Package ‘valr’

January 3, 2019

Type Package

Title Genome Interval Arithmetic in R

Version 0.5.0

Description Read and manipulate genome intervals and signals. Provides functionality similar to command-line tool suites within R, enabling interactive analysis and visualization of genome-scale data.

License MIT + file LICENSE

Depends R (>= 3.1.2)

Imports dplyr (>= 0.7.0), rlang, readr, stringr, tibble (>= 1.4.2), broom, ggplot2

SystemRequirements C++11

LinkingTo Rcpp (>= 1.0.0),

Suggests knitr, rmarkdown, testthat, microbenchmark, covr, curl, RMySQL, purrr, tidyr, devtools, DT, cowplot, dbplyr, GenomicRanges, IRanges, S4Vectors,

VignetteBuilder knitr

RoxygenNote 6.1.1


BugReports https://github.com/rnabioco/valr/issues

Encoding UTF-8

NeedsCompilation yes

Author Jay Hesselberth [aut, cre] (<https://orcid.org/0000-0002-6299-179X>), Kent Riemondy [aut] (<https://orcid.org/0000-0003-0750-1273>)

Maintainer Jay Hesselberth <jay.hesselberth@gmail.com>

Repository CRAN

Date/Publication 2019-01-03 16:20:04 UTC
R topics documented:

as.tbl_genome ........................................ 3
as.tbl_interval ........................................ 3
bed12_to_exons ........................................ 4
bed_absdist ............................................ 5
bed_closest ............................................. 6
bed_cluster ............................................. 8
bed_complement ......................................... 9
bed_coverage ........................................... 10
bed_fisher .............................................. 12
bed_flank ............................................... 13
bed_glyph ................................................ 14
bed_intersect ........................................... 15
bed_jaccard ............................................ 17
bed_makewindows ..................................... 18
bed_map .................................................. 19
bed_merge ................................................ 21
bed_partition .......................................... 23
bed_projection ......................................... 24
bed_random ............................................. 25
bed_reldist ............................................. 26
bed_shift ............................................... 27
bed_shuffle ............................................ 29
bed_slop ................................................. 30
bed_sort .................................................. 31
bed_subtract ........................................... 32
bed_window .............................................. 34
bound_intervals ....................................... 35
create_introns ......................................... 36
create_tss ............................................... 37
create_utrs3 ........................................... 37
create_utrs5 ........................................... 38
db .......................................................... 38
flip_strands ............................................ 39
interval_spacing ....................................... 40
is.tbl_genome .......................................... 41
is.tbl_interval ......................................... 41
read_bed ................................................ 42
read_genome ........................................... 43
read_vcf ................................................ 44
tbl_genome ............................................. 45
tbl_interval ............................................ 46
valr ....................................................... 46
valr_example ........................................... 47

Index .................................................. 48
as.tbl_genome

Coerce objects to tbl_genome.

Description
This is an S3 generic. valr includes methods to coerce tbl_df and data.frame objects.

Usage

```r
as.tbl_genome(x)
```

## S3 method for class 'tbl_df'
```
as.tbl_genome(x)
```

## S3 method for class 'data.frame'
```
as.tbl_genome(x)
```

Arguments

- `x` object to convert to tbl_genome.

Value

- `tbl_genome()`

as.tbl_interval

Coerce objects to tbl_intervals.

Description
This is an S3 generic. valr includes methods to coerce tbl_df and GRanges objects.

Usage

```r
as.tbl_interval(x)
```

## S3 method for class 'tbl_df'
```
as.tbl_interval(x)
```

## S3 method for class 'data.frame'
```
as.tbl_interval(x)
```

## S3 method for class 'GRanges'
```
as.tbl_interval(x)
```
Arguments

  x object to convert to tbl_interval.

Value

  tbl_interval()

Examples

```r
## not run:
gr <- GenomicRanges::GRanges(
  seqnames = S4Vectors::Rle(
    c("chr1", "chr2", "chr1", "chr3"),
    c(1, 1, 1, 1)),
  ranges = IRanges::IRanges(
    start = c(1, 10, 50, 100),
    end = c(100, 500, 1000, 2000),
    names = head(letters, 4)),
  strand = S4Vectors::Rle(
    c("-", "+"), c(2, 2))
)

as.tbl_interval(gr)
```

# There are two ways to convert a tbl_interval to GRanges:

```r
gr <- GenomicRanges::GRanges(
  seqnames = S4Vectors::Rle(x$chrom),
  ranges = IRanges::IRanges(
    start = x$start + 1,
    end = x$end,
    names = x$name),
  strand = S4Vectors::Rle(x$strand)
)
```

# or:

```r
gr <- GenomicRanges::makeGRangesFromDataFrame(dplyr::mutate(x, start = start + 1))
```

## End(Not run)

---

**bed12_to_exons**

Convert BED12 to individual exons in BED6.

Description

After conversion to BED6 format, the score column contains the exon number, with respect to strand (i.e., the first exon for - strand genes will have larger start and end coordinates).
Usage

bed_absdist(x)

Arguments

x: tbl_interval()

See Also

Other utilities: bed_makewindows, bound_intervals, flip_strands, interval_spacing

Examples

x <- read_bed12(valr_example('mm9.refGene.bed.gz'))

bed12_to_exons(x)

bed_absdist(x, y, genome)

Arguments

x: tbl_interval()
y: tbl_interval()
genome: tbl_genome()

Details

Absolute distances are scaled by the inter-reference gap for the chromosome as follows. For Q query points and R reference points on a chromosome, scale the distance for each query point i to the closest reference point by the inter-reference gap for each chromosome. If an x interval has no matching y chromosome, .absdist is NA.

\[ d_i(x, y) = \min_k(|q_i - r_k|) \frac{R}{\text{Length of chromosome}} \]

Both absolute and scaled distances are reported as .absdist and .absdist_scaled.

Interval statistics can be used in combination with dplyr::group_by() and dplyr::do() to calculate statistics for subsets of data. See vignette('interval-stats') for examples.
**Value**

`tbl_interval()` with `.absdist` and `.absdist_scaled` columns.

**See Also**

[http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002529](http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002529)

Other interval statistics: `bed_fisher, bed_jaccard, bed_projection, bed_reldist`

**Examples**

```r
genie <- read_genome(valr_example('hg19.chrom.sizes.gz'))

x <- bed_random(genome, seed = 1010486)
y <- bed_random(genome, seed = 9203911)

bed_absdist(x, y, genome)
```

---

**Description**

Identify closest intervals.

**Usage**

```r
bed_closest(x, y, overlap = TRUE, suffix = c(".x", ".y"))
```

**Arguments**

- **x** `tbl_interval()`
- **y** `tbl_interval()`
- **overlap** report overlapping intervals
- **suffix** colname suffixes in output

**Details**

Input tbls are grouped by chrom by default, and additional groups can be added using `dplyr::group_by()`. For example, grouping by strand will constrain analyses to the same strand. To compare opposing strands across two tbls, strands on the y tbl can first be inverted using `flip_strands()`.

**Value**

`tbl_interval()` with additional columns:

- `.dist` distance to closest interval. Negative distances denote upstream intervals.
- `.overlap` overlap with closest interval
See Also

http://bedtools.readthedocs.io/en/latest/content/tools/closest.html

Other multiple set operations: bed_coverage, bed_intersect, bed_map, bed_subtract, bed_window

Examples

```r
x <- trbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 100, 125
)

y <- trbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 25, 50,
  'chr1', 140, 175
)

bed_glyph(bed_closest(x, y))

x <- trbl_interval(
  ~chrom, ~start, ~end,
  "chr1", 500, 600,
  "chr2", 5000, 6000
)

y <- trbl_interval(
  ~chrom, ~start, ~end,
  "chr1", 100, 200,
  "chr1", 150, 200,
  "chr1", 550, 580,
  "chr2", 7000, 8500
)

bed_closest(x, y)

bed_closest(x, y, overlap = FALSE)

# Report distance based on strand
x <- trbl_interval(
  ~chrom, ~start, ~end, ~name, ~score, ~strand,
  "chr1", 10, 20, "a", 1, "-"
)

y <- trbl_interval(
  ~chrom, ~start, ~end, ~name, ~score, ~strand,
  "chr1", 8, 9, "b", 1, "+",
  "chr1", 21, 22, "b", 1, "-"
)

res <- bed_closest(x, y)

# convert distance based on strand
```
res$dist_strand <- ifelse(res$strand.x == "+", res$dist, -(res$dist))
res

# report absolute distances
res$abs_dist <- abs(res$dist)
res

---

**bed_cluster**

*Cluster neighboring intervals.*

**Description**

The output .id column can be used in downstream grouping operations. Default `max_dist = 0` means that both overlapping and book-ended intervals will be clustered.

**Usage**

```r
bed_cluster(x, max_dist = 0)
```

**Arguments**

- **x**: `tbl_interval()`
- **max_dist**: maximum distance between clustered intervals.

**Details**

Input tbls are grouped by `chrom` by default, and additional groups can be added using `dplyr::group_by()`. For example, grouping by `strand` will constrain analyses to the same strand. To compare opposing strands across two tbls, strands on the y tbl can first be inverted using `flip_strands()`.

**Value**

`tbl_interval()` with .id column specifying sets of clustered intervals.

**See Also**

- [http://bedtools.readthedocs.org/en/latest/content/tools/cluster.html](http://bedtools.readthedocs.org/en/latest/content/tools/cluster.html)
- Other single set operations: bed_complement, bed_flank, bed_merge, bed_partition, bed_shift, bed_slop
Examples

```r
x <- tblr_interval(
  ~chrom, ~start, ~end,
  'chr1', 100, 200,
  'chr1', 180, 250,
  'chr1', 250, 500,
  'chr1', 501, 1000,
  'chr2', 1, 100,
  'chr2', 150, 200
)
bed_cluster(x)

# glyph illustrating clustering of overlapping and book-ended intervals
x <- tblr_interval(
  ~chrom, ~start, ~end,
  'chr1', 1, 10,
  'chr1', 5, 20,
  'chr1', 30, 40,
  'chr1', 40, 50,
  'chr1', 80, 90
)
bed_glyph(bed_cluster(x), label = '.id')
```

Description

Identify intervals in a genome not covered by a query.

Usage

```r
bed_complement(x, genome)
```

Arguments

- `x`: `tbl_interval`
- `genome`: `tbl_genome`

Value

`tbl_interval`

See Also

Other single set operations: `bed_cluster`, `bed_flank`, `bed_merge`, `bed_partition`, `bed_shift`, `bed_slop`
Examples

```r
x <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 0, 10,
  'chr1', 75, 100
)

geno <- tbl_genome(
  ~chrom, ~size,
  'chr1', 200
)

bed_glyph(bed_complement(x, geno))

geno <- tbl_genome(
  ~chrom, ~size,
  'chr1', 500,
  'chr2', 600,
  'chr3', 800
)

x <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 100, 300,
  'chr1', 200, 400,
  'chr2', 0, 100,
  'chr2', 200, 400,
  'chr3', 500, 600
)

# intervals not covered by x
bed_complement(x, geno)
```

---

**bed_coverage**  
*Compute coverage of intervals.*

**Description**

Compute coverage of intervals.

**Usage**

```r
bed_coverage(x, y, ...)
```

**Arguments**

- `x`  
  *tbl_interval()*

- `y`  
  *tbl_interval()*

- `...`  
  extra arguments (not used)
Details

input tbls are grouped by chrom by default, and additional groups can be added using `dplyr::group_by()`. For example, grouping by strand will constrain analyses to the same strand. To compare opposing strands across two tbls, strands on the y tbl can first be inverted using `flip_strands()`.

Value

tbl_interval() with the following additional columns:

- .ints number of x intersections
- .cov per-base coverage of x intervals
- .len total length of y intervals covered by x intervals
- .frac .len scaled by the number of y intervals

Note

Book-ended intervals are included in coverage calculations.

See Also

http://bedtools.readthedocs.org/en/latest/content/tools/coverage.html

Other multiple set operations: `bed_closest, bed_intersect, bed_map, bed_subtract, bed_window`

Examples

```r
x <- tbl_interval(
  ~ chrom, ~ start, ~ end, ~ strand,
  "chr1", 100, 500, '+',
  "chr2", 200, 400, '+',
  "chr2", 300, 500, '-',
  "chr2", 800, 900, '-'
)

y <- tbl_interval(
  ~ chrom, ~ start, ~ end, ~ value, ~ strand,
  "chr1", 150, 400, 100, '+',
  "chr1", 500, 550, 100, '+',
  "chr2", 230, 430, 200, '-',
  "chr2", 350, 430, 300, '-'
)

bed_coverage(x, y)
```
**Fisher’s test to measure overlap between two sets of intervals.**

**Description**

Calculate Fisher’s test on number of intervals that are shared and unique between two sets of x and y intervals.

**Usage**

```r
bed_fisher(x, y, genome)
```

**Arguments**

- `x` : `tbl_interval()`
- `y` : `tbl_interval()`
- `genome` : `tbl_genome()`

**Details**

Interval statistics can be used in combination with `dplyr::group_by()` and `dplyr::do()` to calculate statistics for subsets of data. See vignette('interval-stats') for examples.

**Value**

`tbl_interval()`

**See Also**

[http://bedtools.readthedocs.org/en/latest/content/tools/fisher.html](http://bedtools.readthedocs.org/en/latest/content/tools/fisher.html)

Other interval statistics: `bed_absdist, bed_jaccard, bed_projection, bed_reldist`

**Examples**

```r
genome <- read_genome(valr_example('hg19.chrom.sizes.gz'))
x <- bed_random(genome, seed = 1018486)
y <- bed_random(genome, seed = 9203911)
bed_fisher(x, y, genome)
```
bed_flank

Create flanking intervals from input intervals.

Description
Create flanking intervals from input intervals.

Usage
bed_flank(x, genome, both = 0, left = 0, right = 0,
fraction = FALSE, strand = FALSE, trim = FALSE, ...)

Arguments
- x  tbl_interval()
- genome tbl_genome()
- both number of bases on both sizes
- left   number of bases on left side
- right  number of bases on right side
- fraction define flanks based on fraction of interval length
- strand  define left and right based on strand
- trim  adjust coordinates for out-of-bounds intervals
- ... extra arguments (not used)

Value
tbl_interval()

See Also
http://bedtools.readthedocs.org/en/latest/content/tools/flank.html

Other single set operations: bed_cluster, bed_complement, bed_merge, bed_partition, bed_shift, bed_slop

Examples
x <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1',     25,     50,
  'chr1',    100,    125
)

genome <- tbl_genome(
  ~chrom, ~size,
  'chr1', 130
)
bed_glyph(bed_flank(x, genome, both = 20))

x <- tbl_interval(
  ~chrom, ~start, ~end, ~name, ~score, ~strand,
  'chr1', 500, 1000, '.', '.', '+',
  'chr1', 1000, 1500, '.', '.', '-',
)

geno <- tbl_genome(
  ~chrom, ~size,
  'chr1', 5000
)

bed_flank(x, genome, left = 100)

bed_flank(x, genome, right = 100)

bed_flank(x, genome, both = 100)

bed_flank(x, genome, both = 0.5, fraction = TRUE)

---

**bed_glyph**

Create example glyphs for valr functions.

**Description**

Used to illustrate the output of valr functions with small examples.

**Usage**

```r
bed_glyph(expr, label = NULL)
```

**Arguments**

- `expr`: expression to evaluate
- `label`: column name to use for label values. should be present in the result of the call.

**Value**

`ggplot2::ggplot()`

**Examples**

```r
x <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 25, 50,
  'chr1', 100, 125
```
Identify intersecting intervals.

**Description**

Report intersecting intervals from x and y tbls. Book-ended intervals have \( \text{overlap} \) values of 0 in the output.

**Usage**

```r
bed_intersect(x, ..., invert = FALSE, suffix = c(".x", ".y"))
```

**Arguments**

- **x**: `tbl_interval()`
- **...**: one or more (e.g. a list of) y `tbl_interval()`s
- **invert**: report x intervals not in y
- **suffix**: colname suffixes in output

**Details**

Input tbls are grouped by `chrom` by default, and additional groups can be added using `dplyr::group_by()`. For example, grouping by `strand` will constrain analyses to the same strand. To compare opposing strands across two tbls, strands on the y tbl can first be inverted using `flip_strands()`.
Value

tbl_interval() with original columns from x and y suffixed with \texttt{x} and \texttt{.y}, and a new \texttt{.overlap} column with the extent of overlap for the intersecting intervals.

If multiple \texttt{y} tbsls are supplied, the \texttt{.source} contains variable names associated with each interval. All original columns from the \texttt{y} are suffixed with \texttt{.y} in the output.

If \texttt{...} contains named inputs (i.e \texttt{a = y, b = z} or \texttt{list(a = y, b = z)}), then \texttt{.source} will contain supplied names (see examples).

See Also

http://bedtools.readthedocs.org/en/latest/content/tools/intersect.html

Other multiple set operations: \texttt{bed_closest, bed_coverage, bed_map, bed_subtract, bed_window}

Examples

```r
x <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 25, 50,
  'chr1', 100, 125
)

y <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 30, 75
)

bed_glyph(bed_intersect(x, y))

bed_glyph(bed_intersect(x, y, invert = TRUE))

x <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 100, 500,
  'chr2', 200, 400,
  'chr2', 300, 500,
  'chr2', 800, 900
)

y <- tbl_interval(
  ~chrom, ~start, ~end, ~value,
  'chr1', 150, 400, 100,
  'chr1', 500, 550, 100,
  'chr2', 230, 430, 200,
  'chr2', 350, 430, 300
)

bed_intersect(x, y)

bed_intersect(x, y, invert = TRUE)
```
bed_jaccard

# start and end of each overlapping interval
res <- bed_intersect(x, y)
dplyr::mutate(res, start = pmax(start.x, start.y),
              end = pmin(end.x, end.y))

z <- trbl_interval(
    ~chrom, ~start, ~end, ~value,
    'chr1', 150, 400, 100,
    'chr1', 500, 550, 100,
    'chr2', 230, 430, 200,
    'chr2', 750, 900, 400
)

bed_intersect(x, y, z)

bed_intersect(x, exons = y, introns = z)

# a list of tbl_intervals can also be passed
bed_intersect(x, list(exons = y, introns = z))

---

**bed_jaccard**  
*Calculate the Jaccard statistic for two sets of intervals.*

**Description**

Quantifies the extent of overlap between two sets of intervals in terms of base-pairs. Groups that are shared between input are used to calculate the statistic for subsets of data.

**Usage**

```r
bed_jaccard(x, y)
```

**Arguments**

- **x**: `tbl_interval`
- **y**: `tbl_interval`

**Details**

The Jaccard statistic takes values of \([0,1]\) and is measured as:

\[
J(x, y) = \frac{|x \cap y|}{|x \cup y|} = \frac{|x \cap y|}{|x| + |y| - |x \cap y|}
\]

Interval statistics can be used in combination with `dplyr::group_by()` and `dplyr::do()` to calculate statistics for subsets of data. See `vignette('interval-stats')` for examples.
Value

tibble with the following columns:
- `len_i` length of the intersection in base-pairs
- `len_u` length of the union in base-pairs
- `jaccard` value of jaccard statistic
- `n_int` number of intersecting intervals between `x` and `y`

If inputs are grouped, the return value will contain one set of values per group.

See Also

http://bedtools.readthedocs.org/en/latest/content/tools/jaccard.html

Other interval statistics: `bed_absdist`, `bed_fisher`, `bed_projection`, `bed_reldist`

Examples

g = read_genome(valr_example('hg19.chrom.sizes.gz'))
x <- bed_random(g, seed = 1010486)
y <- bed_random(g, seed = 9203911)

jaccard(x, y)

# calculate jaccard per chromosome
bed_jaccard(dplyr::group_by(x, chrom),
            dplyr::group_by(y, chrom))

```
bed_makewindows

Divide intervals into new sub-intervals ("windows").

Description

Divide intervals into new sub-intervals ("windows").

Usage

bed_makewindows(x, genome = NULL, win_size = 0, step_size = 0,
                 num_win = 0, reverse = FALSE)

Arguments

- `x` : `tbl_interval()`
- `genome` : this argument has been deprecated and is not used
- `win_size` : divide intervals into fixed-size windows
- `step_size` : size to step before next window
- `num_win` : divide intervals to fixed number of windows
- `reverse` : reverse window numbers
**Value**

tbl_interval() with .win_id column that contains a numeric identifier for the window.

**Note**

The name and .win_id columns can be used to create new interval names (see 'namenum' example below) or in subsequent group_by operations (see vignette).

**See Also**

Other utilities: bed12_to_exons, bound_intervals, flip_strands, interval_spacing

**Examples**

```r
x <- tbl_interval(
  ~chrom, ~start, ~end, ~name, ~score, ~strand,
  "chr1", 100, 200, 'A', '.', '+'
)

bed_glyph(bed_makewindows(x, num_win = 10), label = '.win_id')

# Fixed number of windows
bed_makewindows(x, num_win = 10)

# Fixed window size
bed_makewindows(x, win_size = 10)

# Fixed window size with overlaps
bed_makewindows(x, win_size = 10, step_size = 5)

# reverse win_id
bed_makewindows(x, win_size = 10, reverse = TRUE)

# bedtools 'namenum'
wins <- bed_makewindows(x, win_size = 10)
dplyr::mutate(wins, namenum = stringr::str_c(name, '_', .win_id))
```

---

**Description**

Apply functions like min() and count() to intersecting intervals. bed_map() uses bed_intersect() to identify intersecting intervals, so output columns will be suffixed with .x and .y. Expressions that refer to input columns from x and y columns must take these suffixes into account.
Usage

```r
bed_map(x, y, ..., min_overlap = 1)
```

```r
concat(.data, sep = ",")
```

```r
values_unique(.data, sep = ",")
```

```r
values(.data, sep = ",")
```

Arguments

- **x**: `tbl_interval`
- **y**: `tbl_interval`
- **...**: name-value pairs specifying column names and expressions to apply
- **min_overlap**: minimum overlap for intervals.
- **.data**: data
- **sep**: separator character

Details

Book-ended intervals can be included by setting `min_overlap = 0`. Non-intersecting intervals from `x` are included in the result with NA values.

Input tbls are grouped by `chrom` by default, and additional groups can be added using `dplyr::group_by()`. For example, grouping by `strand` will constrain analyses to the same strand. To compare opposing strands across two tbls, strands on the `y` tbl can first be inverted using `flip_strands()`.

Value

`tbl_interval()`

See Also

- [http://bedtools.readthedocs.io/en/latest/content/tools/map.html](http://bedtools.readthedocs.io/en/latest/content/tools/map.html)

Other multiple set operations: `bed_closest`, `bed_coverage`, `bed_intersect`, `bed_subtract`, `bed_window`

Examples

```r
x <- trbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 100, 250,
  'chr2', 250, 500
)

y <- trbl_interval(
  ~chrom, ~start, ~end, ~value,
  'chr1', 100, 250, 10,
  'chr1', 150, 250, 20,
)```
bed_merge

'Merge overlapping intervals.'

**Description**

Operations can be performed on merged intervals by specifying name-value pairs. Default `max_dist` of 0 means book-ended intervals are merged.

**Usage**

`bed_merge(x, max_dist = 0, ...)`

**Arguments**

- `x` 
  `tbl_interval()`
- `max_dist` 
  maximum distance between intervals to merge
- `...` 
  name-value pairs that specify operations on merged intervals
Details

input tbls are grouped by chrom by default, and additional groups can be added using `dplyr::group_by()`. For example, grouping by strand will constrain analyses to the same strand. To compare opposing strands across two tbls, strands on the y tbl can first be inverted using `flip_strands()`.

Value

tbl_interval()

See Also

http://bedtools.readthedocs.org/en/latest/content/tools/merge.html

Other single set operations: `bed_cluster, bed_complement, bed_flank, bed_partition, bed_shift, bed_slop`

Examples

```r
x <- tbl_interval(~chrom, ~start, ~end, 
  'chr1', 1, 50, 
  'chr1', 10, 75, 
  'chr1', 100, 120)

bed_glyph(bed_merge(x))

x <- tbl_interval(~chrom, ~start, ~end, ~value, ~strand, 
  "chr1", 1, 50, 1, '+', 
  "chr1", 100, 200, 2, '+', 
  "chr1", 150, 250, 3, '-', 
  "chr2", 1, 25, 4, '+', 
  "chr2", 200, 400, 5, '-', 
  "chr2", 400, 500, 6, '+', 
  "chr2", 450, 550, 7, '+')

bed_merge(x)

bed_merge(x, max_dist = 100)

# merge intervals on same strand
bed_merge(dplyr::group_by(x, strand))

bed_merge(x, .value = sum(value))
bed_partition

Partition intervals into elemental intervals

Description

Convert a set of intervals into elemental intervals that contain each start and end position in the set.

Usage

bed_partition(x, ...)

Arguments

x ~ tbl_interval()

... name-value pairs specifying column names and expressions to apply

Details

Summary operations, such as min() or count() can be performed on elemental intervals by specifying name-value pairs.

This function is useful for calculating summaries across overlapping intervals without merging the intervals.

input tbls are grouped by chrom by default, and additional groups can be added using dplyr::group_by().

For example, grouping by strand will constrain analyses to the same strand. To compare opposing strands across two tbls, strands on the y tbl can first be inverted using flip_strands().

Value

tbl_interval()

See Also

https://bedops.readthedocs.io/en/latest/content/reference/set-operations/bedops.html#partition-p-partition

Other single set operations: bed_cluster, bed_complement, bed_flank, bed_merge, bed_shift, bed_slop

Examples

x <- tbl_interval(
  ~ chrom, ~ start, ~ end, ~ value, ~ strand,
  'chr1', 100, 500, 10, "+",
  'chr1', 200, 400, 20, "-",
  'chr1', 300, 550, 30, "+",
  'chr1', 550, 575, 2, "+",
  'chr1', 800, 900, 5, "+"
)
```
bed_glyph(bed_partition(x))
bed_glyph(bed_partition(x, value = sum(value)), label = "value")

bed_partition(x)

# compute summary over each elemental interval
bed_partition(x, value = sum(value))

# partition and compute summaries based on group
x <- dplyr::group_by(x, strand)
bed_partition(x, value = sum(value))

# combine values across multiple tibbles
y <- tbl_interval(~chrom, ~start, ~end, ~value, ~strand,
  'chr1', 10, 500, 100, "+",
  'chr1', 250, 420, 200, "-",
  'chr1', 350, 550, 300, "+",
  'chr1', 550, 555, 20, "+",
  'chr1', 800, 900, 50, "+")

x <- dplyr::bind_rows(x, y)
bed_partition(x, value = sum(value))
```

---

**bed_projection**

Projection test for query interval overlap.

**Description**

Projection test for query interval overlap.

**Usage**

```
bed_projection(x, y, genome, by_chrom = FALSE)
```

**Arguments**

- `x` : `tbl_interval()`
- `y` : `tbl_interval()`
- `genome` : `tbl_genome()`
- `by_chrom` : compute test per chromosome

**Details**

Interval statistics can be used in combination with `dplyr::group_by()` and `dplyr::do()` to calculate statistics for subsets of data. See `vignette('interval-stats')` for examples.
**value**

tbl_interval() with the following columns:

- `chrom` the name of chromosome tested if by_chrom = TRUE, otherwise has a value of whole_genome
- `p.value` p-value from a binomial test. p-values > 0.5 are converted to 1 - p-value and lower_tail is FALSE
- `obs_exp_ratio` ratio of observed to expected overlap frequency
- `lower_tail` TRUE indicates the observed overlaps are in the lower tail of the distribution (e.g., less overlap than expected). FALSE indicates that the observed overlaps are in the upper tail of the distribution (e.g., more overlap than expected)

**see also**

http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002529

Other interval statistics: `bed_absdist, bed_fisher, bed_jaccard, bed_reldist`

**examples**

geno <- read_genome(valr_example('hg19.chrom.sizes.gz'))

x <- bed_random(geno, seed = 1010486)
y <- bed_random(geno, seed = 9203911)

bed_projection(x, y, geno)

bed_projection(x, y, geno, by_chrom = TRUE)

---

**bed_random**

*Generate randomly placed intervals on a genome.*

**description**

Generate randomly placed intervals on a genome.

**usage**

```r
bed_random(geno, length = 1000, n = 1e+06, sort_by = c("chrom", "start"), seed = 0)
```

**arguments**

- `genome`: tbl_genome()
- `length`: length of intervals
- `n`: number of intervals to generate
- `sort_by`: sorting variables
- `seed`: seed RNG for reproducible intervals
Details

Sorting can be suppressed with sort_by = NULL.

Value

tbl_interval()

See Also

http://bedtools.readthedocs.org/en/latest/content/tools/random.html

Other randomizing operations: bed_shuffle

Examples

genome <- tbl_genome(
  ~chrom, ~size,
  "chr1", 10000000, 
  "chr2", 50000000, 
  "chr3", 60000000, 
  "chrX", 5000000)

bed_random(genome, seed = 10104)

# sorting can be suppressed
bed_random(genome, sort_by = NULL, seed = 10104)

# 500 random intervals of length 500
bed_random(genome, length = 500, n = 500, seed = 10104)

---

bed_reldist Compute relative distances between intervals.

Description

Compute relative distances between intervals.

Usage

bed_reldist(x, y, detail = FALSE)

Arguments

x tbl_interval()
y tbl_interval()
detail report relative distances for each x interval.
Details

Interval statistics can be used in combination with `dplyr::group_by()` and `dplyr::do()` to calculate statistics for subsets of data. See vignette('interval-stats') for examples.

Value

If `detail = FALSE`, a `tbl_interval()` that summarizes calculated `.reldist` values with the following columns:

- `.reldist` relative distance metric
- `.counts` number of metric observations
- `.total` total observations
- `.freq` frequency of observation

If `detail = TRUE`, the `.reldist` column reports the relative distance for each input x interval.

See Also

http://bedtools.readthedocs.io/en/latest/content/tools/reldist.html

Other interval statistics: `bed_absdist`, `bed_fisher`, `bed_jaccard`, `bed_projection`

Examples

```r
genome <- read_genome(valr_example('hg19.chrom.sizes.gz'))
x <- bed_random(genome, seed = 1010486)
y <- bed_random(genome, seed = 9203911)

bed_reldist(x, y)

bed_reldist(x, y, detail = TRUE)
```

---

**Description**

Adjust intervals by a fixed size.

**Usage**

```r
bed_shift(x, genome, size = 0, fraction = 0, trim = FALSE)
```
Arguments

- \( x \) : \texttt{tbl_interval()}
- \( \text{genome} \) : \texttt{tbl_genome()}
- \( \text{size} \) : number of bases to shift. Positive numbers shift right, negative shift left.
- \( \text{fraction} \) : define size as a fraction of interval
- \( \text{trim} \) : adjust coordinates for out-of-bounds intervals

Value

\( \texttt{tbl_interval()} \)

See Also

- \texttt{http://bedtools.readthedocs.org/en/latest/content/tools/shift.html}
- Other single set operations: \texttt{bed_cluster, bed_complement, bed_flank, bed_merge, bed_partition, bed_slop}

Examples

```r
x <- trbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 25, 50,
  'chr1', 100, 125
)

gene <- trbl_genome(
  ~chrom, ~size,
  'chr1', 125
)

bed_glyph(bed_shift(x, gene, size = -20))

x <- trbl_interval(
  ~chrom, ~start, ~end, ~strand,
  "chr1", 100, 150, "+",
  "chr1", 200, 250, "+",
  "chr2", 300, 350, "+",
  "chr2", 400, 450, "-",
  "chr3", 500, 550, "-",
  "chr3", 600, 650, "-"
)

gene <- trbl_genome(
  ~chrom, ~size,
  "chr1", 1000,
  "chr2", 2000,
  "chr3", 3000
)

bed_shift(x, gene, 100)
```
**Description**

Shuffle input intervals.

**Usage**

```r
bed_shuffle(x, genome, incl = NULL, excl = NULL, max_tries = 1000, within = FALSE, seed = 0)
```

**Arguments**

- `x` `tbl_interval()`
- `genome` `tbl_genome()`
- `incl` `tbl_interval()` of included intervals
- `excl` `tbl_interval()` of excluded intervals
- `max_tries` maximum tries to identify a bounded interval
- `within` shuffle within chromosomes
- `seed` seed for reproducible intervals

**Value**

`tbl_interval()`

**See Also**

[http://bedtools.readthedocs.io/en/latest/content/tools/shuffle.html](http://bedtools.readthedocs.io/en/latest/content/tools/shuffle.html)

Other randomizing operations: `bed_random`
**bed_slop**

*Increase the size of input intervals.*

**Description**

Increase the size of input intervals.

**Usage**

```r
bed_slop(x, genome, both = 0, left = 0, right = 0,
fraction = FALSE, strand = FALSE, trim = FALSE, ...)
```

**Arguments**

- `x`: `tbl_interval`
- `genome`: `tbl_genome`
- `both`: number of bases on both sizes
- `left`: number of bases on left side
- `right`: number of bases on right side
- `fraction`: define flanks based on fraction of interval length
- `strand`: define left and right based on strand
- `trim`: adjust coordinates for out-of-bounds intervals
- `...`: extra arguments (not used)

**Value**

`tbl_interval`

**See Also**

- [http://bedtools.readthedocs.org/en/latest/content/tools/slop.html](http://bedtools.readthedocs.org/en/latest/content/tools/slop.html)
- Other single set operations: `bed_cluster`, `bed_complement`, `bed_flank`, `bed_merge`, `bed_partition`, `bed_shift`
Examples

```r
x <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 110, 120,
  'chr1', 225, 235
)

genome <- tbl_genome(
  ~chrom, ~size,
  'chr1', 400
)

bed_glyph(bed_slop(x, genome, both = 20, trim = TRUE))

genome <- tbl_genome(
  ~chrom, ~size,
  "chr1", 5000
)

x <- tbl_interval(
  ~chrom, ~start, ~end, ~name, ~score, ~strand,
  "chr1", 500, 1000, '.', '.', '+',
  "chr1", 1000, 1500, '.', '.', '-',
)

bed_slop(x, genome, left = 100)
bed_slop(x, genome, right = 100)
bed_slop(x, genome, both = 100)
bed_slop(x, genome, both = 0.5, fraction = TRUE)
```

---

**bed_sort**

*Sort a set of intervals.*

**Description**

Sort a set of intervals.

**Usage**

```r
bed_sort(x, by_size = FALSE, by_chrom = FALSE, reverse = FALSE)
```

**Arguments**

- `x` : `tbl_interval()`
- `by_size` : sort by interval size
by_chrom      sort within chromosome
reverse       reverse sort order

See Also

http://bedtools.readthedocs.org/en/latest/content/tools/sort.html

Examples

x <- tbl_interval(
  ~chrom, ~start, ~end,
  "chr8", 500, 1000,
  "chr8", 1000, 5000,
  "chr8", 100, 200,
  "chr1", 100, 300,
  "chr1", 100, 200
)

# sort by chrom and start
bed_sort(x)

# reverse sort order
bed_sort(x, reverse = TRUE)

# sort by interval size
bed_sort(x, by_size = TRUE)

# sort by decreasing interval size
bed_sort(x, by_size = TRUE, reverse = TRUE)

# sort by interval size within chrom
bed_sort(x, by_size = TRUE, by_chrom = TRUE)

---

bed_subtract

Subtract two sets of intervals.

Description

Subtract y intervals from x intervals.

Usage

bed_subtract(x, y, any = FALSE)

Arguments

x        tbl_interval()
y        tbl_interval()
any      remove any x intervals that overlap y
**Details**

input tbls are grouped by chrom by default, and additional groups can be added using `dplyr::group_by()`. For example, grouping by strand will constrain analyses to the same strand. To compare opposing strands across two tbls, strands on the y tbl can first be inverted using `flip_strands()`.

**See Also**


Other multiple set operations: `bed_closest, bed_coverage, bed_intersect, bed_map, bed_window`

**Examples**

```r
x <- tbl_interval(~chrom, ~start, ~end, 'chr1', 1, 100)

y <- tbl_interval(~chrom, ~start, ~end, 'chr1', 50, 75)

bed_glyph(bed_subtract(x, y))

x <- tbl_interval(~chrom, ~start, ~end, 'chr1', 100, 200, 'chr1', 250, 400, 'chr1', 500, 600, 'chr1', 1000, 1200, 'chr1', 1300, 1500)

y <- tbl_interval(~chrom, ~start, ~end, 'chr1', 150, 175, 'chr1', 510, 525, 'chr1', 550, 575, 'chr1', 900, 1050, 'chr1', 1150, 1250, 'chr1', 1299, 1501)

bed_subtract(x, y)

bed_subtract(x, y, any = TRUE)
```
bed_window

Identify intervals within a specified distance.

Description

Identify intervals within a specified distance.

Usage

bed_window(x, y, genome, ...)

Arguments

x tbl_interval()
y tbl_interval()
genome tbl_genome()
... params for bed_slop and bed_intersect

Details

Input tbls are grouped by chrom by default, and additional groups can be added using dplyr::group_by(). For example, grouping by strand will constrain analyses to the same strand. To compare opposing strands across two tbls, strands on the y tbl can first be inverted using flip_strands().

See Also

http://bedtools.readthedocs.org/en/latest/content/tools/window.html
Other multiple set operations: bed_closest, bed_coverage, bed_intersect, bed_map, bed_subtract

Examples

x <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 50,
  'chr1', 125)

y <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 75)

genome <- tbl_genome(
  ~chrom, ~size,
  'chr1', 125)

bed_glyph(bed_window(x, y, genome, both = 15))
bound_intervals

Select intervals bounded by a genome.

Description

Used to remove out-of-bounds intervals, or trim interval coordinates using a genome.

Usage

bound_intervals(x, genome, trim = FALSE)

Arguments

x tbl_interval()

 genome tbl Genome()

 trim adjust coordinates for out-of-bounds intervals

Value

tbl_interval()

See Also

Other utilities: bed12_to_exons, bed_makewindows, flip_strands, interval_spacing
create_introns

Create intron features.

Description

Numbers in the score column are intron numbers from 5’ to 3’ independent of strand. I.e., the first introns for + and - strand genes both have score values of 1.

Usage

create_introns(x)

Arguments

x tbl_interval() in BED12 format

See Also

Other feature functions: create_tss, create_utrs3, create_utrs5

Examples

x <- read_bed12(valr_example('mm9.refGene.bed.gz'))

create_introns(x)
**create_tss**

Create transcription start site features.

**Description**

Create transcription start site features.

**Usage**

```r
create_tss(x)
```

**Arguments**

- `x` `tbl_interval()` in BED format

**See Also**

Other feature functions: `create_introns, create_utrs3, create_utrs5`

**Examples**

```r
x <- read_bed12(valr_example('mm9.refGene.bed.gz'))
create_tss(x)
```

---

**create_utrs3**

Create 3’ UTR features.

**Description**

Create 3’ UTR features.

**Usage**

```r
create_utrs3(x)
```

**Arguments**

- `x` `tbl_interval()` in BED12 format

**See Also**

Other feature functions: `create_introns, create_tss, create_utrs5`
Examples

x <- read_bed12(valr_example('mm9.refGene.bed.gz'))

create_utrs3(x)

create_utrs5  
**Create 5' UTR features.**

Description

Create 5' UTR features.

Usage

create_utrs5(x)

Arguments

x  
`tbl_interval()` in BED12 format

See Also

Other feature functions: `create_introns, create_tss, create_utrs3`

Examples

x <- read_bed12(valr_example('mm9.refGene.bed.gz'))

create_utrs5(x)

db  
**Fetch data from remote databases.**

Description

Currently db_ucsc and db_ensembl are available for connections.

Usage

db_ucsc(dbname, host = "genome-mysql.cse.ucsc.edu", user = "genomep", 
        password = "password", port = 3306, ...)

db_ensembl(dbname, host = "ensemblldb.ensembl.org", user = "anonymous", 
           password = "", port = 3306, ...)
### flip_strands

Flip strands in intervals.

####Arguments

- `dbname`: name of database
- `host`: hostname
- `user`: username
- `password`: password
- `port`: MySQL connection port
- `...`: params for connection

####See Also

- [https://genome.ucsc.edu/goldenpath/help/mysql.html](https://genome.ucsc.edu/goldenpath/help/mysql.html)
- [http://www.ensembl.org/info/data/mysql.html](http://www.ensembl.org/info/data/mysql.html)

####Examples

```r
## Not run:
if(require(RMySQL)) {
  ucsc <- db_ucsc('hg38')

  # fetch the `refGene` tbl
  tbl(ucsc, "refGene")

  # the `chromInfo` tbls have size information
  tbl(ucsc, "chromInfo")
}

## End(Not run)

## Not run:
if(require(RMySQL)) {
  # squirrel genome
  ensembl <- db_ensembl('spermophilus_tridecemlineatus_core_67.2')

  tbl(ensembl, "gene")
}

## End(Not run)
```

####Description

Flips positive (+) stranded intervals to negative (−) strands, and vice-versa. Facilitates comparisons among intervals on opposing strands.
Usage

flip_strands(x)

Arguments

x tbl_interval()

See Also

Other utilities: bed12_to_exons, bed_makewindows, bound_intervals, interval_spacing

Examples

x <- tbl_interval(
  ~chrom, ~start, ~end, ~strand,
  'chr1', 1, 100, '+',
  'chr2', 1, 100, '-'
)

flip_strands(x)

interval_spacing Calculate interval spacing.

Description

Spacing for the first interval of each chromosome is undefined (NA). The leading interval of an overlapping interval pair has a negative value.

Usage

interval_spacing(x)

Arguments

x tbl_interval()

Value

tbl_interval() with .spacing column.

See Also

Other utilities: bed12_to_exons, bed_makewindows, bound_intervals, flip_strands
is.tbl_genome

Examples

```r
x <- tbl_interval(
  ~chrom, ~start, ~end,
  'chr1', 1, 100,
  'chr1', 150, 200,
  'chr2', 200, 300
)

interval_spacing(x)
```

is.tbl_genome Test if the object is a tbl_genome.

Description

Test if the object is a tbl_genome.

Usage

```r
is.tbl_genome(x)
```

Arguments

- `x` An object

Value

TRUE if the object inherits from the tbl_genome() class.

is.tbl_interval Test if the object is a tbl_interval.

Description

Test if the object is a tbl_interval.

Usage

```r
is.tbl_interval(x)
```

Arguments

- `x` An object

Value

TRUE if the object inherits from the tbl_interval() class.
read_bed

Description

read functions for BED and related formats. Filenames can be local file or URLs. The read functions load data into tbls with consistent chrom, start and end colnames.

Usage

read_bed(filename, n_fields = 3, col_types = bed12_coltypes, sort = TRUE, ...)

read_bed12(filename, ...)

read_bedgraph(filename, ...)

read_narrowpeak(filename, ...)

read_broadpeak(filename, ...)

Arguments

filename file or URL

n_fields number fields in the BED file

col_types column type spec for readr::read_tsv()

sort sort the tbl by chrom and start

... options to pass to readr::read_tsv()

Details

https://genome.ucsc.edu/FAQ/FAQformat.html#format1
https://genome.ucsc.edu/FAQ/FAQformat.html#format1
https://genome.ucsc.edu/goldenPath/help/bedgraph.html
https://genome.ucsc.edu/FAQ/FAQformat.html#format12
https://genome.ucsc.edu/FAQ/FAQformat.html#format13

Value

tbl_interval()

See Also

Other read functions: read_genome, read_vcf
**read_genome**

Read genome files.

**Description**

Genome files (UCSC "chromSize" files) contain chromosome name and size information. These sizes are used by downstream functions to identify computed intervals that have coordinates outside of the genome bounds.

**Usage**

read_genome(path)

**Arguments**

- **path**: containing chrom/contig names and sizes, one-pair-per-line, tab-delimited

**Value**

- **tbl_genome()**, sorted by size

**Note**

URLs to genome files can also be used.
read_vcf

See Also

Other read functions: read_bed, read_vcf

Examples

```r
read_genome(valr_example('hg19.chrom.sizes.gz'))

## not run:
# `read_genome` accepts a URL
read_genome('https://genome.ucsc.edu/goldenpath/help/hg19.chrom.sizes')

## end(not run)
```

---

read_vcf

Read a VCF file.

Description

Read a VCF file.

Usage

```r
read_vcf(vcf)
```

Arguments

- `vcf` vcf filename

Value

data_frame

Note

return value has chrom, start and end columns. Interval lengths are the size of the 'REF' field.

See Also

Other read functions: read_bed, read_genome

Examples

```r
vcf_file <- valr_example('test.vcf.gz')
read_vcf(vcf_file)
```
tbl_genome

Description

Equivalent to information in UCSC "chromSizes" files. Required column names are: chrom and size

Usage

```r
tbl_genome(x, ..., .validate = TRUE)
```

Arguments

- `x` A data_frame
- `...` params for `tibble::tibble()`
- `.validate` check valid column names

Value

`tbl_genome()`

Examples

```r
genome <- tibble::tribble(
  ~chrom, ~size,
  'chr1', 1e6,
  'chr2', 1e7
)

is.tbl_genome(genome)
genome <- tbl_genome(genome)
is.tbl_genome(genome)

trbl_genome(
  ~chrom, ~size,
  'chr1', 1e6
)
```
tbl_interval Tibble for intervals.

Description

Required column names are chrom, start and end.

Usage

tbl_interval(x, ..., .validate = TRUE)

Arguments

x A data_frame
... params for tibble::tibble()
.validate check valid column names

Value

tbl_interval()

Examples

x <- tibble::tribble(
  ~chrom, ~start, ~end,
  'chr1', 1,  50,
  'chr1', 10,  75,
  'chr1', 100, 120
)

is.tbl_interval(x)

x <- tbl_interval(x)
is.tbl_interval(x)
Details
To learn more about valr, start with the vignette: `browseVignettes(package = "valr")`

Author(s)
Jay Hesselberth jay.hesselberth@gmail.com
Kent Riemondy kent.riemondy@gmail.com

See Also
Report bugs at https://github.com/rnabioco/valr/issues

valr_example

Provide working directory for valr example files.

Description
Provide working directory for valr example files.

Usage
valr_example(path)

Arguments
  path    path to file

Examples
valr_example('hg19.chrom.sizes.gz')
Index

as.tbl_genome, 3
as.tbl_interval, 3
bed12_to_exons, 4, 19, 35, 40
bed_absdist, 5, 12, 18, 25, 27
bed_closest, 6, 11, 16, 20, 33, 34
bed_cluster, 8, 9, 13, 22, 23, 28, 30
bed_complement, 8, 9, 13, 22, 23, 28, 30
bed_coverage, 7, 10, 16, 20, 33, 34
bed_flank, 8, 9, 13, 22, 23, 28, 30
bed_glymph, 14
bed_intersect, 7, 11, 15, 20, 33, 34
bed_intersect, 19
bed_jaccard, 6, 12, 17, 25, 27
bed_make_windows, 5, 18, 35, 40
bed_map, 7, 11, 16, 19, 33, 34
bed_map, 19
bed_merge, 5, 12, 19, 35, 38
bed_partition, 5, 12, 19, 35, 38
bed_projection, 6, 12, 18, 24, 27
bed_random, 25, 29
bed_reldist, 6, 12, 18, 25, 26
bed_shift, 8, 9, 13, 22, 23, 27, 30
bed_shuffle, 26, 29
bed_slop, 8, 9, 13, 22, 23, 28, 30
bed_sort, 31
bed_subtract, 7, 11, 16, 20, 32, 34
bed_window, 7, 11, 16, 20, 33, 34
bound_intervals, 5, 19, 35, 40
concat (bed_map), 19
count(), 19, 23
create_introns, 36, 37, 38
create_tss, 36, 37, 38
create_utrs3, 36, 37, 38
create_utrs5, 36, 37, 38
db, 38
db_ensembl (db), 38
db_ucsc (db), 38
dplyr::do(), 5, 12, 17, 24, 27
dplyr::group_by(), 5, 6, 8, 11, 12, 15, 17, 20, 22–24, 27, 33, 34
flip_strands, 5, 19, 35, 39, 40
flip_strands(), 6, 8, 11, 15, 20, 22, 23, 33, 34
ggplot2::ggplot(), 14
interval_spacing, 5, 19, 35, 40, 40
is.tbl_genome, 41
is.tbl_interval, 41
min(), 19, 23
read_bed, 42, 44
read_bed12 (read_bed), 42
read_bedgraph (read_bed), 42
read_broadpeak (read_bed), 42
read_genome, 42, 43, 44
read_narrowpeak (read_bed), 42
read_vcf, 42, 44, 44
readr::read_tsv(), 42
tbl_genome, 45
tbl_genome(), 3, 5, 9, 12, 13, 24, 25, 28–30, 34, 35, 41, 43, 45
tbl_interval, 46
tbl_interval(), 4–6, 8–13, 15–32, 34–38, 40–42, 46
tibble::tibble(), 45, 46
tbl_genome (tbl_genome), 45
tbl_interval (tbl_interval), 46
tbl_interval(), 23
valr, 46
valr-package (valr), 46
valr_example, 47
values (bed_map), 19
values_unique (bed_map), 19