

Package ‘valr’

June 22, 2025

Type Package

Title Genome Interval Arithmetic

Version 0.8.4

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. Riemony et al. (2017) <[doi:10.12688/f1000research.11997.1](https://doi.org/10.12688/f1000research.11997.1)>.

License MIT + file LICENSE

URL <https://github.com/rnabioco/valr/>,
<https://rnabioco.github.io/valr/>

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

Depends R (>= 4.0.0)

Imports broom, cli, cpp11bigwig, dplyr (>= 0.8.0), ggplot2, lifecycle, Rcpp (>= 1.0.0), readr, rlang, stringr, tibble (>= 1.4.2)

Suggests bench, covr, cowplot, curl, DBI, dbplyr, devtools, DT, GenomicRanges, IRanges, knitr, purrr, RMariaDB, rmarkdown, S4Vectors, testthat (>= 3.0.0), vdiffr (>= 1.0.0), tidyr

LinkingTo Rcpp (>= 1.0.0)

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.2

Config/Needs/website rnabioco/rbitemplate

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation yes

Author Jay Hesselberth [aut] (ORCID: <<https://orcid.org/0000-0002-6299-179X>>),
Kent Riemony [aut, cre] (ORCID:
<<https://orcid.org/0000-0003-0750-1273>>),
RNA Bioscience Initiative [fnd, cph] (ROR: <<https://ror.org/03wmf1y16>>)

Maintainer Kent Riemony <kent.riemony@gmail.com>

Repository CRAN

Date/Publication 2025-06-22 18:30:02 UTC

Contents

bed12_to_exons	3
bed_absdist	3
bed_closest	4
bed_cluster	6
bed_complement	7
bed_coverage	9
bed_fisher	10
bed_flank	11
bed_genomecov	12
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	26
bed_reldist	27
bed_shift	28
bed_shuffle	29
bed_slop	31
bed_sort	32
bed_subtract	33
bed_window	35
bound_intervals	36
create_introns	37
create_tss	38
create_utrs3	38
create_utrs5	39
db	39
flip_strands	41
gr_to_bed	42
interval_spacing	43
ivl_df	44
read_bed	45
read_bigwig	46
read_genome	47
read_gtf	48
read_vcf	48
valr	49

bed12_to_exons

3

valr_example

50

Index

51

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

```
bed12_to_exons(x)
```

Arguments

x [ivl_df](#)

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	Compute absolute distances between intervals.
-------------	---

Description

Computes the absolute distance between the midpoint of each x interval and the midpoints of each closest y interval.

Usage

```
bed_absdist(x, y, genome)
```

Arguments

x [ivl_df](#)
y [ivl_df](#)
genome [genome_df](#)

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

`ivl_df` with .absdist and .absdist_scaled columns.

See Also

<https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002529>

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

Examples

```
genome <- 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)
```

bed_closest	<i>Identify closest intervals.</i>
-------------	------------------------------------

Description

Identify closest intervals.

Usage

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

Arguments

x	<code>ivl_df</code>
y	<code>ivl_df</code>
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

`ivl_df` with additional columns:

- `.overlap` amount of overlap with overlapping interval. Non-overlapping or adjacent intervals have an overlap of 0. `.overlap` will not be included in the output if `overlap = FALSE`.
- `.dist` distance to closest interval. Negative distances denote upstream intervals. Book-ended intervals have a distance of 1.

Note

For each interval in x `bed_closest()` returns overlapping intervals from y and the closest non-intersecting y interval. Setting `overlap = FALSE` will report the closest non-intersecting y intervals, ignoring any overlapping y intervals.

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/closest.html>

Other multiple set operations: `bed_coverage()`, `bed_intersect()`, `bed_map()`, `bed_subtract()`, `bed_window()`

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 100,    125
)

y <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 25,     50,
  "chr1", 140,    175
)

bed_glyph(bed_closest(x, y))

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

y <- tibble::tribble(
  ~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 <- tibble::tribble(
    ~chrom, ~start, ~end, ~name, ~score, ~strand,
    "chr1", 10, 20, "a", 1, "-"
  )

  y <- tibble::tribble(
    ~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

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

Arguments

<code>x</code>	ivl_df
<code>max_dist</code>	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

`ivl_df` with `.id` column specifying sets of clustered intervals.

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/cluster.html>

Other single set operations: `bed_complement()`, `bed_flank()`, `bed_genomecov()`, `bed_merge()`, `bed_partition()`, `bed_shift()`, `bed_slop()`

Examples

```
x <- tibble::tribble(
  ~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 <- tibble::tribble(
  ~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")
```

bed_complement	<i>Identify intervals in a genome not covered by a query.</i>
----------------	---

Description

Identify intervals in a genome not covered by a query.

Usage

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

Arguments

x	ivl_df
genome	ivl_df

Value

[ivl_df](#)

See Also

Other single set operations: [bed_cluster\(\)](#), [bed_flank\(\)](#), [bed_genomecov\(\)](#), [bed_merge\(\)](#), [bed_partition\(\)](#), [bed_shift\(\)](#), [bed_slop\(\)](#)

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 0,      10,
  "chr1", 75,     100
)

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 200
)

bed_glyph(bed_complement(x, genome))

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 500,
  "chr2", 600,
  "chr3", 800
)

x <- tibble::tribble(
  ~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, genome)
```

bed_coverage	<i>Compute coverage of intervals.</i>
--------------	---------------------------------------

Description

Compute coverage of intervals.

Usage

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

Arguments

x	ivl_df
y	ivl_df
...	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

[ivl_df](#) 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

<https://bedtools.readthedocs.io/en/latest/content/tools/coverage.html>

Other multiple set operations: [bed_closest\(\)](#), [bed_intersect\(\)](#), [bed_map\(\)](#), [bed_subtract\(\)](#), [bed_window\(\)](#)

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end, ~strand,
  "chr1", 100, 500, "+",
  "chr2", 200, 400, "+",
  "chr2", 300, 500, "-",
  "chr2", 800, 900, "-"
)

y <- tibble::tribble(
  ~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)
```

bed_fisher

*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

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

Arguments

x	ivl_df
y	ivl_df
genome	genome_df

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

[ivl_df](#)

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/fisher.html>

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

Examples

```
genome <- read_genome(valr_example("hg19.chrom.sizes.gz"))

x <- bed_random(genome, n = 1e4, seed = 1010486)
y <- bed_random(genome, n = 1e4, 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	ivl_df
genome	genome_df
both	number of bases on both sides
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

ivl_df

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/flank.html>

Other single set operations: `bed_cluster()`, `bed_complement()`, `bed_genomecov()`, `bed_merge()`, `bed_partition()`, `bed_shift()`, `bed_slop()`

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 25, 50,
  "chr1", 100, 125
)

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 130
)

bed_glyph(bed_flank(x, genome, both = 20))

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

genome <- tibble::tribble(
  ~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_genomecov

Calculate coverage across a genome

Description

This function is useful for calculating interval coverage across an entire genome.

Usage

```
bed_genomecov(x, genome, zero_depth = FALSE)
```

Arguments

x	ivl_df
genome	genome_df
zero_depth	If TRUE, report intervals with zero depth. Zero depth intervals will be reported with respect to groups.

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

[ivl_df](#) with the an additional column:

- .depth depth of interval coverage

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/genomecov.html>

Other single set operations: [bed_cluster\(\)](#), [bed_complement\(\)](#), [bed_flank\(\)](#), [bed_merge\(\)](#), [bed_partition\(\)](#), [bed_shift\(\)](#), [bed_slop\(\)](#)

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end, ~strand,
  "chr1", 20, 70, "+",
  "chr1", 50, 100, "-",
  "chr1", 200, 250, "+",
  "chr1", 220, 250, "+"
)

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 500,
  "chr2", 1000
)

bed_genomecov(x, genome)

bed_genomecov(dplyr::group_by(x, strand), genome)

bed_genomecov(dplyr::group_by(x, strand), genome, zero_depth = TRUE)
```

bed_glyph*Create example glyphs for valr functions.*

Description

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

Usage

```
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

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 25,    50,
  "chr1", 100,   125
)

y <- tibble::tribble(
  ~chrom, ~start, ~end, ~value,
  "chr1", 30, 75, 50
)

bed_glyph(bed_intersect(x, y))

x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 30,    75,
  "chr1", 50,    90,
  "chr1", 91,    120
)

bed_glyph(bed_merge(x))

bed_glyph(bed_cluster(x), label = ".id")
```

bed_intersect	<i>Identify intersecting intervals.</i>
---------------	---

Description

Report intersecting intervals from x and y tbls. Book-ended intervals have .overlap values of 0 in the output.

Usage

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

Arguments

x	ivl_df
...	one or more (e.g. a list of) y ivl_df() 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

[ivl_df](#) with original columns from x and y suffixed with .x and .y, and a new .overlap column with the extent of overlap for the intersecting intervals.

If multiple y tbls are supplied, the .source contains variable names associated with each interval. All original columns from the y are suffixed with .y in the output.

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

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/intersect.html>

Other multiple set operations: [bed_closest\(\)](#), [bed_coverage\(\)](#), [bed_map\(\)](#), [bed_subtract\(\)](#), [bed_window\(\)](#)

Examples

```

x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 25, 50,
  "chr1", 100, 125
)

y <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 30, 75
)

bed_glyph(bed_intersect(x, y))

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

x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 100, 500,
  "chr2", 200, 400,
  "chr2", 300, 500,
  "chr2", 800, 900
)

y <- tibble::tribble(
  ~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)

# 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 <- tibble::tribble(
  ~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

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

Arguments

x ivl_df
y ivl_df

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

<https://bedtools.readthedocs.io/en/latest/content/tools/jaccard.html>

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

Examples

```
genome <- read_genome(valr_example("hg19.chrom.sizes.gz"))

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

bed_jaccard(x, y)

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

bed_makewindows	<i>Divide intervals into new sub-intervals ("windows").</i>
-----------------	---

Description

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

Usage

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

Arguments

x	ivl_df
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

[ivl_df](#) 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

```
x <- tibble::tribble(
  ~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))
```

bed_map

*Calculate summaries from overlapping intervals.***Description**

Apply functions like `min()` and `max()` 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

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

concat(.data, sep = ",")

values_unique(.data, sep = ",")

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

Arguments

```
x          ivl_df
y          ivl_df
```

...	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

`ivl_df`

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/map.html>

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

Examples

```
x <- tibble::tribble(
  ~chrom,
  ~start,
  ~end,
  'chr1',
  100,
  250,
  'chr2',
  250,
  500
)
```

```
y <- tibble::tribble(
  ~chrom,
  ~start,
  ~end,
  ~value,
  'chr1',
  100,
  250,
  10,
  'chr1',
  150,
  250,
  20,
```

```

    'chr2',
    250,
    500,
    500
  )

  bed_glyph(bed_map(x, y, value = sum(value)), label = 'value')

  # summary examples
  bed_map(x, y, .sum = sum(value))

  bed_map(x, y, .min = min(value), .max = max(value))

  # identify non-intersecting intervals to include in the result
  res <- bed_map(x, y, .sum = sum(value))
  x_not <- bed_intersect(x, y, invert = TRUE)
  dplyr::bind_rows(res, x_not)

  # create a list-column
  bed_map(x, y, .values = list(value))

  # use `nth` family from dplyr
  bed_map(x, y, .first = dplyr::first(value))

  bed_map(x, y, .absmax = abs(max(value)))

  bed_map(x, y, .count = length(value))

  bed_map(x, y, .vals = values(value))

  # count defaults are NA not 0; differs from bedtools2 ...
  bed_map(x, y, .counts = dplyr::n())

  # ... but NA counts can be converted to 0's
  dplyr::mutate(
    bed_map(x, y, .counts = dplyr::n()),
    .counts = ifelse(is.na(.counts), 0, .counts)
  )

```

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	ivl_df
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

[ivl_df](#)

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/merge.html>

Other single set operations: [bed_cluster\(\)](#), [bed_complement\(\)](#), [bed_flank\(\)](#), [bed_genomecov\(\)](#), [bed_partition\(\)](#), [bed_shift\(\)](#), [bed_slop\(\)](#)

Examples

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

bed_glyph(bed_merge(x))

x <- tibble::tribble(
  ~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	<i>Partition intervals into elemental intervals</i>
---------------	---

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

Details

Summary operations, such as [min\(\)](#) or [max\(\)](#) 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

[ivl_df\(\)](#)

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_genomecov\(\)](#), [bed_merge\(\)](#), [bed_shift\(\)](#), [bed_slop\(\)](#)

Examples

```

x <- tibble::tribble(
  ~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 <- tibble::tribble(
  ~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	ivl_df
y	ivl_df
genome	genome_df
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

`ivl_df` 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

<https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002529>

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

Examples

```
genome <- read_genome(valr_example("hg19.chrom.sizes.gz"))

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

bed_projection(x, y, genome)

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

bed_random	<i>Generate randomly placed intervals on a genome.</i>
------------	--

Description

Generate randomly placed intervals on a genome.

Usage

```
bed_random(genome, length = 1000, n = 1e+06, seed = 0, sorted = TRUE)
```

Arguments

genome	genome_df
length	length of intervals
n	number of intervals to generate
seed	seed RNG for reproducible intervals
sorted	return sorted output

Details

Sorting can be suppressed with `sorted = FALSE`.

Value

[ivl_df](#)

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/random.html>

Other randomizing operations: [bed_shuffle\(\)](#)

Examples

```
genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 10000000,
  "chr2", 50000000,
  "chr3", 60000000,
  "chrX", 5000000
)

bed_random(genome, seed = 10104)

# sorting can be suppressed
bed_random(genome, sorted = FALSE, seed = 10104)
```

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

bed_reldist	<i>Compute relative distances between intervals.</i>
-------------	--

Description

Compute relative distances between intervals.

Usage

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

Arguments

x	ivl_df
y	ivl_df
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 [ivl_df](#) 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

<https://bedtools.readthedocs.io/en/latest/content/tools/reldist.html>

Other interval statistics: [bed_absdist\(\)](#), [bed_fisher\(\)](#), [bed_jaccard\(\)](#), [bed_projection\(\)](#)

Examples

```
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)
```

bed_shift	<i>Adjust intervals by a fixed size.</i>
-----------	--

Description

Out-of-bounds intervals are removed by default.

Usage

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

Arguments

x	ivl_df
genome	ivl_df
size	number of bases to shift. positive numbers shift right, negative shift left.
fraction	define size as a fraction of interval
trim	adjust coordinates for out-of-bounds intervals

Value

[ivl_df](#)

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/shift.html>

Other single set operations: [bed_cluster\(\)](#), [bed_complement\(\)](#), [bed_flank\(\)](#), [bed_genomecov\(\)](#), [bed_merge\(\)](#), [bed_partition\(\)](#), [bed_slop\(\)](#)

Examples

```

x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 25, 50,
  "chr1", 100, 125
)

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 125
)

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

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

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 1000,
  "chr2", 2000,
  "chr3", 3000
)

bed_shift(x, genome, 100)

bed_shift(x, genome, fraction = 0.5)

# shift with respect to strand
stranded <- dplyr::group_by(x, strand)
bed_shift(stranded, genome, 100)

```

bed_shuffle

Shuffle input intervals.

Description

Shuffle input intervals.

Usage

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

Arguments

x	ivl_df
genome	genome_df
incl	ivl_df of included intervals
excl	ivl_df of excluded intervals
max_tries	maximum tries to identify a bounded interval
within	shuffle within chromosomes
seed	seed for reproducible intervals

Value

[ivl_df](#)

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/shuffle.html>

Other randomizing operations: [bed_random\(\)](#)

Examples

```
genome <- tibble::tribble(  
  ~chrom, ~size,  
  "chr1", 1e6,  
  "chr2", 2e6,  
  "chr3", 4e6  
)  
  
x <- bed_random(genome, seed = 1010486)  
  
bed_shuffle(x, genome, seed = 9830491)
```

bed_slop	<i>Increase the size of input intervals.</i>
----------	--

Description

Increase the size of input intervals.

Usage

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

Arguments

x	ivl_df
genome	genome_df
both	number of bases on both sides
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

[ivl_df](#)

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/slop.html>

Other single set operations: [bed_cluster\(\)](#), [bed_complement\(\)](#), [bed_flank\(\)](#), [bed_genomecov\(\)](#), [bed_merge\(\)](#), [bed_partition\(\)](#), [bed_shift\(\)](#)

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 110,    120,
  "chr1", 225,    235
)

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 400
)

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

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 5000
)

x <- tibble::tribble(
  ~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	<i>Sort a set of intervals.</i>
----------	---------------------------------

Description

Sort a set of intervals.

Usage

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

Arguments

x	ivl_df
by_size	sort by interval size

by_chrom	sort within chromosome
reverse	reverse sort order

See Also

<https://bedtools.readthedocs.io/en/latest/content/tools/sort.html>

Examples

```
x <- tibble::tribble(
  ~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	<i>Subtract two sets of intervals.</i>
--------------	--

Description

Subtract y intervals from x intervals.

Usage

```
bed_subtract(x, y, any = FALSE)
```

Arguments

x	ivl_df
y	ivl_df
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

<https://bedtools.readthedocs.io/en/latest/content/tools/subtract.html>

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

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 1,      100
)

y <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 50,     75
)

bed_glyph(bed_subtract(x, y))

x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 100,    200,
  "chr1", 250,    400,
  "chr1", 500,    600,
  "chr1", 1000,   1200,
  "chr1", 1300,   1500
)

y <- tibble::tribble(
  ~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	<i>Identify intervals within a specified distance.</i>
------------	--

Description

Identify intervals within a specified distance.

Usage

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

Arguments

x	ivl_df
y	ivl_df
genome	genome_df
...	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

<https://bedtools.readthedocs.io/en/latest/content/tools/window.html>

Other multiple set operations: [bed_closest\(\)](#), [bed_coverage\(\)](#), [bed_intersect\(\)](#), [bed_map\(\)](#), [bed_subtract\(\)](#)

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 25,    50,
  "chr1", 100,   125
)

y <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 60,    75
)

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 125
)
```

```

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

x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 10, 100,
  "chr2", 200, 400,
  "chr2", 300, 500,
  "chr2", 800, 900
)

y <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 150, 400,
  "chr2", 230, 430,
  "chr2", 350, 430
)

genome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 500,
  "chr2", 1000
)

bed_window(x, y, genome, both = 100)

```

bound_intervals	<i>Select intervals bounded by a genome.</i>
-----------------	--

Description

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

Usage

```
bound_intervals(x, genome, trim = FALSE)
```

Arguments

x	ivl_df
genome	genome_df
trim	adjust coordinates for out-of-bounds intervals

Value

[ivl_df](#)

See Also

Other utilities: [bed12_to_exons\(\)](#), [bed_makewindows\(\)](#), [flip_strands\(\)](#), [interval_spacing\(\)](#)

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", -100,   500,
  "chr1", 100,   1e9,
  "chr1", 500,   1000
)

genome <- read_genome(valr_example("hg19.chrom.sizes.gz"))

# out-of-bounds are removed by default ...
bound_intervals(x, genome)

# ... or can be trimmed within the bounds of a genome
bound_intervals(x, genome, trim = TRUE)
```

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 [ivl_df](#) 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

```
create_tss(x)
```

Arguments

x [ivl_df](#) in BED format

See Also

Other feature functions: [create_introns\(\)](#), [create_utrs3\(\)](#), [create_utrs5\(\)](#)

Examples

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

create_utrs3	Create 3' UTR features.
--------------	-------------------------

Description

Create 3' UTR features.

Usage

```
create_utrs3(x)
```

Arguments

x [ivl_df](#) 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 [ivl_df](#) 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 = "ensembl.ensembl.org",
  user = "anonymous",
  password = "",
  port = 3306,
  ...
)

```

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://www.ensembl.org/info/data/mysql.html>

Examples

```

## Not run:
if (require(RMariaDB)) {
  library(dplyr)
  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(RMariaDB)) {
  library(dplyr)
  # squirrel genome
  ensembl <- db_ensembl("spermophilus_tridecemlineatus_core_67_2")

  tbl(ensembl, "gene")
}

## End(Not run)
```

flip_strands	<i>Flip strands in intervals.</i>
--------------	-----------------------------------

Description

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

Usage

```
flip_strands(x)
```

Arguments

x [ivl_df](#)

See Also

Other utilities: [bed12_to_exons\(\)](#), [bed_makewindows\(\)](#), [bound_intervals\(\)](#), [interval_spacing\(\)](#)

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end, ~strand,
  "chr1", 1,      100, "+",
  "chr2", 1,      100, "-"
)

flip_strands(x)
```

gr_to_bed	<i>Convert Granges to bed tibble</i>
-----------	--------------------------------------

Description

Convert Granges to bed tibble

Usage

```
gr_to_bed(x)
```

Arguments

x GRanges object to convert to bed tibble.

Value

```
tibble::tibble()
```

Examples

```
## 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)
  )
)

gr_to_bed(gr)

# There are two ways to convert a bed-like data.frame to GRanges:

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:

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

## End(Not run)
```

interval_spacing	<i>Calculate interval spacing.</i>
------------------	------------------------------------

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 [ivl_df](#)

Value

[ivl_df](#) with .spacing column.

See Also

Other utilities: [bed12_to_exons\(\)](#), [bed_makewindows\(\)](#), [bound_intervals\(\)](#), [flip_strands\(\)](#)

Examples

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 1,      100,
  "chr1", 150,    200,
  "chr2", 200,    300
)

interval_spacing(x)
```

ivl_df*Bed-like data.frame requirements for valr functions*

Description

Required column names for interval dataframes are chrom, start and end. Internally interval dataframes are validated using `check_interval()`

Required column names for genome dataframes are chrom and size. Internally genome dataframes are validated using `check_genome()`.

Usage

```
check_interval(x)
```

```
check_genome(x)
```

Arguments

x A `data.frame` or `tibble::tibble`

Examples

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

check_interval(x)

# using base R data.frame
x <- data.frame(
  chrom = "chr1",
  start = 0,
  end = 100,
  stringsAsFactors = FALSE
)

check_interval(x)

# example genome input
x <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 1e6
)
```

```
check_genome(x)
```

read_bed	<i>Read BED and related files.</i>
----------	------------------------------------

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,
  col_types = bed12_coltypes,
  sort = TRUE,
  ...,
  n_fields = NULL
)
```

```
read_bed12(filename, ...)
```

```
read_bedgraph(filename, ...)
```

```
read_narrowpeak(filename, ...)
```

```
read_broadpeak(filename, ...)
```

Arguments

filename	file or URL
col_types	column type spec for <code>readr::read_tsv()</code>
sort	sort the tbl by chrom and start
...	options to pass to <code>readr::read_tsv()</code>
n_fields	[Deprecated]

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`ivl_df`**See Also**Other read functions: `read_genome()`, `read_vcf()`**Examples**

```
# read_bed assumes 3 field BED format.
read_bed(valr_example("3fields.bed.gz"))

# result is sorted by chrom and start unless `sort = FALSE`
read_bed(valr_example("3fields.bed.gz"), sort = FALSE)

read_bed12(valr_example("mm9.refGene.bed.gz"))

read_bedgraph(valr_example("test.bg.gz"))

read_narrowpeak(valr_example("sample.narrowPeak.gz"))

read_broadpeak(valr_example("sample.broadPeak.gz"))
```

`read_bigwig`*Read a bigwig file into a valr compatible tbl*

Description

This function will output a 4 column tibble with zero-based chrom, start, end, value columns.

Usage`read_bigwig(path, ...)`**Arguments**

<code>path</code>	path to bigWig file
<code>...</code>	params for <code>cpp11bigwig::read_bigwig()</code>

Examples

```
read_bigwig(valr_example("hg19.dnase1.bw"))

read_bigwig(valr_example("hg19.dnase1.bw"), as = "GRanges")
```

read_genome	<i>Read genome files.</i>
-------------	---------------------------

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

[genome_df](#), sorted by size

Note

URLs to genome files can also be used.

See Also

Other read functions: [read_bed\(\)](#), [read_vcf\(\)](#)

Examples

```
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_gtf	<i>Import and convert a GTF/GFF file into a valr compatible bed tbl format</i>
----------	--

Description

[Deprecated]

This function will output a tibble with the required chrom, start, and end columns, as well as other columns depending on content in GTF/GFF file.

Usage

```
read_gtf(path, zero_based = TRUE)
```

Arguments

path	path to gtf or gff file
zero_based	if TRUE, convert to zero based

Examples

```
## Not run:  
gtf <- read_gtf(valr_example("hg19.gencode.gtf.gz"))  
head(gtf)  
  
## End(Not run)
```

read_vcf	<i>Read a VCF file.</i>
----------	-------------------------

Description

Read a VCF file.

Usage

```
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

```
vcf_file <- valr_example("test.vcf.gz")
read_vcf(vcf_file)
```

valr

valr: genome interval arithmetic in R

Description

valr provides tools to read and manipulate intervals and signals on a genome reference. valr was developed to facilitate interactive analysis of genome-scale data sets, leveraging the power of dplyr and piping.

Details

To learn more about valr, start with the vignette: `browseVignettes(package = "valr")`

Author(s)

Jay Hesselberth jay.hesselberth@gmail.com

Kent Riemony kent.riemony@gmail.com

See Also

Report bugs at <https://github.com/rnabioco/valr/issues>

valr_example	<i>Provide working directory for valr example files.</i>
--------------	--

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

* feature functions

- [create_introns](#), 37
- [create_tss](#), 38
- [create_utrs3](#), 38
- [create_utrs5](#), 39

* interval statistics

- [bed_absdist](#), 3
- [bed_fisher](#), 10
- [bed_jaccard](#), 17
- [bed_projection](#), 24
- [bed_reldist](#), 27

* multiple set operations

- [bed_closest](#), 4
- [bed_coverage](#), 9
- [bed_intersect](#), 15
- [bed_map](#), 19
- [bed_subtract](#), 33
- [bed_window](#), 35

* randomizing operations

- [bed_random](#), 26
- [bed_shuffle](#), 29

* read functions

- [read_bed](#), 45
- [read_genome](#), 47
- [read_vcf](#), 48

* single set operations

- [bed_cluster](#), 6
- [bed_complement](#), 7
- [bed_flank](#), 11
- [bed_genomecov](#), 12
- [bed_merge](#), 21
- [bed_partition](#), 23
- [bed_shift](#), 28
- [bed_slop](#), 31

* utilities

- [bed12_to_exons](#), 3
- [bed_makewindows](#), 18
- [bound_intervals](#), 36
- [flip_strands](#), 41

- [interval_spacing](#), 43

- [bed12_to_exons](#), 3, 18, 36, 41, 43
- [bed_absdist](#), 3, 11, 17, 25, 27
- [bed_closest](#), 4, 9, 15, 20, 34, 35
- [bed_cluster](#), 6, 8, 12, 13, 22, 23, 28, 31
- [bed_complement](#), 7, 7, 12, 13, 22, 23, 28, 31
- [bed_coverage](#), 5, 9, 15, 20, 34, 35
- [bed_fisher](#), 4, 10, 17, 25, 27
- [bed_flank](#), 7, 8, 11, 13, 22, 23, 28, 31
- [bed_genomecov](#), 7, 8, 12, 12, 22, 23, 28, 31
- [bed_glyph](#), 14
- [bed_intersect](#), 5, 9, 15, 20, 34, 35
- [bed_intersect\(\)](#), 19
- [bed_jaccard](#), 4, 11, 17, 25, 27
- [bed_makewindows](#), 3, 18, 36, 41, 43
- [bed_map](#), 5, 9, 15, 19, 34, 35
- [bed_map\(\)](#), 19
- [bed_merge](#), 7, 8, 12, 13, 21, 23, 28, 31
- [bed_partition](#), 7, 8, 12, 13, 22, 23, 28, 31
- [bed_projection](#), 4, 11, 17, 24, 27
- [bed_random](#), 26, 30
- [bed_reldist](#), 4, 11, 17, 25, 27
- [bed_shift](#), 7, 8, 12, 13, 22, 23, 28, 31
- [bed_shuffle](#), 26, 29
- [bed_slop](#), 7, 8, 12, 13, 22, 23, 28, 31
- [bed_sort](#), 32
- [bed_subtract](#), 5, 9, 15, 20, 33, 35
- [bed_window](#), 5, 9, 15, 20, 34, 35
- [bound_intervals](#), 3, 18, 36, 41, 43

- [check_genome \(ivl_df\)](#), 44
- [check_interval \(ivl_df\)](#), 44
- [concat \(bed_map\)](#), 19
- [create_introns](#), 37, 38, 39
- [create_tss](#), 37, 38, 38, 39
- [create_utrs3](#), 37, 38, 38, 39
- [create_utrs5](#), 37, 38, 39

- [db](#), 39

db_ensembl (db), 39
db_ucsc (db), 39
dplyr::do(), 4, 10, 17, 25, 27
dplyr::group_by(), 4, 5, 7, 9, 10, 13, 15, 17,
20, 22, 23, 25, 27, 34, 35

flip_strands, 3, 18, 36, 41, 43
flip_strands(), 5, 7, 9, 13, 15, 20, 22, 23,
34, 35

genome_df, 3, 10, 11, 13, 25, 26, 30, 31, 35,
36, 47
genome_df (ivl_df), 44
ggplot2::ggplot(), 14
gr_to_bed, 42

interval_spacing, 3, 18, 36, 41, 43
ivl_df, 3–13, 15, 17–20, 22, 23, 25–28,
30–33, 35–39, 41, 43, 44, 46
ivl_df(), 15, 23

max(), 19, 23
min(), 19, 23

read_bed, 45, 47, 49
read_bed12 (read_bed), 45
read_bedgraph (read_bed), 45
read_bigwig, 46
read_broadpeak (read_bed), 45
read_genome, 46, 47, 49
read_gtf, 48
read_narrowpeak (read_bed), 45
read_vcf, 46, 47, 48
readr::read_tsv(), 45

tibble::tibble(), 42

valr, 49
valr-package (valr), 49
valr_example, 50
values (bed_map), 19
values_unique (bed_map), 19