pybedtools Documentation

Release 0.2.0dev

Ryan Dale

January 19, 2011

CONTENTS

1 Overview **3**

2 Overview **5**
- 2.1 Installation . . . 5
- 2.2 Three brief examples . . . 5

3 Tutorial **7**
- 3.1 Why use `pybedtools`? . . . 7
- 3.2 Limitations . . . 7
- 3.3 Creating a `bedtool` . . . 8
- 3.4 Design principles: an example . . . 9
- 3.5 Saving, printing, viewing . . . 12
- 3.6 Individual methods . . . 12
- 3.7 Example: Flanking seqs . . . 13
- 3.8 Example: Region centers that are fully intergenic . . . 13
- 3.9 Example: Histogram of feature lengths . . . 14

4 Module documentation **15**
- 4.1 `pybedtools` module-level functions . . . 15
- 4.2 `bedtool` methods . . . 15

5 Indices and tables **29**

Python Module Index **31**

Index **33**

Contents:

CHAPTER

ONE

OVERVIEW

pybedtools is a Python wrapper for Aaron Quinlan's BEDtools and is designed to leverage the "genome algebra" power of BEDtools from within Python scripts.

This documentation is written assuming you know how to use BEDTools and Python.

See full online documentation at http://daler.github.com/pybedtools.

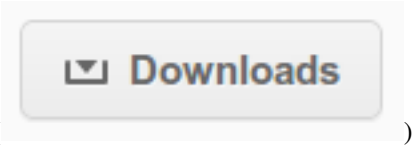
CHAPTER

TWO

OVERVIEW

2.1 Installation

To use `pybedtools` you'll need the latest version of the package and the latest version of BEDTools.

1. To install the latest version of `pybedtools`:
   - go to http://github.com/daler/pybedtools
   - click the Downloads link (

)
   - choose either a `.tar.gz` or a `.zip` file, whatever you're comfortable with
   - unzip into a temporary directory
   - from the command line, run:

     ```
     python setup.py install
     ```

     (you may need admin rights to do this)
2. To install BEDTools
   - follow the instructions at https://github.com/arq5x/bedtools to install
   - make sure all its programs are on your path

2.2 Three brief examples

Here are three examples to show typical usage of `pybedtools`. More info can be found in the docstrings of `pybedtools` methods and in the *Tutorial*

2.2.1 Example 1

Save a new BED file of intersections between `a.bed` and `b.bed`, adding a track line to the output:

```
>>> from pybedtools import bedtool
>>> a = bedtool('a.bed')
>>> a.intersect('b.bed').saveas('a-and-b.bed', trackline="track name='a and b' color=128,0,0")
```

2.2.2 Example 2

Get values for a 3-way Venn diagram of overlaps. This demonstrates operator overloading of `bedtool` objects:

```
>>> from pybedtools import bedtool
>>> # set up 3 different bedtools
>>> a = bedtool('a.bed')
>>> b = bedtool('b.bed')
>>> c = 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.
```

2.2.3 Example 3

Get the sequences of the 100 bp on either side of features.

The `bedtool.slop()` method automatically downloads the `chromSizes` table from UCSC for the dm3 genome, but you can pass your own file using the standard BEDTools `slop` argument of `g`. Note that this example assumes you have a local copy of the entire dm3 genome saved as `dm3.fa`.

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

For more, continue on to the *Tutorial*.

CHAPTER

THREE

TUTORIAL

3.1 Why use pybedtools?

I find the BEDTools command line programs indispensable for working with genomic data. Much of my analysis code is written in Python, and I found myself calling BEDTools from my Python code. However, this got quite awkward, because I would end up doing things like this just to get the number of intersecting features between two bed files::

```
>>> 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 = int(results.split()[-1])
```

To get the number of features in `a.bed` and not `b.bed` would mean another 4 lines of this. This got old quickly, hence the creation of `pybedtools`.

As a quick illustration of the streamlining possible with `pybedtools`, 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
```

For comparison, 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 -v | wc -l   # unique to a
intersectBed -a b.bed -b a.bed -v | wc -l   # unique to b
```

Behind the scenes, the `pybedtools.bedtool` class does something very similar to this – but conveniently makes the functionality available as `a+b` or `a-b`. The `bedtool.count()` method does the line counting (automatically ignoring comment lines or track lines as well).

In addition to wrapping the BEDtools programs, there are many additional `bedtool` methods provided in this module that you can use in your Python code.

3.2 Limitations

There are some limitations you need to be aware of.

- pybedtools makes heavy use of temporary files. This makes it very convenient to work with, but if you are limited by disk space, you'll have to pay attention (see principle 1 below for more info).
- Second, bedtool methods that wrap BEDTools programs will work on BAM, GFF, VCF, and everything that BEDTools supports. However, many pybedtools-specific methods (for example bedtool.lengths() or bedtool.size_filter()) currently only work on BED files. I hope to add support for all interval files soon.

3.3 Creating a bedtool

To create a bedtool, first you need to import the pybedtools module. For the rest of the tutorial, I'm assuming you have already done the following:

```
>>> import pybedtools
>>> from pybedtools import bedtool
```

Next, you need a BED file to work with. Luckily, pybedtools comes with some example bed files. You can take a look at the list of example files that ship with pybedtools with the list_example_beds() function:

```
>>> # list the example bed files
>>> pybedtools.list_example_beds()
['a.bed']
```

Once you decide on a file to use, feed the your choice to the example_bed() function to get the full path:

```
>>> # get the full path to an example bed file
>>> bedfn = pybedtools.example_bed('a.bed')
```

The full path of *bedfn* will depend on your installation (this is similar to the data() function in R, if you're familiar with that).

Now that you have a filename – either one of the example files or your own, you create a new bedtool simply by pointing it to that filename:

```
>>> # create a new bedtool from the example bed file
>>> mybedtool = bedtool(bedfn)
```

Alternatively, you can construct BED files from scratch by using the from_string keyword argument. However, all spaces will be converted to tabs using this method, so you'll have to be careful if you add "name" columns. This can be useful if you want to create *de novo* BED files on the fly:

```
>>> # an "inline" example:
>>> fromscratch1 = pybedtools.bedtool('chrX 1 100', from_string=True)
>>> print fromscratch1
chrX        1       100


>>> # using a longer string to make a bed file.  Note that
>>> # newlines don't matter, and one or more consecutive
>>> # spaces will be converted to a tab character.
>>> larger_string = """
... chrX 1    100   feature1  0 +
... chrX 50   350   feature2  0 -
... chr2 5000 10000 another_feature 0 +
... """

>>> fromscratch2 = bedtool(larger_string, from_string=True)
>>> print fromscratch2
```

```
chrX        1       100     feature1        0       +
chrX        50      350     feature2        0       -
chr2        5000    10000   another_feature 0       +
```

Of course, you'll usually be using your own bed files that have some biological importance for your work that are saved in places convenient for you, for example:

```
>>> a = bedtool('/data/sample1/peaks.bed')
```

But for the purposes of this tutorial, we'll be using two bed files that you can get from the examples directory:

```
>>> a = bedtool(pybedtools.example_bed('a.bed'))
>>> b = bedtool(pybedtools.example_bed('b.bed'))
```

3.4 Design principles: an example

3.4.1 Principle 1: temporarly files are created automatically

Let's illustrate some of the design principles behind `pybedtools` by merging features in `a.bed` that are 100 bp or less apart (*d=100*) in a strand-specific way (*s=True*):

```
>>> from pybedtools import bedtool
>>> import pybedtools
>>> a = bedtool(pybedtools.example_bed('a.bed')
>>> merged_a = a.merge(d=100, s=True)
```

Now *merged_a* is a `bedtool` instance that contains the results of the merge.

`bedtool` objects must always point to a file on disk. So in the example above, *merged_a* is a `bedtool`, but what file does it point to? You can always check the `bedtool.fn` attribute to find out:

```
>>> # what file does *merged_a* point to?
>>> merged_a.fn
'/tmp/pybedtools.MPPp5f.tmp'
```

Note that the specific filename will be different for you since it is a randomly chosen name (handled by Python's `tempfile` module). This shows one important aspect of `pybedtools`: every operation results in a new temporary file. Temporary files are stored in `/tmp` by default, and have the form `/tmp/pybedtools.*.tmp`.

Future work on `pybedtools` will focus on streamlining the temp files, keeping only those that are needed. For now, when you are done using the `pybedtools` module, make sure to clean up all the temp files created with:

```
>>> # Deletes all tempfiles created this session.
>>> # Don't do this yet if you're following the tutorial!
>>> pybedtools.cleanup()
```

If you forget to do this, from the command line you can always do a:

```
rm /tmp/pybedtools.*.tmp
```

to clean everything up.

3.4.2 Principle 2: Names and arguments are similar to BEDTools

Returning again to this example:

```
>>> merged_a = a.merge(d=100, s=True)
```

The second principle is that the bedtool.merge() method does the same thing and takes the same arguments as the BEDTools program mergeBed. In general, remove the "Bed" from the end of the BEDTools program to get the corresponding bedtool method. So there's a bedtool.subtract() method, a bedtool.intersect() method, and so on.

Note that we used the *d=100* keyword argument. Since *mergeBed -d* is an option for the BEDTools mergeBed program, it can also be used by bedtool.merge(). The same goes for any other options.

3.4.3 Principle 3: Sensible default args

When running BEDTools mergeBed program from the command line, you would have to specify the input file with the *mergeBed -i* option.

pybedtools assumes that if you're calling the merge() method on *a*, you want to operate on the bed file that *a* points to.

In general, BEDTools programs that accept a single BED file as input (typically specified with the *-i* option) the default for pybedtools is to use the bedtool's file as input.

For BEDTools programs that accept two BED files as input (like intersectBed, with the first file as *-a* and the second file as *-b*), the default for pybedtools is to consider the bedtool's file as "a" and the first non-keyword argument as "b". Furthermore, the first non-keyword argument can either be a filename *or* another bedtool object.

You can still pass a file in using the *i* keyword argument, if you want. In fact, the following two versions produce the same output:

```
>>> # The default is to use existing file for input -- no need
>>> # to specify "i" . . .
>>> result1 = a.merge(d=100, s=True)

>>> # . . . but you can always be explicit if you'd like
>>> result2 = a.merge(i=a.fn, d=100, s=True)

>>> # Confirm that the output is identical
>>> str(result1) == str(result2)
True
```

3.4.4 Principal 4: Other arguments have no defaults

-d is an option to BEDTools mergeBed that accepts a value, while *-s* is an option that acts as a switch. In pybedtools, simply pass boolean values (True or False) for the switch-type options, and pass a value for the value-type options.

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. As another example:

```
>>> a.intersect(b, v=True, f=0.5)
```

Other than the *-i*, *-a*, and *-b* options for input files mentioned above, these other options like *-d* and *s* have no defaults.

Again, any option that can be passed to a BEDTools program can be passed to the corresonding bedtool method.

3.4.5 Principle 5: Chaining together commands

Most methods return new `bedtool` objects, allowing you to chain things together just like piping commands together on the command line. To give you a flavor of this, here is how you would get the merged regions of features shared between `a.bed` (as referred to by the `bedtool` *a* we made previously) and `b.bed`: (as referred to by the `bedtool` *b*):

```
>>> a.intersect(b).merge().saveas('shared_merged.bed')
```

This is equivalent to the following BEDTools commands:

```
intersectBed -a a.bed -b b.bed | merge -i stdin > shared_merged.bed
```

3.4.6 Principle 6: Check the help

If you're unsure of whether a method uses a default, or if you want to read about what options an underlying BEDTools program accepts, check the help. Each `pybedtool` method that wraps a BEDTools program also wraps the BEDTools program help string. There are often examples of how to use a method in the docstring as well:

```
>>> # Checking the help!
>>> help (a.merge)
Help on method merge in module pybedtools.bedtool:

merge(self, **kwargs) method of pybedtools.bedtool.bedtool instance
    *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.
```

3.5 Saving, printing, viewing

Let's make two files from scratch:

```
>>> from pybedtools import bedtool
>>> string1 = '''
...           chrX 1   100
...           chrX 200 500
...           chrY 499 600
...           '''
>>> string2 = '''
...           chrX 10  60
...           chrY 200 500
...           '''
>>> x = bedtool(string1, from_string=True)
>>> y = bedtool(string2, from_string=True)
```

We can save a new file, adding a track line as well (a newline is added for you if you forget):

```
>>> y.saveas('y.bed', trackline="track name='example bed' color=135,0,85")
```

View the first 10 lines of the bed file:

```
>>> y.head()
```

Print the *whole* file:

```
>>> print y
```

3.6 Individual methods

3.6.1 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]
```

3.7 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()
```

3.8 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')
```

3.9 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()
```

CHAPTER

FOUR

MODULE DOCUMENTATION

4.1 pybedtools module-level functions

pybedtools.**data_dir**()
Returns the data directory that contains example files for tests and documentation.

pybedtools.**example_bed**(*bed*)
Return a bed file from the pybedtools examples directory. Use `list_example_beds()` to see a list of files that are included.

pybedtools.**list_example_beds**()
Returns a list of bed files in the examples dir. Choose one and pass it to `example_bed()` to get the full path to an example BED file.

Example usage:

```
>>> choices = list_example_beds()
>>> bedfn = example_bed(choices[0])
>>> mybedtool = bedtool(bedfn)
```

4.2 bedtool methods

class pybedtools.**bedtool**(*fn*, *genome=None*, *from_string=False*)
Wrapper around Aaron Quinlans `BEDtools` suite of programs (https://github.com/arq5x/bedtools); also contains many useful methods for more detailed work with BED files.

Typical usage is to point to an existing file:

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

But you can also create one from scratch from a string:

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

Note: This method returns a new bedtool instance

Note: This method accepts either a bedtool or a file name as the first unnamed argument

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)
```

Note: This method returns a new bedtool instance

Note: For convenience, the file this bedtool object points to is passed as "-a"

Note: This method accepts either a bedtool or a file name as the first unnamed argument

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)
```

Note: This method returns a new bedtool instance

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:

Note: For convenience, the file this bedtool object points to is passed as "-i"

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)
```

Note: This method returns a new bedtool instance

Note: For convenience, the file this bedtool object points to is passed as "-a"

Note: This method accepts either a bedtool or a file name as the first unnamed argument

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')

Note: This method accepts either a bedtool or a file name as the first unnamed argument

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)
```

Note: This method returns a new bedtool instance

Note: For convenience, the file this bedtool object points to is passed as "-i"

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)
```

Note: This method returns a new bedtool instance

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".

Note: This method returns a new bedtool instance

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')
```

Note: This method returns a new bedtool instance

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()
```

Note: This method returns a new bedtool instance

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:

Note: For convenience, the file this bedtool object points to is passed as "-i"

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=1000000000000000.0*)

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)
```

Note: This method returns a new bedtool instance

Note: For convenience, the file this bedtool object points to is passed as "-i"

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:

Note: For convenience, the file this bedtool object points to is passed as "-i"

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)
```

Note: This method returns a new bedtool instance

Note: This method accepts either a bedtool or a file name as the first unnamed argument

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)
```

Note: For convenience, the file this bedtool object points to is passed as "-a"

Note: This method accepts either a bedtool or a file name as the first unnamed argument

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."

with_attrs(***kwargs*)

Given arbitrary keyword arguments, turns the keys and values into attributes.

Example usage:

```
>>> a = bedtool('a.bed').with_attrs(label='transcription factor 1')
>>> b = bedtool('b.bed').with_attrs(label='transcription factor 2')
>>> for i in [a,b]:
...     print i.count(), 'features for', i.label
```

Note: This method returns a new bedtool instance

CHAPTER

FIVE

INDICES AND TABLES

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

PYTHON MODULE INDEX

p

pybedtools, 15

INDEX

B

bedtool (class in pybedtools), 15

C

cat() (pybedtools.bedtool method), 15
closest() (pybedtools.bedtool method), 16
count() (pybedtools.bedtool method), 17
counts() (pybedtools.bedtool method), 17

D

data_dir() (in module pybedtools), 15

E

example_bed() (in module pybedtools), 15

F

feature_centers() (pybedtools.bedtool method), 17
features() (pybedtools.bedtool method), 17

G

get_genome() (pybedtools.bedtool method), 17
groupBy() (pybedtools.bedtool method), 17

H

head() (pybedtools.bedtool method), 18

I

intersect() (pybedtools.bedtool method), 18
intersection_report() (pybedtools.bedtool method), 20

L

lengths() (pybedtools.bedtool method), 20
list_example_beds() (in module pybedtools), 15

M

merge() (pybedtools.bedtool method), 20

N

newshuffle() (pybedtools.bedtool method), 21
normalized_counts() (pybedtools.bedtool method), 21

P

parse_kwargs() (pybedtools.bedtool method), 21
print_randomstats() (pybedtools.bedtool method), 21
print_sequence() (pybedtools.bedtool method), 21
pybedtools (module), 15

R

random_subset() (pybedtools.bedtool method), 22
randomintersection() (pybedtools.bedtool method), 22
randomstats() (pybedtools.bedtool method), 22
rename_features() (pybedtools.bedtool method), 22

S

save_seqs() (pybedtools.bedtool method), 22
saveas() (pybedtools.bedtool method), 23
sequence() (pybedtools.bedtool method), 23
sequence_coverage() (pybedtools.bedtool method), 23
shuffle() (pybedtools.bedtool method), 24
size_filter() (pybedtools.bedtool method), 24
slop() (pybedtools.bedtool method), 24
sort() (pybedtools.bedtool method), 25
sorted() (pybedtools.bedtool method), 26
subtract() (pybedtools.bedtool method), 26

T

tostring() (pybedtools.bedtool method), 26

W

window() (pybedtools.bedtool method), 27
with_attrs() (pybedtools.bedtool method), 28