*fn*, *genome=None*, *from_string=False*)

Wrapper around BEDtools suite of programs; also contains many useful methods for more detailed work with BED files.

Example usage:

```
>>> from pybedtools import bedtool
>>> s = '''
... chrX  1  100
... chrX 25  800
... '''
>>> a = bedtool(s,from_string=True).saveas('a.bed')
```

Or, from an existing bed file:

```
>>> a = bedtool('a.bed')

>>> a = '''
...         chrX 1   100
...         chrX 200 500
... '''
>>> b = '''
...         chrX 10  60
...         chrY 200 500
... '''
>>> a = bedtool(a, from_string=True).saveas('a.bed')
>>> b = bedtool(b, from_string=True).saveas('b.bed')
```

cat(*other*, *postmerge=True*, ***kwargs*)

Concatenates two bedtools objects (or an object and a file) and does an optional post-merge of the features.

Use *postmerge=False* if you want to keep features separate.

TODO:

currently truncates at BED3 format!

kwargs are sent to bedtool.merge().

Example usage:

```
a = bedtool('in.bed')

# concatenate and merge features together if they overlap and are
# on the same strand
b = a.cat('other.bed', s=True)
```

closest(*other*, ***kwargs*)

pybedtools help:

Return a new bedtool object containing closest features in *other*. Note that the resulting file is no longer a valid BED format; use the special "_closest" methods to work with the resulting file.

Example usage:

```
a = bedtool('in.bed')

# get the closest feature in 'other.bed' on the same strand
b = a.closest('other.bed', s=True)
```

Original BEDtools program help:

Program: closestBed (v2.10.0) Authors: Aaron Quinlan (aaronquinlan@gmail.com)

Erik Arner, Riken

Summary: For each feature in A, finds the closest feature (upstream or downstream) in B.

Usage: closestBed [OPTIONS] -a <bed/gff/vcf> -b <bed/gff/vcf>

Options:

-s	Force strandedness. That is, find the closest feature in B that overlaps A on the same strand. - By default, overlaps are reported without respect to strand.
-d	In addition to the closest feature in B, report its distance to A as an extra column. - The reported distance for overlapping features will be 0.
-t	How ties for closest feature are handled. This occurs when two features in B have exactly the same overlap with A. By default, all such features in B are reported. Here are all the options: - "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.

Notes: Reports "none" for chrom and "-1" for all other fields when a feature is not found in B on the same chromosome as the feature in A. E.g. none -1 -1

count()

Number of features in BED file. Does the same thing as len(self), which actually just calls this method.

Only counts the actual features. Ignores any track lines, browser lines, lines starting with a "#", or blank lines.

Example usage:

```
a = bedtool('in.bed')
a.count()
```

counts()

After running `bedtool.intersect()` with the kwarg *c=True*, use this method to return a list of the count of features in "b" that intersected each feature in "a".

Example usage:

```
a = bedtool('in.bed')
b = a.intersect('other.bed', c=True)
counts = b.counts()

# assuming you have matplotlib installed, plot a histogram

import pylab
pylab.hist(counts)
pylab.show()
```

feature_centers(*n*, *report_smaller=True*)

Returns a new bedtools object with just the centers of size n extracted from this object's features.

If *report_smaller* is True, then report features that are smaller than *n*. Otherwise, ignore them.

Example usage:

```
a = bedtool('in.bed')

# 5bp on either side of the center of each feature
b = a.feature_centers(100)
```

features()

Returns an iterator of `bedfeature` objects.

get_genome(*genome*)

Download chrom size info for *genome* from UCSC, removes the header line, and saves in a temp file. Could be useful for end users, but mostly called internally by `bedtool.slop()` and other methods that need the genome file.

Example usage:

```
a = bedtool('in.bed')
fn = a.get_genome('dm3')
```

groupBy(***kwargs*)

pybedtools help:

Original BEDtools program help:

Program: groupBy (v2.10.0) Author: Aaron Quinlan (aaronquinlan@gmail.com) Summary: Summarizes a dataset column based upon

common column groupings. Akin to the SQL "group by" command.

Usage: groupBy -i <input> -g <group**column(s)> -c <op**column(s)> -o <ops>

Options:

-i Input file. Use "stdin" for pipes.

-g -grp Specify the columns (1-based) for the grouping. The columns must be comma separated. - Default: 1,2,3

-c -opCols Specify the column (1-based) that should be summarized.

- Required.

-o -ops Specify the operation that should be applied to opCol.

Valid operations: sum, count, min, max, mean, median, mode, antimode, stdev, sstdev (sample standard dev.), collapse (i.e., print a comma separated list), freqdesc (i.e., print desc. list of values:freq) freqasc (i.e., print asc. list of values:freq)

- Default: sum

Examples: $ cat ex1.out chr1 10 20 A chr1 15 25 B.1 1000 chr1 10 20 A chr1 25 35 B.2 10000

$ groupBy -i ex1.out -g 1,2,3,4 -c 9 -o sum chr1 10 20 A 11000

$ groupBy -i ex1.out -grp 1,2,3,4 -opCols 9,9 -ops sum,max chr1 10 20 A 11000 10000

$ groupBy -i ex1.out -g 1,2,3,4 -c 8,9 -o collapse,mean chr1 10 20 A B.1,B.2, 5500

Notes:

1. The input file/stream should be sorted/grouped by the -grp. columns
2. If -i is unspecified, input is assumed to come from stdin.

head(*n=10*)
Prints the first *n* lines

intersect(*other*, ***kwargs*)
pybedtools help:

Intersect with another BED file. If you want to use BAM, specify *abam='filename.bam'*. Returns a new bedtool object.

Example usage:

```
# create new bedtool object
a = bedtool('in.bed')

# get overlaps with "other.bed"
overlaps = a.intersect('other.bed')

# use v=True to get the inverse, or those unique to in.bed
unique_to_a = a.intersect('other.bed', v=True)

# features unique to "other.bed"
unique_to_other = bedtool('other.bed').intersect(a, v=True)
```

Original BEDtools program help:

Program: intersectBed (v2.10.0) Author: Aaron Quinlan (aaronquinlan@gmail.com) Summary: Report overlaps between two feature files.

Usage: intersectBed [OPTIONS] -a <bed/gff/vcf> -b <bed/gff/vcf>

Options:

-abam	The A input file is in BAM format. Output will be BAM as well.
-ubam	Write uncompressed BAM output. Default is to write compressed BAM.
-bed	When using BAM input (-abam), write output as BED. The default is to write output in BAM when using -abam.
-wa	Write the original entry in A for each overlap.
-wb	Write the original entry in B for each overlap. - Useful for knowing **what** A overlaps. Restricted by -f and -r.

-wo	Write the original A and B entries plus the number of base pairs of overlap between the two features. - Overlaps restricted by -f and -r. Only A features with overlap are reported.
-wao	Write the original A and B entries plus the number of base pairs of overlap between the two features. - Overlapping features restricted by -f and -r. However, A features w/o overlap are also reported with a NULL B feature and overlap = 0.
-u	Write the original A entry **once** if **any** overlaps found in B. - In other words, just report the fact >=1 hit was found. - Overlaps restricted by -f and -r.
-c	For each entry in A, report the number of overlaps with B. - Reports 0 for A entries that have no overlap with B. - Overlaps restricted by -f and -r.
-v	Only report those entries in A that have **no overlaps** with B. - Similar to "grep -v" (an homage).
-f	Minimum overlap required as a fraction of A. - Default is 1E-9 (i.e., 1bp). - FLOAT (e.g. 0.50)
-r	Require that the fraction overlap be reciprocal for A and B. - In other words, if -f is 0.90 and -r is used, this requires that B overlap 90% of A and A **also** overlaps 90% of B.
-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.
-split	Treat "split" BAM or BED12 entries as distinct BED intervals.

intersection_report(*other*, *basename=True*, ***kwargs*)

Prints a report of the reciprocal intersections with another bed file or `bedtool` object.

If *basename* is True (default), only prints the basename of the file and not the whole path.

a = bedtool('in.bed') a.intersection_report('other.bed')

lengths()

Returns a list of feature lengths.

Example usage:

```
a = bedtool('in.bed')

lengths = a.lengths()

# if you have pylab installed, plot a histogram
import pylab
pylab.hist(lengths)
pylab.show()
```

merge(**kwargs)

pybedtools help:

Merge overlapping features together. Returns a new bedtool object.

Example usage:

```
a = bedtool('in.bed')

# allow merging of features 100 bp apart
b = a.merge(d=100)
```

Original BEDtools program help:

Program: mergeBed (v2.10.0) Author: Aaron Quinlan (aaronquinlan@gmail.com) Summary: Merges overlapping BED/GFF/VCF entries into a single interval.

Usage: mergeBed [OPTIONS] -i <bed/gff/vcf>

Options:

-s	Force strandedness. That is, only merge features that are the same strand. - By default, merging is done without respect to strand.
-n	Report the number of BED entries that were merged. - Note: "1" is reported if no merging occurred.
-d	Maximum distance between features allowed for features to be merged. - Def. 0. That is, overlapping & book-ended features are merged. - (INTEGER)
-nms	Report the names of the merged features separated by semicolons.

newshuffle()

Quite fast implementation of shuffleBed; assumes 'dm3' as the genome and assumes shuffling within chroms.

Example usage:

```
a = bedtool('in.bed')

# randomly shuffled version of "a"
b = a.newshuffle()
```

This is equivalent to the following command-line usage of `shuffleBed`:

```
shuffleBed -i in.bed -g dm3.genome -chrom -seed $RANDOM > /tmp/tmpfile
```

normalized_counts()

After running `bedtool.intersect()` with the kwarg *c=True*, use this method to return a list of the density of features in "b" that intersected each feature in "a".

This takes the counts in each feature and divides by the bp in that feature.

Example usage:

```
a = bedtool('in.bed')
b = a.intersect('other.bed', c=True)
counts = b.normalized_counts()

# assuming you have matplotlib installed, plot a histogram
```

```
import pylab
pylab.hist(counts)
pylab.show()
```

parse_kwargs(***kwargs*)
Given a set of keyword arguments, turns them into a command line-ready list of strings. E.g., the kwarg dict:

```
kwargs = dict(c=True,f=0.5)
```

will be returned as:

```
['-c','-f','0.5']
```

If there are symbols (e.g., "|"), then the parameter is quoted."

print_randomstats(*other*, *iterations*, *intersectkwargs={}*)
Nicely prints the reciprocal randomization of two files.

print_sequence()
Print the sequence that was retrieved by the `bedtool.sequence()` method.

See usage example in `bedtool.sequence()`.

random_subset(*n*)
Returns a new bedtools object containing a random subset of the features in this subset. Currently does so by reading in all features; future updates should fix this to something more robust (e.g., newlines in a memory map)

Example usage:

```
a = bedtool('in.bed')

# Choose 5 random features from 'in.bed'
b = a.random_subset(5)
```

randomintersection(*other*, *iterations*, *intersectkwargs={}*)
Performs *iterations* shufflings of self, each time intersecting with *other*. *intersectkwargs* are passed to self.intersect(). Returns a list of integers where each integer is the number of intersections of one shuffled file with *other*.

Example usage:

```
r = bedtool('in.bed').randomintersection('other.bed', 100, {'u': True})
```

randomstats(*other*, *iterations*, *intersectkwargs={}*)
Sends args to `bedtool.randomintersection()` and compiles results into a dictionary with useful stats. Requires scipy and numpy.

Example usage:

```
a = bedtool('in.bed')

# Randomization results from 100 iterations, using the u=True kwarg (report
# features in "a" only once for each intersection).
results = a.randomstats('other.bed', iterations=100, intersectkwargs={'u':True})
```

rename_features(*new_name*)
Forces a rename of all features. Useful for if you have a BED file of exons and you want all of them to have the name "exon".

save_seqs (*fn*)
Save sequences of features in this bedtool object as a fasta file *fn*.

In order to use this function, you need to have called the `bedtool.sequence()` method.

A new bedtool object is returned which references the newly saved file.

Example usage:

```
a = bedtool('in.bed')

# specify the filename of the genome in fasta format
a.sequence('data/genomes/genome.fa')

# use this method to save the seqs that correspond to the features
# in "a"
a.save_seqs('seqs.fa')
```

saveas (*fn*, *trackline=None*)
Save BED file as a new file, adding the optional *trackline* to the beginning.

Returns a new bedtool for the newly saved file.

A newline is automatically added to the trackline if it does not already have one.

Example usage:

```
a = bedtool('in.bed')
b = a.random_subset(5)
b.saveas('random-5.bed',trackline='track name="random subset" color=128,128,255')
```

sequence (***kwargs*)
pybedtools help:

Wraps `fastaFromBed`. *fi* is passed in by the user; *bed* is automatically passed in as the bedfile of this object; *fo* by default is a temp file. Use save_seqs() to save as a file.

Example usage:

```
a = bedtool('in.bed')
a.sequence(fi='genome.fa')
a.print_sequence()
```

Original BEDtools program help:

Program: fastaFromBed (v2.10.0) Author: Aaron Quinlan (aaronquinlan@gmail.com) Summary: Extract DNA sequences into a fasta file based on feature coordinates.

Usage: fastaFromBed [OPTIONS] -fi <fasta> -bed <bed/gff/vcf> -fo <fasta>

Options:

-fi	Input FASTA file
-bed	BED/GFF/VCF file of ranges to extract from -fi
-fo	Output file (can be FASTA or TAB-delimited)
-name	Use the name field for the FASTA header
-tab	Write output in TAB delimited format. - Default is FASTA format.

-s	Force strandedness. If the feature occupies the antisense strand, the sequence will be reverse complemented. - By default, strand information is ignored.

sequence_coverage()
Returns the number of bases covered by this BED file. Does a self.merge() first to remove potentially multiple-counting bases.

Example usage:

```
a = bedtool('in.bed')

# total bp in genome covered by 'in.bed'
total_bp = a.sequence_coverage()
```

shuffle(*genome=None*, ***kwargs*)
pybedtools help:

Original BEDtools program help:

Program: shuffleBed (v2.10.0) Author: Aaron Quinlan (aaronquinlan@gmail.com) Summary: Randomly permute the locations of a feature file among a genome.

Usage: shuffleBed [OPTIONS] -i <bed/gff/vcf> -g <genome>

Options:

-excl	A BED/GFF/VCF file of coordinates in which features in -i should not be placed (e.g. gaps.bed).
-chrom	Keep features in -i on the same chromosome. - By default, the chrom and position are randomly chosen.
-seed	Supply an integer seed for the shuffling. - By default, the seed is chosen automatically. - (INTEGER)

Notes:

1. The genome file should tab delimited and structured as follows: <chromName><TAB><chromSize>

For example, Human (hg19): chr1 249250621 chr2 243199373 ... chr18**gl000207**random 4262

Tips: One can use the UCSC Genome Browser's MySQL database to extract chromosome sizes. For example, H. sapiens:

mysql –user=genome –host=genome-mysql.cse.ucsc.edu -A -e / "select chrom, size from hg19.chromInfo" > hg19.genome

size_filter(*min=0*, *max=10000000000000000.0*)
Returns a new bedtool object containing only those features that are > *min* and < *max*.

Example usage:

```
a = bedtool('in.bed')

# Only return features that are over 10 bp.
b = a.size_filter(min=10)
```

slop(*genome=None*, ***kwargs*)
pybedtools help:

Wraps slopBed, which adds bp to each feature. Returns a new bedtool object.

If *genome* is specified with a genome name, the genome file will be automatically retrieved from UCSC Genome Browser.

Example usage:

```
a = bedtool('in.bed')

# increase the size of features by 100 bp in either direction
b = a.slop(genome='dm3', b=100)

# grow features by 10 bp upstream and 500 bp downstream,
# using a genome file you already have constructed called
# dm3.genome.
c = a.slop(g='dm3.genome', l=10, r=500, s=True)
```

Original BEDtools program help:

Program: slopBed (v2.10.0) Author: Aaron Quinlan (aaronquinlan@gmail.com) Summary: Add requested base pairs of "slop" to each feature.

Usage: slopBed [OPTIONS] -i <bed/gff/vcf> -g <genome> [-b <int> or (-l and -r)]

Options:

-b	Increase the BED/GFF/VCF entry by -b 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. E.g. if used, -l 500 for a negative-stranded feature, it will add 500 bp downstream. Default = false.

Notes:

1. Starts will be set to 0 if options would force it below 0.

(2) Ends will be set to the chromosome length if requested slop would force it above the max chrom length. (3) The genome file should tab delimited and structured as follows:

<chromName><TAB><chromSize>

For example, Human (hg19): chr1 249250621 chr2 243199373 ... chr18**gl000207**random 4262

Tips: One can use the UCSC Genome Browser's MySQL database to extract chromosome sizes. For example, H. sapiens:

mysql –user=genome –host=genome-mysql.cse.ucsc.edu -A -e / "select chrom, size from hg19.chromInfo" > hg19.genome

sort (***kwargs*)

pybedtools help:

Original BEDtools program help:

Program: sortBed (v2.10.0) Author: Aaron Quinlan (aaronquinlan@gmail.com) Summary: Sorts a feature file in various and useful ways.

Usage: sortBed [OPTIONS] -i <bed/gff/vcf>

Options:

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

sorted(*col*, *reverse=None*)

Returns a new bedtool object, sorted by the column specified. col can be a list of columns. BED columns that are ints (start, stop and value) will be sorted numerically; other columns will be alphabetical.

reverse is a list of booleans, same length as col, specifying which fields to reverse-sort.

TODO: currently multiple columns aren't working!

a = bedtool('in.fn') b = a.sorted(col=2) # sort by start position c = a.sorted(col=5,reverse=True) # reverse sort on the values

subtract(*other*, ***kwargs*)

pybedtools help:

Subtracts from another BED file and returns a new bedtool object.

Example usage:

```
a = bedtool('in.bed')

# do a "stranded" subtraction
b = a.subtract('other.bed',s=True)

# Require 50% of features in a to overlap
c = a.subtract('other.bed', s=0.5)
```

Original BEDtools program help:

Program: subtractBed (v2.10.0) Author: Aaron Quinlan (aaronquinlan@gmail.com) Summary: Removes the portion(s) of an interval that is overlapped

by another feature(s).

Usage: subtractBed [OPTIONS] -a <bed/gff/vcf> -b <bed/gff/vcf>

Options:

-f	Minimum overlap required as a fraction of A. - Default is 1E-9 (i.e., 1bp). - (FLOAT) (e.g. 0.50)
-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.

tostring()

Returns the BED file as a string. You can also `print` the bedtool object to view its contents.

Example usage:

```
a = bedtool('in.bed')

# this is one looong string which contains the entire file
long_string = a.tostring()
```

window(*other*, ***kwargs*)

pybedtools help:

Intersect with a window.

Example usage:

```
a = bedtool('in.bed')

# Consider features up to 500 bp away as overlaps
b = a.window(w=500)
```

Original BEDtools program help:

Program: windowBed (v2.10.0) Author: Aaron Quinlan (aaronquinlan@gmail.com) Summary: Examines a "window" around each feature in A and

reports all features in B that overlap the window. For each overlap the entire entry in A and B are reported.

Usage: windowBed [OPTIONS] -a <bed/gff/vcf> -b <bed/gff/vcf>

Options:

-abam	The A input file is in BAM format. Output will be BAM as well.
-ubam	Write uncompressed BAM output. Default is to write compressed BAM.
-bed	When using BAM input (-abam), write output as BED. The default is to write output in BAM when using -abam.
-w	Base pairs added upstream and downstream of each entry in A when searching for overlaps in B. - Creates symterical "windows" around A. - Default is 1000 bp. - (INTEGER)
-l	Base pairs added upstream (left of) of each entry in A when searching for overlaps in B. - Allows one to define assymterical "windows". - Default is 1000 bp. - (INTEGER)
-r	Base pairs added downstream (right of) of each entry in A when searching for overlaps in B. - Allows one to define assymterical "windows". - Default is 1000 bp. - (INTEGER)
-sw	Define -l and -r based on strand. For example if used, -l 500 for a negative-stranded feature will add 500 bp downstream. - Default = 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 the original A entry **once** if **any** overlaps found in B. - In other words, just report the fact >=1 hit was found.
-c	For each entry in A, report the number of overlaps with B. - Reports 0 for A entries that have no overlap with B. - Overlaps restricted by -f.
-v	Only report those entries in A that have **no overlaps** with B. - Similar to "grep -v."

class pybedtools.**bedtool**

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX