Package 'gUtils'

January 20, 2018

Title R Package Providing Additional Capabilities and Speed for GenomicRanges Operations

Version 0.2.0

Description R package providing additional capabilities and speed for GenomicRanges operations.

Depends R (>= 3.1.0), GenomicRanges (>= 1.18), data.table (>= 1.9)

Imports IRanges (>= 2.0), S4Vectors (>= 0.4), GenomeInfoDb (>= 1.2), parallel, BiocGenerics(>= 0.12), methods, Matrix, stringr

Suggests BSgenome.Hsapiens.UCSC.hg19, testthat, rtracklayer

License GPL-2

BugReports http://github.com/mskilab/gUtils/issues

LazyData true

RoxygenNote 6.0.1.9000

NeedsCompilation no

Author Jeremiah Wala [aut], Marcin Imielinski [aut, cre]

Maintainer Marcin Imielinski <mimielinski@nygenome.org>

R topics documented:

anchorlift . . . 3
dt2gr . . . 4
example_dnase . . . 4
example_genes . . . 4
gr.bind . . . 5
gr.chr . . . 5
gr.collapse . . . 6
gr.dice . . . 6
gr.disjoin . . . 7
gr.dist . . . 7
gr.duplicated . . . 8
gr.end . . . 9
gr.findoverlaps . . . 9
gr.fix . . . 10
gr.flatten . . . 11
gr.in . . . 11

gr.match . . . 12
gr.merge . . . 13
gr.mid . . . 13
gr.nochr . . . 14
gr.pairflip . . . 14
gr.rand . . . 15
gr.sample . . . 15
gr.simplify . . . 16
gr.start . . . 17
gr.strandflip . . . 17
gr.string . . . 18
gr.stripstrand . . . 19
gr.sub . . . 19
gr.sum . . . 20
gr.tile . . . 20
gr.tile.map . . . 21
gr.trim . . . 22
gr.val . . . 22
gr2dt . . . 24
grl.bind . . . 24
grl.eval . . . 25
grl.hiC . . . 25
grl.in . . . 26
grl.pivot . . . 26
grl.reduce . . . 27
grl.string . . . 28
grl.unlist . . . 28
grl1 . . . 29
grl2 . . . 29
hg_seqlengths . . . 30
parse.gr . . . 30
parse.grl . . . 31
ra.merge . . . 31
rle.query . . . 32
rrbind . . . 33
seg2gr . . . 34
si . . . 34
si2gr . . . 35
standardize_segs . . . 35
streduce . . . 36
%&% . . . 37
%&&% . . . 38
%+% . . . 38
%-% . . . 39
%Q% . . . 39
%_% . . . 40
%O% . . . 41
%OO% . . . 41
%$$% . . . 42
%NN% . . . 42
%N% . . . 43
%oo% . . . 43

%o% 44
%**% 44
%*% 45

Index **46**

---

anchorlift *anchorlift*

Description

"lifts" all queries with respect to subject in coordinates that are within "pad" i.e. puts the queries into subject-centric coordinates, which is a new genome with label "Anchor" (default)

Respects strand of subject (i.e. if subject strand gr is "-" then will lift all queries to the left of it into positive subject-centric coordinates). Keeps track of subject and query id for later deconvolution if need be.

Usage

```
anchorlift(query, subject, window = 1e+09, by = NULL, seqname = "Anchor",
  include.values = TRUE)
```

Arguments

`query`	GRanges that will be lifted around the subject
`subject`	GRanges around which the queries will be lifted
`window`	integer specifying how far around each subject to gather query intervals to lift (default = 1e9)
`by`	character vector specifying additional columms (e.g. sample id) around which to restrict overlaps (via gr.findoverlaps()). Refer to 'gr.findoverlaps()' documentation. (default = NULL)
`seqname`	String specifying the name of the output sequence around which to anchor (default = "Anchor")
`include.values`	Boolean Flag whether to include values from query and subject (default = TRUE)

Value

anchorlifted GRanges

Author(s)

Marcin Imielinski

dt2gr *Convert data.table to GRanges*

Description

Takes as input a data.table which must have the following fields: `start`, `end`, `strand`, `seqnames`. Will throw an error if any one of these is not present. All of the remaining fields are added as metadata to the `GRanges`.

Usage

```
dt2gr(dt, key = NULL, seqlengths = hg_seqlengths(), seqinfo = Seqinfo())
```

Arguments

`dt`	data.table or data.frame to convert to `GRanges`
`seqlengths`	named integer vector representing genome (default = hg_seqlengths())
`seqinfo`	seqinfo of output GRanges object

Value

`GRanges` object of `length = nrow(dt)`

Examples

```
converted_gr = dt2gr(data.table(start=c(1,2), seqnames=c("X", "1"), end=c(10,20), strand = c('+', '-')))
```

example_dnase *DNAaseI hypersensitivity sites for hg19A*

Description

DNAaseI hypersensitivity sites from UCSC Table Browser hg19, subsampled to 10,000 sites

Format

`GRanges`

example_genes *RefSeq genes for hg19*

Description

RefSeq genes with exon count and name

Format

`GRanges`

gr.bind — *Concatenate* GRanges*, robust to different* mcols

Description

Concatenates GRanges objects, taking the union of their features if they have non-overlapping features

Usage

```
gr.bind(x, ...)
```

Arguments

x	GRanges input GRanges
...	additional input GRanges

Value

Concatenated GRanges gr.bind(example_genes, example_dnase)

Note

Does not fill in the Seqinfo for the output GRanges

gr.chr — *Prepend "chr" to* GRanges seqlevels

Description

Prepend "chr" to GRanges seqlevels

Usage

```
gr.chr(gr)
```

Arguments

gr	GRanges object to append 'chr' to

Value

Identical GRanges, but with 'chr' prepended to each seqlevel

Examples

```
gr <- gr.chr(GRanges(c(1,"chrX"), IRanges(c(1,2), 1)))
seqnames(gr)
```

gr.collapse *Collapse adjacent ranges*

Description

Like GenomicRanges::reduce except only collapses «adjacent» ranges in the input

Usage

```
gr.collapse(gr, pad = 1)
```

Arguments

gr	GRanges to collapse
pad	Padding that allows for not quite adjacent elements to be considered overlapping. (default = 1)

Value

GRanges with collapsed adjacent GRanges

Author(s)

Marcin Imielinski

gr.dice *Dice up* GRanges *into* width = 1 GRanges *spanning the input (warning can produce a very large object)*

Description

Dice up GRanges into width = 1 GRanges spanning the input (warning can produce a very large object)

Usage

```
gr.dice(gr)
```

Arguments

gr	GRanges object to dice

Value

GRangesList where kth element is a diced pile of GRanges from kth input GRanges

Author(s)

Marcin Imielinski

Examples

```
gr.dice(GRanges(c(1,4), IRanges(c(10,10),20)))
```

---

gr.disjoin *GenomicRanges disjoin with some additional functionality*

Description

Identical to GRanges disjoin, except outputs inherit metadata from first overlapping parent instance on input

Usage

```
gr.disjoin(x, ..., ignore.strand = TRUE)
```

Arguments

x	GRanges to disjoin
...	arguments to disjoin (e.g. with.revmap=FALSE). Please see documentation for GenomicRanges::disjoin()
ignore.strand	logical scalar (default = TRUE)

Value

GRanges of non-overlapping ranges with metadata

---

gr.dist *Pairwise distance between two* GRanges

Description

Computes matrix of pairwise distance between elements of two GRanges objects of length n and m.

Distances are computed as follows:

- NA for ranges on different seqnames
- 0 for overlapping ranges
- min(abs(end1-end2), abs(end1-start2), abs(start1-end2), abs(start1-end1),) for all others

If only gr1 is provided, then will return n x n matrix of gr1 vs itself
If max.dist = TRUE, then will replace min with max above

Usage

```
gr.dist(gr1, gr2 = NULL, ignore.strand = FALSE, ...)
```

Arguments

gr1	First GRanges
gr2	Second GRanges
ignore.strand	Don't required elements be on same strand to avoid NA [FALSE]
...	Additional arguments to be supplied to GenomicRanges::distance

Value

N by M matrix with the pairwise distances, with gr1 on rows and gr2 on cols

Author(s)

Marcin Imielinski

---

gr.duplicated *Allows to restrict duplicates using "by" columns and allows in exact matching*

Description

Allows to restrict duplicates using "by" columns and allows in exact matching

Usage

```
gr.duplicated(query, by = NULL, type = "any")
```

Arguments

query	GRanges to query
by	string Column 'by' used to restrict duplicates. See the 'by' argument for function gr.match()
type	string 'type' used. See the 'type' argument for function gr.match()

Value

boolean vector of match status

gr.end — *Get the right ends of a* GRanges

Description

Alternative to GenomicRanges::flank that will provide end positions *within* intervals

Usage

```
gr.end(x, width = 1, force = FALSE, ignore.strand = TRUE, clip = TRUE)
```

Arguments

x	GRanges object to operate on
width	integer Specify subranges of greater width including the start of the range. (default = 1)
force	boolean Allows returned GRanges to have ranges outside of its Seqinfo bounds. (default = FALSE)
ignore.strand	boolean If set to FALSE, will extend '-' strands from the other direction. (default = TRUE)
clip	boolean Trims returned GRanges so that it does not extend beyond bounds of the input (default = TRUE)

Value

GRanges object of width = width ranges representing end of each genomic range in the input.

Author(s)

Marcin Imielinski

Examples

```
gr.end(example_dnase, width=200, clip=TRUE)
```

gr.findoverlaps — *Wrapper to* GenomicRanges::findOverlaps *with added functionality*

Description

Wrapper to GenomicRanges::findOverlaps with added functionality

Returns GRanges of matches with two additional fields:

- $query.id - index of matching query
- $subject.id - index of matching subject

Optional "by" field is a character scalar that specifies a metadata column present in both query and subject that will be used to additionally restrict matches, i.e. to pairs of ranges that overlap and also have the same values of their "by" fields

Usage

```
gr.findoverlaps(query, subject, ignore.strand = TRUE, first = FALSE,
  qcol = NULL, scol = NULL, type = "any", by = NULL,
  return.type = "same", max.chunk = 1e+13, verbose = FALSE,
  mc.cores = 1, ...)
```

Arguments

query	Query GRanges pile
subject	Subject GRanges pile
ignore.strand	Don't consider strand information during overlaps. (default = TRUE)
first	boolean Flag if TRUE restricts to only the first match of the subject. If FALSE will return all matches. (default = FALSE)
qcol	character vector of query meta-data columns to add to results (default = NULL)
scol	character vector of subject meta-data columns to add to results (default = NULL)
type	type argument as defined by IRanges::findOverlaps ("any", "start", "end", "within", "equal"). (default = 'any')
by	vector Meta-data column to consider when performing overlaps (default = NULL)
return.type	character Select data format to return (supplied as character): "same", "data.table", "GRanges". (default = 'same')
max.chunk	integer Maximum number of query*subject ranges to consider at once. Lower number increases runtime but decreased memory. If length(query)*length(subject) is less than max.chunk, overlaps will run in one batch. (default = 1e3)
verbose	boolean Flag to increase the verbosity. (default = FALSE)
mc.cores	Number of cores to use when running in chunked mode (default = 1)
...	Additional arguments sent to IRanges::findOverlaps.

Value

GRanges pile of the intersection regions, with query.id and subject.id marking sources

---

gr.fix *"Fixes"* seqlengths / seqlevels

Description

If "genome" not specified will replace NA seqlengths in GRanges to reflect largest coordinate per seqlevel and removes all NA seqlevels after this fix.

If "genome" defined (i.e. as Seqinfo object, or a BSgenome, GRanges, GRangesList object with populated seqlengths), then will replace seqlengths in gr with those for that genome

Usage

```
gr.fix(gr, genome = NULL, gname = NULL, drop = FALSE)
```

Arguments

gr	GRanges object to fix
genome	Genome to fix to: Seqinfo, BSgenome, GRanges (w/seqlengths), GRangesList (w/seqlengths) (default = NULL)
gname	string Name of the genome (optional, just appends to Seqinfo of the output) (default = NULL)
drop	boolean Remove ranges that are not present in the supplied genome (default = FALSE)

Value

GRanges pile with the fixed Seqinfo

---

gr.flatten *Lay ranges end-to-end onto a derivate "chromosome"*

Description

Takes pile of GRanges and returns into a data.frame with nrow = length(gr) with each representing the corresponding input range superimposed onto a single "flattened" chromosome, with ranges laid end-to-end

Usage

```
gr.flatten(gr, gap = 0)
```

Arguments

gr	GRanges to flatten
gap	integer Number of bases between ranges on the new chromosome (default = 0)

Value

data.frame with start and end coordinates, and all of the original metadata

---

gr.in *Versatile implementation of* GenomicRanges::findOverlaps

---

Description

Versatile implementation of GenomicRanges::findOverlaps

Returns boolean vector. TRUE if query range i is found in any range in subject.

Usage

```
gr.in(query, subject, ...)
```

Arguments

query	GRanges Set of GRanges to query. Refer to gr.findoverlaps() and GenomicRanges::findOverlaps()
subject	GRanges Set of GRanges as 'subject' in query. Refer to gr.findoverlaps() and GenomicRanges::findOverlaps()
...	Arguments to be passed to gr.findoverlaps

Value

boolean vector whereby TRUE is if query range i is found in any range in subject

---

gr.match *Alternative* GenomicRanges::match *that accepts additional* gr.findoverlaps *options*

Description

Alternative GenomicRanges::match that accepts additional gr.findoverlaps options

Wrapper to GenomicRanges::match (uses gr.findoverlaps). This allows users to match on additional by fields, or chunk into smaller pieces for lower memory.

Usage

```
gr.match(query, subject, max.slice = Inf, verbose = FALSE, ...)
```

Arguments

query	Query GRanges pile
subject	Subject GRanges pile
max.slice	max slice of query to match at a time
verbose	whether to give verbose output
...	Additional arguments to be passed along to gr.findoverlaps.

Value

Vector of length = length(query) with subject indices of *first* subject in query, or NA if none found. This behavior is different from gr.findoverlaps, which will return *all* indicies of subject in query (in the case of one query overlaps with multiple subject) ... = additional args for findOverlaps (IRanges version)

Author(s)

Marcin Imielinski

gr.merge *merge GRanges using coordinates as primary key*

Description

Uses gr.findoverlaps() to enable internal and external joins of GRanges using syntax similar to "merge" where merging is done using coordinates +/- "by" fields

Uses gr.findoverlaps() / GRanges::findOverlaps for heavy lifting, but returns outputs with metadata populated as well as query and subject ids. For external joins, overlaps x with gaps(y) and gaps(x) with y.

Usage

```
gr.merge(query, subject, by = NULL, all = FALSE, all.query = all,
  all.subject = all, ignore.strand = TRUE, verbose = FALSE, ...)
```

Arguments

query	GRanges Set of GRanges to query. Refer to gr.findoverlaps() and GenomicRanges::findOverlaps()
subject	GRanges Set of GRanges as 'subject' in query. Refer to gr.findoverlaps() and GenomicRanges::findOverlaps()
by	vector Additional metadata fields to join on
all	boolean Flag whether to include left and right joins
all.query	boolean Flag whether to do a left join (default = all)
all.subject	boolean Flag whether to do a right join (default = all)

Value

GRanges merged on 'by' vector

gr.mid *Get the midpoints of* GRanges *ranges*

Description

Get the midpoints of GRanges ranges

Usage

```
gr.mid(x)
```

Arguments

x	GRanges object to operate on

Value

GRanges of the midpoint, calculated from `floor(width(x)/2)`

Examples

```
gr.mid(GRanges(1, IRanges(1000,2000), seqinfo=Seqinfo("1", 2000)))
```

`gr.nochr` *Remove chr prefix from GRanges seqlevels*

Description

Remove chr prefix from GRanges seqlevels

Usage

```
gr.nochr(gr)
```

Arguments

`gr`	`GRanges` with chr seqlevel prefixes

Value

GRanges without chr seqlevel prefixes

`gr.pairflip` *Create pairs of ranges and their strand-inverse*

Description

From a `GRanges` returns a `GRangesList` with each item consisting of the original `GRanges` and its strand flip

Usage

```
gr.pairflip(gr)
```

Arguments

`gr`	`GRanges`

Value

`GRangesList` with each element of length 2

gr.rand *Generate random* GRanges *on genome*

Description

Randomly generates non-overlapping GRanges with supplied widths on supplied genome. Seed can be supplied with set.seed

Usage

```
gr.rand(w, genome)
```

Arguments

w	vector of widths (length of w determines length of output)
genome	GRanges, GRangesList, or Seqinfo genome. Default is "hg19" from the BSGenome package.

Value

GRanges with random intervals on the specifed "chromosomes"

Note

This function is currently quite slow, needs optimization

Author(s)

Marcin Imielinski

Examples

```
## Generate 5 non-overlapping regions of width 10 on hg19
gr.rand(rep(10,5), BSgenome.Hsapiens.UCSC.hg19::Hsapiens)
```

gr.sample *Randomly sample* GRanges *intervals within territory*

Description

Samples k intervals of length "len" from a pile of GRanges.

- If k is a scalar then will (uniformly) select k intervals from the summed territory of GRanges
- If k is a vector of length(gr) then will uniformly select k intervals from each.

Usage

```
gr.sample(gr, k, wid = 100, replace = TRUE)
```

Arguments

gr	Granges defining the territory to sample from
k	integer Number of ranges to sample
wid	integer Length of the GRanges element to produce (default = 100)
replace	boolean If TRUE, will bootstrap, otherwise will sample without replacement. (default = TRUE)

Value

GRanges of max length sum(k) [if k is vector) or k*length(gr) (if k is scalar) with labels indicating the originating range.

Note

This is different from GenomicRanges::sample function, which just samples from a pile of GRanges

Author(s)

Marcin Imielinski

Examples

```
## sample 5 \code{GRanges} of length 10 each from territory of RefSeq genes
gr.sample(reduce(example_genes), k=5, wid=10)
```

---

gr.simplify *Calc pairwise distance for rearrangements represented by GRangesList objects*

Description

Calc pairwise distance for rearrangements represented by GRangesList objects

Usage

```
gr.simplify(gr, field = NULL, val = NULL, include.val = TRUE,
  split = FALSE, pad = 1)
```

Arguments

gr	GRanges or GRangesList input
field	character scalar, corresponding to value field of gr. (default = NULL)
val	character scalar (default = NULL)
include.val	boolean Flag will include in out gr values field of first matching record in input gr. [TRUE]
split	boolean Flag to split the output into GRangesList split by "field". [FALSE]
pad	integer Pad ranges by this amount before doing merge. [1], which merges contiguous but non-overlapping ranges.

Value

Simplified GRanges with "field" populated with uniquely contiguous values

---

`gr.start` *Get GRanges corresponding to beginning of range*

Description

Get GRanges corresponding to beginning of range

Usage

```
gr.start(x, width = 1, force = FALSE, ignore.strand = TRUE, clip = TRUE)
```

Arguments

`x`	`GRanges` object to operate on
`width`	integer Specify subranges of greater width including the start of the range. (default = 1)
`force`	boolean Allows returned `GRanges` to have ranges outside of its `Seqinfo` bounds. (default = FALSE)
`ignore.strand`	boolean If set to `FALSE`, will extend '-' strands from the other direction (default = TRUE)
`clip`	boolean Trims returned `GRanges` so that it does not extend beyond bounds of the input `GRanges` (default = TRUE)

Value

`GRanges` object of width 1 ranges representing start of each genomic range in the input.

Examples

```
gr.start(example_dnase, width=200)
gr.start(example_dnase, width=200, clip=TRUE)
```

`gr.strandflip` *Flip strand on* `GRanges`

Description

Flip strand on `GRanges`

Usage

```
gr.strandflip(gr)
```

Arguments

gr	GRanges pile with strands to be flipped

Value

GRanges with flipped strands (+ to -, * to *, - to *)

Examples

```
gr.strandflip(GRanges(1, IRanges(c(10,10,10),20), strand=c("+","*","-")))
```

---

gr.string — *Return UCSC style interval string corresponding to* GRanges *pile (ie chr:start-end)*

Description

Return UCSC style interval string corresponding to GRanges pile (ie chr:start-end)

Usage

```
gr.string(gr, add.chr = FALSE, mb = FALSE, round = 3, other.cols = c(),
  pretty = FALSE)
```

Arguments

gr	GRanges pile to get intervals from
add.chr	boolean Flag to prepend seqnames with "chr" (default = FALSE)
mb	boolean Flag to round to the nearest megabase (default = FALSE)
round	integer If mb supplied, the number of digits to round to. (default = 3)
other.cols	character vector Names of additional mcols fields to add to the string (seperated by ";")
pretty	boolean Flag to output interval string in more readable format

Value

UCSC style interval string corresponding to GRanges pile

Author(s)

Marcin Imielinski

Examples

```
gr.string(example_genes, other.cols = c("name", "name2"))
```

gr.stripstrand *gr.stripstrand*

Description

Sets strand to "*"

Usage

```
gr.stripstrand(gr)
```

Arguments

gr	GRanges to remove strand information from

Value

GRanges with strand set to *

gr.sub *Apply* gsub *to seqlevels of a* GRanges

Description

Apply gsub to seqlevels of gr, by default removing 'chr', and "0.1" suffixes, and replacing "MT" with "M"

Usage

```
gr.sub(gr, a = c("(^chr)(\\.1$)", "MT"), b = c("", "M"))
```

Arguments

gr	GRanges to switch out seqlevels for
a	vector of regular expressions of things to substitute out (default = c("(^chr)(\.1$)", "MT"))
b	vector of values to substitute in (default = c("", "M"))

Value

GRanges with substitutions

gr.sum *gr.sum*

Description

Sums GRanges either by doing coverage and either weighting them equally or using a field "weight". Will return either sum or average.

Usage

```
gr.sum(gr, field = NULL, mean = FALSE)
```

Arguments

gr	GRanges to sum
field	metadata field from gr to use as a weight
mean	logical scalar specifying whether to divide the output at each interval but the total number of intervals overlapping it (only applies if field == NULL) (default FALSE)

Value

non-overlapping GRanges spanning the seqlengths of gr with $score (if field is NULL) or $field specifying the sum / mean at that position

gr.tile *Tile ranges across* GRanges

Description

Tiles interval (or whole genome) with segments of <= specified width.

Usage

```
gr.tile(gr, width = 1000)
```

Arguments

gr	GRanges, seqlengths or Seqinfo range to tile. If has GRanges has overlaps, will reduce first.
width	integer Width of each tile (default = 1e3)

Value

GRanges with tiled intervals

Examples

```
## 10 tiles of width 10
gr1 <- gr.tile(GRanges(1, IRanges(1,100)), width=10)

## make them overlap each other by 5
gr1 + 5
```

`gr.tile.map` *gr.tile.map*

Description

Given two tilings of the genome (e.g. at different resolution) query and subject outputs a length(query) list whose items are integer vectors of indices in subject overlapping that overlap that query (strand non-specific)

Usage

```
gr.tile.map(query, subject, verbose = FALSE)
```

Arguments

`query`	Query `GRanges` pile, perhaps created from some tile (e.g. `gr.tile`), and assumed to have no gaps
`subject`	Subject `GRanges` pile, perhaps created from some tile (e.g. `gr.tile`), and assumed to have no gaps
`verbose`	Increase the verbosity of the output (default = FALSE)

Value

`list` of length = `length(query)`, where each element `i` is a vector of indicies in `subject` that overlaps element `i` of `query`

Note

Assumes that input query and subject have no gaps (including at end) or overlaps, i.e. ignores end() coordinates and only uses "starts"

Author(s)

Marcin Imielinski

gr.trim — *Trims pile of* GRanges *relative to the specified <local> coordinates of each range*

Description

Example: GRanges with genomic coordinates 1:1,000,000-1,001,000 can get the first 20 and last 50 bases trimmed off with `start = 20, end = 950`. if end is larger than the width of the corresponding gr, then the corresponding output will only have `end(gr)` as its coordinate.

This is a role not currently provided by the standard GenomicRanges functions (e.g. `shift`, `reduce`, `restrict`, `shift`, `resize`, `flank`)

Usage

```
gr.trim(gr, starts = 1, ends = 1)
```

Arguments

gr	GRanges to trim
starts	Number of bases to trim off of the front[1]
ends	Number of bases to trim off of the back[1]

Value

GRanges with trimmed intervals relative to the specified <local> coordinates of each range

Examples

```
## trim the first 20 and last 50 bases
gr.trim(GRanges(1, IRanges(1e6, width=1000)), starts=20, ends=950)
## return value: GRanges on 1:1,000,019-1,000,949
```

gr.val — *Annotate* GRanges *with values from another* GRanges

Description

Annotates GRanges in `query` with aggregated values of GRanges in `target` in field `val`. If `val` is numeric: given `target` with value column `target` representing ranged data (i.e. segment intensities), thn computes the value in each `query` GRanges as the weighted mean of its intersection with target (ie the target values weighted by the width of the intersections).

Applications include (among others):

- Querying the average value of target across a given query interval (e.g. exon to gene pileup)
- recasting a high res tiling in terms of low res intervals.

Usually query intervals are bigger than the target intervals.

Usage

```
gr.val(query, target, val = NULL, mean = TRUE, weighted = mean,
  na.rm = FALSE, by = NULL, by.prefix = val, merge = FALSE,
  FUN = NULL, default.val = NA, max.slice = Inf, mc.cores = 1,
  sep = ", ", verbose = FALSE, ...)
```

Arguments

query	GRanges of query ranges whose val column we will populate with aggregated values of target
target	GRanges of target ranges that already have "val" column populated
val	If a character field: then aggregation will paste together the (unique), overlapping values, collapsing by comma. (default = NULL)
mean	boolean If FALSE then will return sum instead of mean, only applies if target val column is numeric. (default = TRUE)
weighted	Calculate a weighted mean. If FALSE, calculates unweighted mean. (default = 'mean')
na.rm	boolean Remove NA values when calulating means. only applies if val column of target is numeric (default = FALSE)
by	scalar character, specifies additional "by" column of query AND target that will be used to match up query and target pairs (i.e. in addition to pure GRanges overlap). (default = NULL)
by.prefix	Choose a set of val fields by a shared prefix. (default = 'val')
merge	boolean If merge = FALSE then will cross every range in query with every level of "by" in target (and create data matrix), otherwise will assume query has "by" and merge only ranges that have matching "by" values in both query and target (default = FALSE)
FUN	Optional different function to call than mean. Takes two arguments (value, na.rm = TRUE) if weighted = FALSE, and three (value, width, na.rm = TRUE) if weighted = TRUE. (default = NULL)
default.val	If no hit in target found in query, fill output val field with this value. (default = NA)
max.slice	integer Maximum number of query ranges to consider in one memory chunk. (default = Inf)
mc.cores	integer Number of cores to use when running in chunked mode (default = 1)
sep	string Specifies character to use as separator when aggregating character "vals" from target, only applies if target is character (default = ', ')
verbose	boolean Increase the verbosity of the output (default = FALSE)
...	Additional arguments to be sent to gr.findoverlaps.

Value

query with the val field populated

Note

query and target can be GRangesList object, in which case val will refer to GRangesList level values fields

Author(s)

Marcin Imielinski

---

gr2dt *Converts* GRanges *to* data.table

Description

Converts GRanges to data.table and a field grl.iix which saves the (local) index that that gr was in its corresponding grl item

Usage

```
gr2dt(x)
```

Arguments

x	GRanges to convert

Value

data.table of GRanges columns ('seqnames', 'start', 'end', 'strand', 'width') and metadata columns

---

grl.bind *Concatenate* GRangesList *objects.*

Description

Concatenates GRangesList objects taking the union of their mcols features if they have non-overlapping features

Usage

```
grl.bind(...)
```

Arguments

...	GRangesList Any number of GRangesList to concatenate together

Value

Concatenated GRangesList with NA filled in for mcols fields that are non-overlapping. Note that the elements are re-named with sequential numbers

Author(s)

Marcin Imielinski

Examples

```
## Concatenate
grl.hiC2 <- grl.hiC[1:20]
mcols(grl.hiC2)$test = 1
grl.bind(grl.hiC2, grl.hiC[1:30])
```

`grl.eval` *Evaluate and aggregate expression on GRanges column in GRangesList*

Description

Evaluate expression 'expr' on indivdual GRanges inside GRangesList. Expression should result in a single i.e. scalar value per GRangesList item.

Usage

```
grl.eval(grl, expr, condition = NULL)
```

Arguments

`grl`	GRangesList to evaluate over
`expr`	Any syntactically valid R expression, on columns of GRanges or GRangesList
`condition`	Optional: any syntactically valid R expression, on columns of GRanges or GRangesList, on which to subset before evaluating main 'expr' (default = NULL)

Value

GRangesList evaluated by R expressions

`grl.hiC` *HiC data for chr14 from Lieberman-Aiden 2009 (in hg19), subsampled to 10,000 interactions*

Description

HiC data for chr14 from Lieberman-Aiden 2009 (in hg19), subsampled to 10,000 interactions

Format

`GRangesList`

grl.in *Check intersection of* GRangesList *with windows on genome*

Description

Check intersection of GRangesList with windows on genome

Like only if the ranges in grl[i] intersect «all», «some», «only» windows in the subject

e.g. can use to identify read pairs whose ends are contained inside two genes)

Usage

```
grl.in(grl, windows, some = FALSE, only = FALSE, logical = TRUE,
  exact = FALSE, ignore.strand = TRUE, ...)
```

Arguments

grl	GRangesList object to query for membership in windows
windows	GRanges pile of windows
some	boolean Will return TRUE for GRangesList elements that intersect at least on window range (default = FALSE)
only	boolean Will return TRUE for GRangesList elements only if there are no elements of query that fail to interesect with windows (default = FALSE)
logical	boolean Will return logical otherwise will return numeric vector of number of windows overlapping each grl (default = TRUE)
exact	boolean Will return exact intersection
...	Additional parameters to be passed on to GenomicRanges::findOverlaps

Value

boolean vector of match status

grl.pivot *Pivot a* GRangesList*, inverting "x" and "y"*

Description

"Pivots" GRangesList object "x" by returning a new GRangesList "y" whose kth item is GRanges object of ranges x[[i]][k] for all i in 1:length(x)

e.g. If the length of a GRangesList 'grl' is 50, 'length(grl)=50 and length of each GRanges element inside is 2, then the function grl.pivot will produce a length 3 GRangesList with 50 elements per GRanges

Note: Assumes all GRanges in "x" are of equal length

Usage

```
grl.pivot(x)
```

Arguments

x	GRangesList object to pivot

Value

GRangesList with inverted 'x' and 'y'

Examples

```
grl.pivot(grl.hiC)
```

---

grl.reduce *grl.reduce*

Description

Quickly computes GRanges +/- padding inside a GRangesList Can use with split / unlist

Usage

```
grl.reduce(grl, pad = 0, clip = FALSE)
```

Arguments

grl	GRangesList input
pad	integer Padding to add to ranges inside GRangesList before reducing (default = 0)
clip	boolean Flag to add to ranges inside GRangesList before reducing (default = FALSE)

Value

GRangesList with GRanges of intervals "original GRanges +/- padding"

Author(s)

Marcin Imielinski

grl.string *Create string representation of* GRangesList

Description

Return UCSC style interval string corresponding to each GRanges in the GRangesList. One line per per GRangesList item. GRanges elements themselves are separated by sep

Usage

```
grl.string(grl, mb = FALSE, sep = ",", ...)
```

Arguments

grl	GRangesList to convert to string vector
mb	boolean Flag to return as MB and round to "round" (default = FALSE)
sep	Character to separate single GRanges ranges (default = ',')
...	Additional arguments to be passed to gr.string

Value

Character vector where each element is a GRanges pile corresponding to a single GRangesList element

Author(s)

Marcin Imielinski

Examples

```
grl.string(grl.hiC, mb=TRUE)
```

grl.unlist *Robust unlisting of* GRangesList *that keeps track of origin*

Description

Robust unlisting of GRangesList that keeps track of origin

Does a "nice" unlist of a GRangesList object adding a field grl.ix denoting which element of the GRangesList each GRanges corresponds to and a field grl.iix which saves the (local) index that that gr was in its corresponding GRangesList item

In this way, grl.unlist is reversible, while BiocGenerics::unlist is not.

Usage

```
grl.unlist(grl)
```

Arguments

grl	GRangeList object to unlist

Value

GRanges with added metadata fields grl.ix and grl.iix.

Examples

```
grl.unlist(grl.hiC)
```

---

grl1 *Fake rearrangement data (set 1)*

Description

Fake rearrangement data (set 1)

Format

GRangesList

---

grl2 *Fake rearrangement data (set 2)*

Description

Fake rearrangement data (set 2)

Format

GRangesList

hg_seqlengths *Output standard human genome seqlengths*

Description

Outputs a standard seqlengths for human genome +/- "chr".

Usage

```
hg_seqlengths(genome = NULL, chr = FALSE, include.junk = FALSE)
```

Arguments

genome	A BSgenome or object with a seqlengths accessor. Default is hg19, but loads with warning unless explicitly provided
chr	boolean Flag for whether to keep "chr". (default = FALSE)
include.junk	boolean Flag for whether to not trim to only 1-22, X, Y, M. (default = FALSE)

Value

Named integer vector with elements corresponding to the genome seqlengths

Note

A default genome can be set with the environment variable DEFAULT_BSGENOME. This can be the full namespace of the genome e.g.: DEFAULT_BSGENOME=BSgenome.Hsapiens.UCSC.hg19::Hsapiens OR a URL / file path pointing to a chrom.sizes text file (e.g. http://genome.ucsc.edu/goldenpath/help/hg19.chrom.sizes) specifying a genome definition

Author(s)

Marcin Imielinski

parse.gr *parse.gr*

Description

Quick function to parse GRanges from character vector IGV-/UCSC-style strings of format gr1;gr2;gr3 wohere each gr is of format chr:start-end[+/-]

Usage

```
parse.gr(...)
```

Arguments

...	arguments to parse.grl i.e. character strings in UCSC style chr:start-end[+-]

Value

GRanges parsed from IGV-/UCSC-style strings

Author(s)

Marcin Imielinski

`parse.grl` *parse.grl*

Description

Quick function to parse `GRangesList` from character vector IGV / UCSC style strings of format gr1;gr2;gr3 where each gr is of format chr:start-end[+/-]

Usage

```
parse.grl(x, seqlengths = hg_seqlengths())
```

Arguments

`x`	character vector representing a GRangesList with UCSC style coordinates (chr:start-end[+-]) representing a [signed] Granges and ";" separators within each item of x separating individaul each GRAnges
`seqlengths`	named integer vector representing genome (default = hg_seqlengths())

Value

GRangesList parsed from IGV-/UCSC-style strings

Author(s)

Marcin Imielinski

`ra.merge` *Merges rearrangements represented by* `GRangesList` *objects*

Description

Determines overlaps between two or more piles of rearrangement junctions (as named or numbered arguments) +/- padding and will merge those that overlap into single junctions in the output, and then keep track for each output junction which of the input junctions it was "seen in" using logical flag meta data fields prefixed by "seen.by." and then the argument name (or "seen.by.ra" and the argument number)

Usage

```
ra.merge(..., pad = 0, ind = FALSE, ignore.strand = FALSE)
```

Arguments

...	GRangesLists which represent rearrangements to be merged
pad	integer specifying padding (default = 0)
ind	boolean Flag specifying whether the "seen.by" fields should contain indices of inputs (rather than logical flags) and NA if the given junction is missing (default = FALSE)
ignore.strand	boolean Flag specifying whether to ignore strand (implies all strand information will be ignored, use at your own risk). Refer to documentation for function 'ra.overlaps()'. (default = FALSE)

Value

GRangesList of merged junctions with meta data fields specifying which of the inputs each outputted junction was "seen.by"

Examples

```
# generate some junctions
gr1 <- GRanges(1, IRanges(1:10, width = 1), strand = rep(c('+', '-'), 5))
gr2 <- GRanges(1, IRanges(4 + 1:10, width = 1), strand = rep(c('+', '-'), 5))
ra1 = split(gr1, rep(1:5, each = 2))
ra2 = split(gr2, rep(1:5, each = 2))

ram = ra.merge(ra1, ra2)
values(ram) # shows the metadata with TRUE / FALSE flags

ram2 = ra.merge(ra1, ra2, pad = 5) # more inexact matching results in more merging
values(ram2)

ram3 = ra.merge(ra1, ra2, ind = TRUE) #indices instead of flags
values(ram3)
```

---

rle.query *Queries an RleList representing genomic data*

Description

Queries an RleList representing genomic data

(i.e. a list whose names represent seqnames ie chromosomes, and lengths represent seqlengths) via GRanges object

Usage

```
rle.query(subject.rle, query.gr, mc.cores = 1, chunksize = 1e+09,
  verbose = FALSE)
```

Arguments

subject.rle	Rle
query.gr	GRangeslist or GRanges
mc.cores	Number of cores to apply when doing chunked operation
chunksize	integer Number of query.gr ranges to consider in one memory chunk. (default = 1e9)
verbose	Set the verbosity of the output

Value

Rle representing the (concatenated) vector of data (reversing order in case of negative strand input)

Note

Throws warning if seqlengths(gr) do not correspond to the lengths of the RleList components

---

rrbind *Improved* rbind *for intersecting/union columns of* data.frames *or* data.tables

---

Description

Improved rbind for intersecting/union columns of data.frames or data.tables

Like rbind, but takes the intersecting columns of the data.

Usage

```
rrbind(..., union = TRUE, as.data.table = FALSE)
```

Arguments

...	Any number of data.frame or data.table objects
union	Take union of columns (and put NA's for columns of df1 not in df2 and vice versa). (default = TRUE)
as.data.table	Return the binded data as a data.table. (default = FALSE)

Value

data.frame or data.table of the rbind operation

Author(s)

Marcin Imielinski

seg2gr *Convert GRanges-like data.frame into GRanges*

Description

Input data.frame of segments "segs" and converts into GRanges object porting over additional value columns

"segs" data.frame/data.table can obey any number of conventions to specify chrom, start, and end of ranges (e.g. $pos1, $pos2, $Start_position, $End_position)

Please see documentation for function 'standardize_segs()' for more details.

Usage

```
seg2gr(segs, seqlengths = NULL, seqinfo = Seqinfo())
```

Arguments

segs	data.frame (or data.table) of segments with fields denoting chromosome, start, end, and other metadata. (See function 'standardize_segs()' for 'seg' data.frame/data.table input formats)
seqlengths	seqlengths of output GRanges object (default = NULL)
seqinfo	seqinfo of output GRanges object (default = Seqinfo())

Value

GRanges from converted "segs" data.frame/data.table

si Seqinfo *object for hg19*

Description

Seqinfo object for hg19

Format

Seqinfo

si2gr — *Create* GRanges *from* Seqinfo *or* BSgenome

Description

Creates a genomic ranges from seqinfo object i.e. a pile of ranges spanning the genome

Usage

```
si2gr(si, strip.empty = FALSE)
```

Arguments

si	Seqinfo object or a BSgenome genome
strip.empty	boolean Flag to output non-zero GRanges only (default = FALSE)

Value

GRanges representing the range of the input genome

Examples

```
si2gr(BSgenome.Hsapiens.UCSC.hg19::Hsapiens)
```

standardize_segs — *Takes and returns segs data frame standardized to a single format*

Description

Takes and returns segs data frame standardized to a single format (i.e. $chr, $pos1, $pos2)

If chr = TRUE will ensure "chr" prefix is added to chromossome(if does not exist)

"segs" data.frame can obey any number of conventions to specify chrom, start, and end of ranges (e.g. $pos1, $pos2, $Start_position, $End_position)

Conventions:

- ID - 'id', 'patient', 'sample'
- chr - 'seqnames', 'chrom', 'chromosome', 'rname', 'space', 'contig'
- pos1 - 'start', 'loc.start', 'start.bp', 'start_position', 'begin', 'pos', 'pos1', 'left', 's1'
- pos2 - 'end', 'loc.end', 'stop', 'end.bp', 'end_posiiton', 'pos2', 'right', 'e1'
- sstrand - 'strand', 'str'

Usage

```
standardize_segs(seg, chr = FALSE)
```

Arguments

seg	data.frame or data.table of segments with fields denoting chromosome, start, end, and other metadata.
chr	boolean Flag to force add chromosomes (default = FALSE)

Value

data.frame or data.table with standardized segments

---

streduce *Reduce* GRanges *and* GRangesList *to miminal footprint*

Description

Reduce GRanges and GRangesList to miminal footprint

Shortcut for reduce(sort(gr.stripstrand(unlist(x))))

Usage

```
streduce(gr, pad = 0, sort = TRUE)
```

Arguments

gr	GRanges or GRangesList
pad	integer Expand the input data before reducing. (default = 0)
sort	boolean Flag to sort the output. (default = TRUE)

Value

GRanges object with no strand information, representing a minimal footprint

Examples

```
streduce(grl.hiC, pad=10)
streduce(example_genes, pad=1000)
```

`%&%` *subset x on y ranges while ignoring strand (strand-agnostic)*

Description

shortcut for x[gr.in(x,y)]

gr1

Shortcut for gr.in (standard arguments)

gr1

Shortcut for gr.in

gr1

Usage

```
x %&% ...

x %^% ...

x %^^% ...
```

Arguments

x	`GRanges` object
...	additional arguments to gr.in
x	See gr.in
...	See gr.in
x	See gr.in
...	See gr.in

Value

subset of gr1 that overlaps gr2

logical vector of length gr1 which is TRUE at entry i only if gr1[i] intersects at least one interval in gr2 (strand agnostic)

logical vector of length gr1 which is TRUE at entry i only if gr1[i] intersects at least one interval in gr2

Author(s)

Marcin Imielinski

Marcin Imielinski

`%&&%` *Subset x on y ranges, strand-specific*

Description

shortcut for x[gr.in(x,y)]

gr1

Usage

```
x %&&% ...
```

Value

subset of gr1 that overlaps gr2

Author(s)

Marcin Imielinski

`%+%` *Nudge GRanges right*

Description

Operator to shift GRanges right "sh" bases

Usage

```
gr %+% ...
```

Value

shifted granges

Author(s)

Marcin Imielinski

%-% *Shift GRanges left*

Description

Operator to shift GRanges left "sh" bases

df

Usage

```
gr %-% ...
```

Value

shifted GRanges

Author(s)

Marcin Imielinski

%Q% *query ranges by applying an expression to ranges metadata*

Description

gr

Usage

```
x %Q% ...
```

Arguments

x	GRanges to match against a query GRanges
y	GRanges with metadata to be queried

Value

subset of gr that matches query

Author(s)

Marcin Imielinski

`%_%` `BiocGenerics::setdiff` *shortcut (strand agnostic)*

Description

Shortcut for `BiocGenerics::setdiff`

gr1 <- GRanges(1, IRanges(10,20), strand="+") gr2 <- GRanges(1, IRanges(15,25), strand="-") gr3 <- "1:1-15" gr1 gr1

More robust and faster implementation of GenomicRanges::setdiff()

Robust to common edge cases of setdiff(gr1, gr2) where gr2 ranges are contained inside gr1's (yieldings setdiffs yield two output ranges for some of the input gr1 intervals.

Usage

```
x %_% ...

gr.setdiff(query, subject, ignore.strand = TRUE, by = NULL, ...)
```

Arguments

`x`	`GRanges` object to to
`...`	A `GRanges` or a character to be parsed into a `GRanges`
`query`	`GRanges` object as query
`subject`	`GRanges` object as subject
`ignore.strand`	boolean Flag to ignore strands. Refer to 'gr.findoverlaps()'. (default = TRUE)
`by`	vector Meta-data column to consider when performing overlaps. Refer to 'gr.findoverlaps()' documentation (default = NULL)
`...`	arguments to be passed to gr.findoverlaps

Value

`GRanges` representing setdiff of input interval

Returns indices of query in subject. If none found, NA

Author(s)

Marcin Imielinski

%0% *gr.val shortcut to get fractional overlap of gr1 by gr2, strand-agnostic*

Description

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %0% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

fractional overlap of gr1 with gr2

Author(s)

Marcin Imielinski

%00% *gr.val shortcut to get fractional overlap of gr1 by gr2, strand-specific*

Description

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %00% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

fractional overlap of gr1 with gr2

Author(s)

Marcin Imielinski

%$$% *gr.val shortcut to get mean values of subject "x" meta data fields in query "y" (strand-specific)*

Description

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %$$% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

gr1 with extra meta data fields populated from gr2

Author(s)

Marcin Imielinski

%NN% *gr.val shortcut to get total numbers of intervals in gr2 overlapping with each interval in gr1, respecting strand*

Description

gr1

Usage

```
x %NN% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

bases overlap of gr1 with gr2

Author(s)

Marcin Imielinski

%N% — *gr.val shortcut to get total numbers of intervals in gr2 overlapping with each interval in gr1, ignoring strand*

Description

gr1

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %N% ...

x %$% ...
```

Arguments

x	See gr.val
...	See gr.val
x	GRanges object

Value

bases overlap of gr1 with gr2

gr1 with extra meta data fields populated from gr2

Author(s)

Marcin Imielinski

Marcin Imielinski

%oo% — *gr.val shortcut to total per interval width of overlap of gr1 with gr2, strand-specific*

Description

gr1

Usage

```
x %oo% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

bases overlap of gr1 with gr2

Author(s)

Marcin Imielinski

`%o%` *gr.val shortcut to total per interval width of overlap of gr1 with gr2, ignoring strand*

Description

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %o% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

bases overlap of gr1 with gr2

Author(s)

Marcin Imielinski

`%**%` *shortcut for gr.findoverlaps (strand-specific)*

Description

Shortcut for gr.findoverlaps

gr1

Usage

```
x %**% ...
```

Arguments

x	See gr.findoverlaps
...	See gr.findoverlaps

Value

new granges containing every pairwise intersection of ranges in gr1 and gr2 with a join of the corresponding metadata

Author(s)

Marcin Imielinski

`%*%` *Metadata join with coordinates as keys (wrapper to* `gr.findoverlaps`*)*

Description

Shortcut for gr.findoverlaps with `qcol` and `scol` filled in with all the query and subject metadata names. This function is useful for piping `GRanges` operations together. Another way to think of join of the metadata, with genomic coordinates as the keys.
Example usage:
x

Usage

```
## S4 method for signature 'GRanges,ANY'
x %*% y
```

Arguments

x	GRanges
y	GRanges

Value

`GRanges` containing every pairwise intersection of ranges in `x` and `y` with a join of the corresponding metadata

Author(s)

Marcin Imielinski

Examples

```
example_genes %*% example_dnase
```

Index

∗Topic **data**
example_dnase, 4
example_genes, 4
grl.hiC, 25
grl1, 29
grl2, 29
si, 34
%**%,GRanges-method *(%**%)*, 44
%*%,GRanges-method *(%*%)*, 45
%+%,GRanges-method *(%+%)*, 38
%NN%,GRanges-method *(%NN%)*, 42
%N%,GRanges-method *(%N%)*, 43
%OO%,GRanges-method *(%OO%)*, 41
%O%,GRanges-method *(%O%)*, 41
%Q%,GRanges-method *(%Q%)*, 39
%$$%,GRanges-method *(%$$%)*, 42
%$% *(%N%)*, 43
%$%,GRanges-method *(%N%)*, 43
%&%, *(%&%)*, 37
%&&%,GRanges-method *(%&&%)*, 38
%_%,GRanges-method *(%_%)*, 40
%^% *(%&%)*, 37
%^%,GRanges-method *(%&%)*, 37
%^^% *(%&%)*, 37
%^^%,GRanges-method *(%&%)*, 37
%o%,GRanges-method *(%o%)*, 44
%oo%,GRanges-method *(%oo%)*, 43
%**%, 44
%*%, 45
%+%, 38
%-%, 39
%NN%, 42
%N%, 43
%OO%, 41
%O%, 41
%Q%, 39
%$$%, 42
%&&%, 38
%&%, 37
%_%, 40
%o%, 44
%oo%, 43

anchorlift, 3

dt2gr, 4

example_dnase, 4
example_genes, 4

gr.bind, 5
gr.chr, 5
gr.collapse, 6
gr.dice, 6
gr.disjoin, 7
gr.dist, 7
gr.duplicated, 8
gr.end, 9
gr.findoverlaps, 9, *12*, *23*, *40*, *44*, *45*
gr.fix, 10
gr.flatten, 11
gr.in, 11, *37*
gr.match, 12
gr.merge, 13
gr.mid, 13
gr.nochr, 14
gr.pairflip, 14
gr.rand, 15
gr.sample, 15
gr.setdiff *(%_%)*, 40
gr.simplify, 16
gr.start, 17
gr.strandflip, 17
gr.string, 18
gr.stripstrand, 19
gr.sub, 19
gr.sum, 20
gr.tile, 20
gr.tile.map, 21
gr.trim, 22
gr.val, 22, *41–44*
gr2dt, 24
GRanges-method *(%&%)*, 37
grl.bind, 24
grl.eval, 25
grl.hiC, 25
grl.in, 26
grl.pivot, 26
grl.reduce, 27

grl.string, 28
grl.unlist, 28
grl1, 29
grl2, 29

hg_seqlengths, 30

parse.gr, 30
parse.grl, 31

ra.merge, 31
rle.query, 32
RleList, *32*
rrbind, 33

seg2gr, 34
si, 34
si2gr, 35
standardize_segs, 35
streduce, 36