Package ‘valr’

December 9, 2021

Type Package
Title Genome Interval Arithmetic
Version 0.6.4
License MIT + file LICENSE
BugReports https://github.com/rnabioco/valr/issues
Depends R (>= 3.1.2)
Imports broom, dplyr (>= 0.8.0), ggplot2, Rcpp (>= 1.0.0), readr, rlang, rtracklayer, stringr, tibble (>= 1.4.2)
Suggests bench, covr, cowplot, curl, DBI, dbplyr, devtools, DT, GenomicRanges, IRanges, knitr, purrr, RMariaDB, rmarkdown, S4Vectors, testthat, tidyr
LinkingTo Rcpp (>= 1.0.0)
VignetteBuilder knitr
Encoding UTF-8
RoxygenNote 7.1.2
SystemRequirements C++11
NeedsCompilation yes
Author Jay Hesselberth [aut] (<https://orcid.org/0000-0002-6299-179X>), Kent Riemondy [aut, cre] (<https://orcid.org/0000-0003-0750-1273>), RNA Bioscience Initiative [fnd]
Maintainer Kent Riemondy <kent.riemondy@cuanschutz.edu>
Repository CRAN
Date/Publication 2021-12-08 23:00:02 UTC
R topics documented:

- bed12_to_exons ......................................................... 3
- bed_absdist .......................................................... 3
- bed_closest ............................................................ 4
- bed_cluster ............................................................. 6
- bed_complement ........................................................ 7
- bed_coverage ............................................................ 8
- bed_fisher .............................................................. 10
- bed_flank ............................................................... 11
- bed_glyph ............................................................... 12
- bed_intersect ......................................................... 13
- bed_jaccard ............................................................. 15
- bed_makewindows ..................................................... 16
- bed_map ................................................................. 18
- bed_merge .............................................................. 20
- bed_partition ........................................................ 21
- bed_projection ......................................................... 23
- bed_random ............................................................ 24
- bed_reldist ............................................................ 25
- bed_shift ............................................................... 26
- bed_shuffle ........................................................... 27
- bed_slop ............................................................... 29
- bed_sort ............................................................... 30
- bed_subtract ........................................................ 31
- bed_window ............................................................ 33
- bound_intervals ..................................................... 34
- create_introns ....................................................... 35
- create_tss ............................................................ 36
- create_utrs3 .......................................................... 36
- create_utrs5 .......................................................... 37
- db ...................................................................... 37
- flip_strands .......................................................... 39
- gr_to_bed ............................................................. 40
- interval_spacing ..................................................... 41
- ivl_df ................................................................. 42
- read_bed ............................................................... 43
- read_bigwig .......................................................... 44
- read_genome .......................................................... 45
- read_gtf .............................................................. 46
- read_vcf ............................................................. 46
- valr ................................................................. 47
- valr_example .......................................................... 47

Index ................................................................. 49
### 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**

```r
bed12_to_exons(x)
```

**Arguments**

- `x` (ivl_df)

**See Also**

Other utilities: `bed_makewindows()`, `bound_intervals()`, `flip_strands()`, `interval_spacing()`

**Examples**

```r
x <- read_bed12(valr_example('/quotesingle.Var/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**

```r
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, \( \text{absdist} \) 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 \( \text{absdist} \) and \( \text{absdist\_scaled} \).

Interval statistics can be used in combination with \texttt{dplyr::group\_by()} and \texttt{dplyr::do()} to calculate statistics for subsets of data. See vignette('interval-stats') for examples.

Value

\( \text{ivl\_df} \) with \( \text{absdist} \) and \( \text{absdist\_scaled} \) columns.

See Also

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

Other interval statistics: \texttt{bed\_fisher()}, \texttt{bed\_jaccard()}, \texttt{bed\_projection()}, \texttt{bed\_reldist()}

Examples

```r
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**

*Identify closest intervals.*

**Description**

Identify closest intervals.

**Usage**

```
bed\_closest(x, y, overlap = TRUE, suffix = c(".x", ".y"))
```

**Arguments**

- **x** \( \text{ivl\_df} \)
- **y** \( \text{ivl\_df} \)
- **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:

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

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

```r
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)
```
```r
# 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**

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

**Arguments**

- `x` \( \text{ivl_df} \)
- `max_dist` \( \text{maximum distance between clustered intervals.} \)

**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_merge(), bed_partition(),
bed_shift(), bed_slop()

Examples

```r
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
```r
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  'chr1', 1, 10,
  'chr1', 5, 20,
  'chr1', 30, 40,
  'chr1', 40, 50,
  'chr1', 80, 90
)
```

```r
bed_glyph(bed_cluster(x), label = '.id')
```

---

`bed_complement` Identify intervals in a genome not covered by a query.

**Description**

Identify intervals in a genome not covered by a query.

**Usage**

```r
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_merge()`, `bed_partition()`, `bed_shift()`, `bed_slop()`

Examples

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

geno <- tibble::tribble(
  ~chrom, ~size,
  'chr1', 200
)

bed_glyph(bed_complement(x, geno))

geno <- 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, geno)
```

---

**bed_coverage**

*Compute coverage of intervals.*

---

**Description**

Compute coverage of intervals.
**bed_coverage**

**Usage**

```r
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**

The `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**

```r
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, '-'
                  )
```
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`: 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](https://bedtools.readthedocs.io/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, 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**

```r
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 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**

`ivl_df`

**See Also**

[https://bedtools.readthedocs.io/en/latest/content/tools/flank.html](https://bedtools.readthedocs.io/en/latest/content/tools/flank.html)

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

```r
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))
```

```r
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)
```

---

**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**

```r
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')
```

---

**Description**

Identify intersecting intervals.

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

```r
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,
)```
### bed_jaccard

Calculate the Jaccard statistic for two sets of intervals.

#### Description

Quantifies the extent of overlap between 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` `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

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

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

bed_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

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

Arguments

- **x**: ivl_df
- **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

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

```r
x <- tibble::tribble(
  ~chrom, ~start, ~end, ~name, ~score, ~strand,
  "chr1", 100, 200, "VarA", 1, "+

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 `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`: `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](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**

```r
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 tibls 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 tibls, strands on the `y` tbl can first be inverted using `flip_strands()`.

**Value**

- `ivl_df`

**See Also**

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

**Examples**

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

bed_glyph(bed_merge(x))
```
```r
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**

*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`: ivl_df
- `...`: 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

`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_merge()`, `bed_shift()`, `bed_slop()`

Examples

```r
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**

```r
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

```r
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**  
*Generate randomly placed intervals on a genome.*

Description

Generate randomly placed intervals on a genome.

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>genome</td>
<td>genome_df</td>
</tr>
<tr>
<td>length</td>
<td>length of intervals</td>
</tr>
<tr>
<td>n</td>
<td>number of intervals to generate</td>
</tr>
<tr>
<td>seed</td>
<td>seed RNG for reproducible intervals</td>
</tr>
<tr>
<td>sorted</td>
<td>return sorted output</td>
</tr>
</tbody>
</table>

Details

Sorting can be suppressed with `sorted = FALSE`.

Value

`ivl_df`

See Also

[https://bedtools.readthedocs.io/en/latest/content/tools/random.html](https://bedtools.readthedocs.io/en/latest/content/tools/random.html)

Other randomizing operations: `bed_shuffle()`
Examples

```r
genie <- 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**

*Compute relative distances between intervals.*

**Description**

Compute relative distances between intervals.

**Usage**

```r
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

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

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

bed_reldist(x, y)

bed_reldist(x, y, detail = TRUE)

---

**bed_shift**

*Adjust intervals by a fixed size.*

**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_merge()`, `bed_partition()`, `bed_slop()`
Examples

```r
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

```r
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](https://bedtools.readthedocs.io/en/latest/content/tools/shuffle.html)
- Other randomizing operations: `bed_random()`

Examples

```r
genoome <- tibble::tribble(
  ~chrom, ~size,
  "chr1", 1e6,
  "chr2", 2e6,
  "chr3", 4e6
)

x <- bed_random(genoome, seed = 1010486)

bed_shuffle(x, genome, seed = 9830491)
```
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`: ivl_df
- `genome`: genome_df
- `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

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_merge(), bed_partition(), bed_shift()
Examples

```r
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**

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` : ivl_df
- `by_size` : sort by interval size
**bed_subtract**

Subtract two sets of intervals.

**Description**

Subtract y intervals from x intervals.

**Usage**

```r
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

```r
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**

Identify intervals within a specified distance.

**Description**

Identify intervals within a specified distance.

**Usage**

```r
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](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**

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

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

gene <- tibble::tribble(
  ~chrom, ~size,
  'chr1', 125
)
```
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`: 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()`
create_introns

Examples

```r
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

```r
create_introns(x)
```

Arguments

- `x` ivl_df in BED12 format

See Also

Other feature functions: `create_tss()`, `create_utrs3()`, `create_utrs5()`

Examples

```r
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

```r
x <- read.bed12(valr_example('mm9.refGene.bed.gz'))
create_utrs3(x)
create_utrs5(x)
```

**Description**
Create 5' UTR features.

**Usage**
```r
create_utrs5(x)
```

**Arguments**
- `x` ivl_df in BED12 format

**See Also**
Other feature functions: `create_introns()`, `create_tss()`, `create_utrs3()`

**Examples**

```r
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

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

```r
db_ensembl(
  dbname,
  host = "ensembldb.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://genome.ucsc.edu/goldenpath/help/mysql.html)
- [https://www.ensembl.org/info/data/mysql.html](https://www.ensembl.org/info/data/mysql.html)

### Examples

```r
## Not run:
if(require(RMariaDB)) {
  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:
flip_strands

if(require(RMariaDB)) {
    # squirrel genome
    ensembl <- db_ensembl("spermophilus_tridecemlineatus_core_67_2")

    tbl(ensembl, "gene")
}

## End(Not run)

flip_strands

Flip strands in intervals.

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

Convert Granges to bed tibble

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(sequenced = S4Vectors::Rle(c("chr1", "chr2", "chr1", "chr3"), c(1, 1, 1, 1)),ranges = IRanges::IRanges(start = c(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(sequenced = 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))
### 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

```r
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

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

interval_spacing(x)
```
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**

**Read BED and related files.**

**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**

```r
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/FAQ/FAQformat.html#format1](https://genome.ucsc.edu/FAQ/FAQformat.html#format1)
- [https://genome.ucsc.edu/goldenPath/help/bedgraph.html](https://genome.ucsc.edu/goldenPath/help/bedgraph.html)
- [https://genome.ucsc.edu/FAQ/FAQformat.html#format12](https://genome.ucsc.edu/FAQ/FAQformat.html#format12)
- [https://genome.ucsc.edu/FAQ/FAQformat.html#format13](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'))

read_bed(valr_example('6fields.bed.gz'), n_fields = 6)

# 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

Import and convert a bigwig file into a valr compatible tbl

Description

This function will output a 5 column tibble with zero-based chrom, start, end, score, and strand columns.

Usage

read_bigwig(path, set_strand = "+")

Arguments

path       path to bigWig file
set_strand strand to add to output (defaults to "+")

Note

This function uses rtracklayer to import bigwigs which has unstable support for the windows platform and therefore may error for windows users (particularly for 32 bit window users).
### Examples

```r
## Not run:
if (.Platform$OS.type != "windows") {
  bw <- read_bigwig(valr_example('hg19.dnase1.bw'))
  head(bw)
}
## End(Not run)
```

### 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

```r
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

```r
read_genome(valr_example('hg19.chrom.sizes.gz'))
```
read_gtf
Import and convert a GTF/GFF file into a valr compatible bed tbl format

Description
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

```r
gtf <- read_gtf(valr_example('hg19.gencode.gtf.gz'))
head(gtf)
```

read_vcf
Read a VCF file.

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

```r
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 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

```r
valr_example(path)
```

Arguments

path path to file
Examples

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

* feature functions
  create_introns, 35
  create_tss, 36
  create_utrs3, 36
  create_utrs5, 37

* interval statistics
  bed_absdist, 3
  bed_fisher, 10
  bed_jaccard, 15
  bed_projection, 23
  bed_reldist, 25

* multiple set operations
  bed_closest, 4
  bed_coverage, 8
  bed_intersect, 13
  bed_map, 18
  bed_subtract, 31
  bed_window, 33

* randomizing operations
  bed_random, 24
  bed_shuffle, 27

* read functions
  read_bed, 43
  read_genome, 45
  read_vcf, 46

* single set operations
  bed_cluster, 6
  bed_complement, 7
  bed_flank, 11
  bed_merge, 20
  bed_partition, 21
  bed_shift, 26
  bed_slop, 29

* utilities
  bed12_to_exons, 3
  bed_makewindows, 16
  bound_intervals, 34
  flip_strands, 39
  interval_spacing, 41
  bed12_to_exons, 3, 17, 34, 39, 41
  bed_absdist, 3, 10, 16, 23, 26
  bed_closest, 4, 9, 14, 19, 32, 33
  bed_cluster, 6, 8, 11, 20, 22, 26, 29
  bed_complement, 7, 7, 11, 20, 22, 26, 29
  bed_coverage, 5, 8, 14, 19, 32, 33
  bed_fisher, 4, 10, 16, 23, 26
  bed_flank, 7, 8, 11, 20, 22, 26, 29
  bed_glyph, 12
  bed_intersect, 5, 9, 13, 19, 32, 33
  bed_intersect(), 18
  bed_jaccard, 4, 10, 15, 23, 26
  bed_makewindows, 3, 16, 34, 39, 41
  bed_map, 5, 9, 14, 18, 32, 33
  bed_map(), 18
  bed_merge, 7, 8, 11, 20, 22, 26, 29
  bed_partition, 7, 8, 11, 20, 21, 26, 29
  bed_projection, 4, 10, 16, 23, 26
  bed_random, 24, 28
  bed_reldist, 4, 10, 16, 23, 25
  bed_shift, 7, 8, 11, 20, 22, 26, 29
  bed_shuffle, 24, 27
  bed_slop, 7, 8, 11, 20, 22, 26, 29
  bed_sort, 30
  bed_subtract, 5, 9, 14, 19, 31, 33
  bed_window, 5, 9, 14, 19, 32, 33
  bound_intervals, 3, 17, 34, 39, 41
  check_genome (ivl_df), 42
  check_interval (ivl_df), 42
  concat (bed_map), 18
  count(), 18, 21
  create_introns, 35, 36, 37
  create_tss, 35, 36, 37
  create_utrs3, 35, 36, 37
  create_utrs5, 35, 36, 37
  db, 37
  db_ensembl (db), 37
  db_ucsc (db), 37
dplyr::do(), 4, 10, 16, 23, 25
dplyr::group_by(), 4–6, 9, 10, 14, 16, 18,
   20, 21, 23, 25, 32, 33

flip_strands, 3, 17, 34, 39, 41
flip_strands(), 5, 6, 9, 14, 18, 20, 21, 32, 33

genome_df, 3, 10, 11, 23, 24, 28, 29, 33, 34, 45
genome_df(ivl_df), 42
ggplot2::ggplot(), 13
gr_to_bed, 40

interval_spacing, 3, 17, 34, 39, 41
ivl_df, 3–11, 13–15, 17–21, 23–26, 28–31,
   33–37, 39, 41, 42, 43
ivl_df(), 13, 22

min(), 18, 21

read_bed, 43, 45, 47
read_bed12(read_bed), 43
read_bedgraph(read_bed), 43
read_bigwig, 44
read_broadpeak(read_bed), 43
read_genome, 43, 45, 47
read_gtf, 46
read_narrowpeak(read_bed), 43
read_vcf, 43, 45, 46
readr::read_tsv(), 43

tibble::tibble(), 40

valr, 47
valr-package(valr), 47
valr_example, 47
values(bed_map), 18
values_unique(bed_map), 18