Package ‘gggenomes’

February 23, 2026

Title A Grammar of Graphics for Comparative Genomics

Version 1.1.3

Description An extension of 'ggplot2' for creating complex genomic maps. It builds on the power of 'ggplot2' and 'tidyverse' adding new 'ggplot2'-style geoms & positions and 'dplyr'-style verbs to manipulate the underlying data. It implements a layout concept inspired by 'ggraph' and introduces tracks to bring tidiness to the mess that is genomics data.

License MIT + file LICENSE

URL https://thackl.github.io/gggenomes/, https://github.com/thackl/gggenomes

BugReports https://github.com/thackl/gggenomes/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

VignetteBuilder knitr

Depends R (>= 4.1.0), ggplot2 (>= 3.5.0),

Imports vctrs, rlang, dplyr (>= 1.1.0), tidyr, readr (>= 2.0.0), purrr, tibble, stringr, grid, jsonlite, snakecase, magrittr, scales, tidyselect, colorspace, methods, utils

Suggests testthat, ggtree, patchwork, Hmisc, knitr, ggrepel, IRanges, rmarkdown

NeedsCompilation no

Author Thomas Hackl [aut, cre], Markus J. Ankenbrand [aut], Bart van Adrichem [aut], Kristina Haslinger [ctb, sad]

Maintainer Thomas Hackl <t.hackl@rug.nl>

Repository CRAN

Date/Publication 2026-02-23 10:10:02 UTC

Contents

add_feats . . . 3
add_seqs . . . 5
align . . . 5
check_strand . . . 7
combine_strands . . . 7
def_formats . . . 8
def_names . . . 9
drop_feat_layout . . . 10
drop_layout . . . 10
drop_link_layout . . . 11
drop_seq_layout . . . 11
emale_ava . . . 12
emale_cogs . . . 12
emale_gc . . . 13
emale_genes . . . 14
emale_ngaros . . . 15
emale_prot_ava . . . 16
emale_seqs . . . 16
emale_tirs . . . 17
ex . . . 18
feats . . . 18
flip . . . 20
flip_strand . . . 22
focus . . . 22
GeomFeatText . . . 25
geom_bin_label . . . 26
geom_coverage . . . 28
geom_feat . . . 32
geom_feat_text . . . 34
geom_gene . . . 39
geom_gene_label . . . 43
geom_link . . . 45
geom_seq . . . 49
geom_seq_break . . . 51
geom_seq_label . . . 54
geom_variant . . . 56
get_seqs . . . 59
gggenomes . . . 60
if_reverse . . . 63
introduce . . . 63
in_range . . . 64
is_reverse . . . 65
layout . . . 65
layout_seqs . . . 66
pick . . . 67
position_strand . . . 69

position_variant 71
read_alitv 72
read_bed 73
read_blast 74
read_context 75
read_gbk 76
read_gff3 77
read_paf 78
read_seq_len 79
read_tracks 81
read_vcf 83
require_vars 84
scale_color_variant 85
scale_x_bp 87
set_class 88
shift 89
strand_chr 90
strand_int 90
strand_lgl 91
swap_if 91
swap_query 92
theme_gggenomes_clean 92
track_ids 93
track_info 93
unnest_exons 94
vars_track 95
width 95
write_gff3 96

Index **98**

`add_feats` *Add different types of tracks*

Description

Add different types of tracks

Usage

```
add_feats(x, ...)

add_links(x, ..., .adjacent_only = TRUE)

add_subfeats(x, ..., .track_id = "genes", .transform = "aa2nuc")

add_sublinks(x, ..., .track_id = "genes", .transform = "aa2nuc")

add_clusters(x, ..., .track_id = "genes")
```

Arguments

`x`	object to add the tracks to (e.g. gggenomes, gggenomes_layout)
`...`	named data.frames, i.e. genes=gene_df, snps=snp_df
`.adjacent_only`	indicate whether links should be drawn only between vertically adjacent tracks
`.track_id`	track_id of the feats that subfeats, sublinks or clusters map to.
`.transform`	one of "aa2nuc", "none", "nuc2aa"

Value

gggenomes object with added features

Functions

- `add_feats()`: Add feature annotations to sequences
- `add_links()`: Add links connecting sequences, such as whole-genome alignment data.
- `add_subfeats()`: Add features of features, such as gene/protein domains, blast hits to genes/proteins, etc.
- `add_sublinks()`: Add links that connect features, such as protein-protein alignments connecting genes.
- `add_clusters()`: Add gene clusters or other feature groups. Takes a data.frame with at least two required columns `cluster_id` and `feat_id`. The data.frame is converted to a link track connecting features belonging to the same cluster over their entire length. Additionally, the data.frame is joined to the parent feature track, adding `cluster_id` and all additional columns to the parent table.

Examples

```
# Add some repeat annotations
gggenomes(seqs = emale_seqs) %>%
  add_feats(repeats = emale_tirs) +
  geom_seq() + geom_feat()

# Add all-vs-all whole-genome alignments
gggenomes(seqs = emale_seqs) %>%
  add_links(links = emale_ava) +
  geom_seq() + geom_link()

# Add domains to genes
genes <- tibble::tibble(seq_id = "A", start = 100, end = 200, feat_id = "gene1")
domains <- tibble::tibble(feat_id = "gene1", start = 40, end = 80)
gggenomes(genes = genes) %>% add_subfeats(domains, .transform = "none") +
  geom_gene() + geom_feat()

# Add protein-protein alignments
gggenomes(emale_genes) %>%
  add_sublinks(emale_prot_ava) +
  geom_gene() + geom_link()
```

```
# add clusters
gggenomes(emale_genes, emale_seqs) %>%
  add_clusters(emale_cogs) %>%
  sync() + # works because clusters
  geom_link() + # become links
  geom_seq() +
  # works because cluster info is joined to gene track
  geom_gene(aes(fill = ifelse(is.na(cluster_id), NA,
    stringr::str_glue("{cluster_id} [{cluster_size}]")
  ))) +
  scale_fill_discrete(name="COGs")
```

`add_seqs` *Add seqs*

Description

Add seqs

Usage

```
add_seqs(x, seqs, ...)
```

Arguments

`x`	a gggenomes or gggenomes_layout object
`seqs`	the sequences to add
`...`	pass through to `as_seqs()`

Value

a gggenomes or gggenomes_layout object with added seqs

`align` *Align genomes relative to target genes, feats, seqs, etc.*

Description

Align your genomes relative to target features, such as genes or regions of interest. Use the `...` argument to indicate a subset of features in one track as targets. If multiple features are selected per bin, they are treated as a single feature spanning from the leftmost to the rightmost end. The genomes will be shifted so that the targets features align according to the `.justify`.

Usage

```
align(x, ..., .track_id = "genes", .justify = "left")
```

Arguments

x	gggenomes object
...	filter expression to identify target features in target track. Works like `dplyr::filter()` <data-masking>.
.track_id	track to pull from, default "genes"
.justify	alignment position, one of "left", "center", "right" or numeric between 0 and 1, default "left"

Value

gggenomes object with shifted coordinates.

Examples

```
library(patchwork)
p <- gggenomes(emale_genes, links = emale_ava) +
  geom_link() +
  geom_gene(aes(fill = name)) +
  scale_fill_brewer(palette = "Dark2", na.value = "cornsilk3") +
  geom_bin_label()

pp <-
  # left-align on MCP gene
  p |> align(name == "MCP") +
  # right-align on MCP gene
  p |> align(name == "MCP", .justify = "right") +
  # center-align on MCP + pri-hel gene after flipping bins
  p |> sync() |> align(name %in% c("MCP", "pri-hel"), .justify = "center") |>
    # and highlight the feature block we are aligning to
    locate(name %in% c("MCP", "pri-hel"), .expand = 0, .max_dist = 1e6) +
    geom_feat(data = feats(loci), color = "plum3", alpha = .5, linewidth = 5)

pp + plot_layout(guides = "collect") & geom_vline(xintercept = 0, linetype = 2)

# multi contig
s0 <- tibble::tibble(
  bin_id = c("A", "B", "B", "B", "C", "C", "C"),
  seq_id = c("a1", "b1", "b2", "b3", "c1", "c2", "c3"),
  length = c(1e4, 6e3, 2e3, 1e3, 3e3, 3e3, 3e3)
)

p <- gggenomes(seqs = s0) +
  geom_seq(aes(color = bin_id), linewidth = 3) +
  geom_bin_label() +
  geom_seq_label() +
  expand_limits(color = c("A", "B", "C"))

pp <-
  # center on everything - just omit ...
  p |> align(.track_id = "seqs", .justify = .5) +
  # right-align on contig ending in "2"
```

```
  # NOTE: there is no 2nd contig in bin A, so nothing is aligned there
  p |> align(stringr::str_detect(seq_id, "2"), .track_id = "seqs", .justify = "right")

pp + plot_layout(guides = "collect") & geom_vline(xintercept = 0, linetype = 2)
```

---

check_strand *Check strand*

Description

Check strand

Usage

```
check_strand(strand, na)
```

Arguments

strand	some representation for strandedness
na	what to use for NA

Value

strand vector with unknown values replaced by na

---

combine_strands *Combine strands*

Description

Combine strands

Usage

```
combine_strands(strand, strand2, ...)
```

Arguments

strand	first strand
strand2	second strand
...	more strands

Value

the combined strand

`def_formats` *Defined file formats and extensions*

Description

For seamless reading of different file formats, gggenomes uses a mapping of known formats to associated file extensions and contexts in which the different formats can be read. The notion of context allows one to read different information from the same format/extension. For example, a gbk file holds both feature and sequence information. If read in "feats" context `read_feats("*.gbk")` it will return a feature table, if read in "seqs" context `read_seqs("*.gbk")`, a sequence index.

Usage

```
def_formats(
  file = NULL,
  ext = NULL,
  context = NULL,
  parser = NULL,
  allow_na = FALSE
)
```

Arguments

`file`	a vector of file names
`ext`	a vector of file extensions
`context`	a vector of file contexts defined in `gggenomes_global$def_formats`
`parser`	a vector of file parsers defined in `gggenomes_global$def_formats`
`allow_na`	boolean

Value

dictionarish vector of file formats with recognized extensions as names

Defined formats, extensions, contexts, and parsers

```
    format                   ext        context                                parser
1  ambigious            txt, tsv, csv           NA                          read_ambigious
2    fasta fa, fas, fasta, ffn, fna, faa          seqs                           read_seq_len
3     fai                   fai          seqs                              read_fai
4    gff3       gff, gff3, gff2, gtf     feats, seqs                     read_gff3, read_seq_len
5     gbk        gbk, gb, gbff, gpff     feats, seqs                     read_gbk, read_seq_len
6     bed                   bed         feats                                read_bed
7    blast             m8, o6, o7     feats, links                     read_blast, read_blast
8     paf                   paf     feats, links                       read_paf, read_paf
9    alitv                  json feats, seqs, links read_alitv_genes, read_alitv_seqs, read_alitv_link
10    vcf                   vcf         feats                                read_vcf
```

Examples

```
# vector of defined zip formats and recognized extensions as names
# format of file
def_formats("foo.fa")

# formats associated with each extension
def_formats(ext = c("fa", "gff"))

# all formats/extensions that can be read in seqs context; includes formats
# that are defined for context=NA, i.e. that can be read in any context.
def_formats(context = "seqs")
```

`def_names` *Default column names and types for defined formats*

Description

Intended to be used in `readr::read_tsv()`-like functions that accept a `col_names` and a `col_types` argument.

Usage

```
def_names(format)

def_types(format)
```

Arguments

`format`	specify a format known to gggenomes, such as `gff3`, gbk, ...

Value

a vector with default column names for the given format

a vector with default column types for the given format

Functions

- `def_names()`: default column names for defined formats
- `def_types()`: default column types for defined formats

Defined formats, column types and names

```
gff3     ccciicccc     seq_id,source,type,start,end,score,strand,phase,attributes
paf      ciiicciiiiid   seq_id,length,start,end,strand,seq_id2,length2,start2,end2,map_match,map_le
blast    ccdiiiiiiidd   seq_id,seq_id2,pident,length,mismatch,gapopen,start,end,start2,end2,evalue
 bed        ciicdc          seq_id,start,end,name,score,strand
 fai        ci---           seq_id,seq_desc,length
 seq_len    cci             seq_id,seq_desc,length
 vcf       cicccdccc      seq_id,start,feat_id,ref,alt,qual,filter,info,format
```

Examples

```
# read a blast-tabular file with read_tsv
readr::read_tsv(ex("emales/emales-prot-ava.o6"), col_names = def_names("blast"))
```

---

drop_feat_layout *Drop feature layout*

Description

Drop feature layout

Usage

```
drop_feat_layout(x, keep = "strand")
```

Arguments

x	feat_layout
keep	features to keep

Value

feat_layout without unwanted features

---

drop_layout *Drop a genome layout*

Description

Drop a genome layout

Usage

```
drop_layout(data, ...)
```

Arguments

data	layout
...	additional data

Value

gggenomes object without layout

drop_link_layout *Drop a link layout*

Description

Drop a link layout

Usage

```
drop_link_layout(x, keep = "strand")
```

Arguments

x	link_layout
keep	features to keep

Value

link_layout without unwanted features

drop_seq_layout *Drop a seq layout*

Description

Drop a seq layout

Usage

```
drop_seq_layout(x, keep = "strand")
```

Arguments

x	seq_layout
keep	features to keep

Value

seq_layout without unwanted features

emale_ava — *All-versus-all whole genome alignments of 6 EMALE genomes*

Description

One row per alignment block. Alignments were computed with minimap2.

Usage

```
emale_ava
```

Format

A data frame with 125 rows and 23 columns

file_id name of the file the data was read from

seq_id identifier of the sequence the feature appears on

length length of the sequence

start start of the feature on the sequence

end end of the feature on the sequence

strand orientation of the feature relative to the sequence (+ or -)

seq_id2 identifier of the sequence the feature appears on

length2 length of the sequence

start2 start of the feature on the sequence

end2 end of the feature on the sequence

map_match, map_length, map_quality, NM, ms, AS, nn, tp, cm, s1, de, rl, cg see https://github.com/lh3/miniasm/blob/master/PAF.md for additional columns

Source

- Derived & bundled data: `ex("emales/emales.paf")`

emale_cogs — *Clusters of orthologs of 6 EMALE proteomes*

Description

One row per feature. Clusters are based on manual curation.

Usage

```
emale_cogs
```

Format

A data frame with 48 rows and 3 columns

cluster_id identifier of the cluster

feat_id identifier of the gene

cluster_size number of features in the cluster

Source

- Derived & bundled data: `ex("emales/emales-cogs.tsv")`

`emale_gc` *Relative GC-content along 6 EMALE genomes*

Description

One row per 50 bp window.

Usage

```
emale_gc
```

Format

A data frame with 2856 rows and 6 columns

file_id name of the file the data was read from

seq_id identifier of the sequence the feature appears on

start start of the feature on the sequence

end end of the feature on the sequence

name name of the feature

score relative GC-content of the window

Source

- Derived & bundled data: `ex("emales/emales-gc.bed")`

`emale_genes` *Gene annotations if 6 EMALE genomes (endogenous virophages)*

Description

A data set containing gene feature annotations for 6 endogenous virophages found in the genomes of the marine protist *Cafeteria burkhardae*.

Usage

```
emale_genes
```

Format

A data frame with 143 rows and 17 columns

file_id name of the file the data was read from

seq_id identifier of the sequence the feature appears on

start start of the feature on the sequence

end end of the feature on the sequence

strand reading orientation relative to sequence (+ or -)

type feature type (CDS, mRNA, gene, ...)

feat_id unique identifier of the feature

introns a list column with internal intron start/end positions

parent_ids a list column with parent IDs - feat_id's of parent features

source source of the annotation

score score of the annotation

phase For "CDS" features indicates where the next codon begins relative to the 5' start

width width of the feature

gc_content relative GC-content of the feature

name name of the feature

Note

geom_id an identifier telling the which features should be plotted as on items (usually CDS and mRNA of same gene)

Source

- Publication: doi:10.1101/2020.11.30.404863
- Raw data: `https://github.com/thackl/cb-emales`
- Derived & bundled data: `ex("emales/emales.gff")`

`emale_ngaros` *Integrated Ngaro retrotransposons of 6 EMALE genomes*

Description

Integrated Ngaro retrotransposons of 6 EMALE genomes

Usage

```
emale_ngaros
```

Format

A data frame with 3 rows and 14 columns

file_id name of the file the data was read from

seq_id identifier of the sequence the feature appears on

start start of the feature on the sequence

end end of the feature on the sequence

strand orientation of the feature relative to the sequence (+ or -)

type feature type (CDS, mRNA, gene, ...)

feat_id unique identifier of the feature

introns a list column with internal intron start/end positions

parent_ids a list column with parent IDs - feat_id's of parent features

source source of the annotation

score score of the annotation

phase For "CDS" features indicates where the next codon begins relative to the 5' start

name name of the feature

geom_id an identifier telling the which features should be plotted as on items (usually CDS and mRNA of same gene)

Source

- Publication: doi:10.1101/2020.11.30.404863
- Raw data: `https://github.com/thackl/cb-emales`
- Derived & bundled data: `ex("emales/emales-ngaros.gff")`

emale_prot_ava — *All-versus-all alignments 6 EMALE proteomes*

Description

One row per alignment. Alignments were computed with mmseqs2 (blast-like).

Usage

```
emale_prot_ava
```

Format

A data frame with 827 rows and 13 columns

file_id name of the file the data was read from

feat_id identifier of the first feature in the alignment

feat_id2 identifier of the second feature in the alignment

pident, length, mismatch, gapopen, start, end, start2, end2, evalue, bitscore see https://github.com/seqan/lambda/wiki/BLAST-Output-Formats for BLAST-tabular format columns

Source

- Derived & bundled data: `ex("emales/emales-prot-ava.o6")`

emale_seqs — *Sequence index of 6 EMALE genomes (endogenous virophages)*

Description

A data set containing the sequence information on 6 endogenous virophages found in the genomes of the marine protist *Cafeteria burkhardae*.

Usage

```
emale_seqs
```

Format

A data frame with 6 rows and 4 columns

file_id name of the file the data was read from

seq_id sequence identifier

seq_desc sequence description

length length of the sequence

Source

- Publication: doi:10.1101/2020.11.30.404863
- Raw data: https://github.com/thackl/cb-emales
- Derived & bundled data: `ex("emales/emales.fna")`

---

emale_tirs *Terminal inverted repeats of 6 EMALE genomes*

Description

Terminal inverted repeats of 6 EMALE genomes

Usage

```
emale_tirs
```

Format

A data frame with 3 rows and 14 columns

file_id name of the file the data was read from

seq_id identifier of the sequence the feature appears on

start start of the feature on the sequence

end end of the feature on the sequence

strand reading orientation relative to sequence (+ or -)

type feature type (CDS, mRNA, gene, ...)

feat_id unique identifier of the feature

introns a list column with internal intron start/end positions

parent_ids a list column with parent IDs - feat_id's of parent features

source source of the annotation

score score of the annotation

phase For "CDS" features indicates where the next codon begins relative to the 5' start

name name of the feature

width end-start+1

geom_id an identifier telling the which features should be plotted as on items (usually CDS and mRNA of same gene)

Source

- Publication: doi:10.1101/2020.11.30.404863
- Raw data: https://github.com/thackl/cb-emales
- Derived & bundled data: `ex("emales/emales-tirs.gff")`

ex — *Get path to gggenomes example files*

Description

Get path to gggenomes example files

Usage

```
ex(file = NULL)
```

Arguments

`file`	name of example file

Value

path to example file

feats — *Use tracks inside and outside* `geom_*` *calls*

Description

Track selection works like `dplyr::pull()` and supports unquoted ids and positional arguments. `...` can be used to subset the data in `dplyr::filter()` fashion. `pull`-prefixed variants return the specified track from a gggenomes object. Unprefixed variants work inside `geom_*` calls.

Usage

```
feats(.track_id = 1, ..., .ignore = "genes", .geneify = FALSE)

feats0(.track_id = 1, ..., .ignore = NA, .geneify = FALSE)

genes(..., .gene_types = c("CDS", "mRNA", "tRNA", "tmRNA", "ncRNA", "rRNA"))

links(.track_id = 1, ..., .ignore = NULL, .adjacent_only = NULL)

seqs(...)

bins(..., .group = vars())

track(.track_id = 1, ..., .track_type = NULL, .ignore = NULL)

pull_feats(.x, .track_id = 1, ..., .ignore = "genes", .geneify = FALSE)
```

```
pull_genes(
  .x,
  ...,
  .gene_types = c("CDS", "mRNA", "tRNA", "tmRNA", "ncRNA", "rRNA")
)

pull_links(.x, .track_id = 1, ..., .ignore = NULL, .adjacent_only = NULL)

pull_seqs(.x, ...)

pull_bins(.x, ..., .group = vars())

## S3 method for class 'gggenomes_layout'
pull_bins(.x, ..., .group = vars())

pull_track(.x, .track_id = 1, ..., .track_type = NULL, .ignore = NULL)
```

Arguments

`.track_id`	The track to pull out, either as a literal variable name or as a positive/negative integer giving the position from the left/right.
`...`	Logical predicates passed on to dplyr::filter. "seqs", "feats", "links". Affects position-based selection.
`.ignore`	track names to ignore when selecting by position. Default is "genes", if using `feats0` this defaults to `NA`.
`.geneify`	add dummy type, introns and geom_id column to play nicely with geoms supporting multi-level and spliced gene models.
`.gene_types`	return only feats of this type (`type %in% .gene_types`)
`.adjacent_only`	filter for links connecting direct neighbors (`abs(y-yend)==1`))
`.group`	what variables to use in grouping of bins from seqs in addition to `y` and `bin_id`. Use this to get additional shared variables from the seqs table into the bins table.
`.track_type`	restrict to these types of tracks - any combination of "seqs", "feats", "links".
`.x`	A gggenomes or gggenomes_layout object.

Value

A function that pulls the specified track from a gggenomes object.

A function that pulls the specified track from a gggenomes object.

A function that pulls the specified track from a gggenomes object.

A function that pulls the specified track from a gggenomes object.

A function that pulls the specified track from a gggenomes object.

A function that pulls the specified track from a gggenomes object.

A function that pulls the specified track from a gggenomes object.

Functions

- `feats()`: by default pulls out the first feat track not named "genes".
- `feats0()`: by default pulls out the first feat track.
- `genes()`: pulls out the first feat track (genes), filtering for records with `type=="CDS"`, and adding a dummy `gene_id` column if missing to play nice with multi-exon `geoms`.
- `links()`: by default pulls out the first link track.
- `seqs()`: pulls out the seqs track (there is only one).
- `bins()`: pulls out a binwise summary table of the seqs data powering `geom_bin_*()` calls. The bin table is not a real track, but recomputed on-the-fly.
- `track()`: pulls from all tracks in order seqs, feats, links.

Examples

```
gg <- gggenomes(emale_genes, emale_seqs, emale_tirs, emale_ava)
gg %>% track_info() # info about track ids, positions and types

# get first feat track that isn't "genes" (all equivalent)
gg %>% pull_feats() # easiest
gg %>% pull_feats(feats) # by id
gg %>% pull_feats(1) # by position
gg %>% pull_feats(2, .ignore = NULL) # default .ignore="genes"

# get "seqs" track (always track #1)
gg %>% pull_seqs()

# plot integrated transposons and GC content for some viral genomes
gg <- gggenomes(seqs = emale_seqs, feats = list(emale_ngaros, GC = emale_gc))
gg + geom_seq() +
  geom_feat(color = "skyblue") + # defaults to data=feats()
  geom_line(aes(x, y + score - .6, group = y), data = feats(GC), color = "gray60")
```

---

flip — *Flip bins and sequences*

Description

`flip` and `flip_seqs` reverse-complement specified bins or individual sequences and their features. `sync` automatically flips bins using a heuristic that maximizes the amount of forward strand links between neighboring bins.

Usage

```
flip(x, ..., .bin_track = seqs)

flip_seqs(x, ..., .bins = everything(), .seq_track = seqs, .bin_track = seqs)

sync(x, link_track = 1, min_support = 0)
```

Arguments

x	a gggenomes object
...	bins or sequences to flip in dplyr::select like syntax (numeric position or unquoted expressions)
.bin_track, .seq_track	when using a function as selector such as tidyselect::where(), this specifies the track in which context the function is evaluated.
.bins	preselection of bins with sequences to flip. Useful if selecting by numeric position. It sets the context for selection, for example the 11th sequences of the total set might more easily described as the 2nd sequences of the 3rd bin: flip_seqs(2, .bins=3).
link_track	the link track to use for flipping bins nicely
min_support	only flip a bin if at least this many more nucleotides support an inversion over the given orientation

Details

For more details see the help vignette: vignette("flip", package = "gggenomes")

Value

a gggenomes object with flipped bins or sequences

Examples

```
library(patchwork)
p <- gggenomes(genes = emale_genes) +
  geom_seq(aes(color = strand), arrow = TRUE) +
  geom_link(aes(fill = strand)) +
  expand_limits(color = c("-")) +
  labs(caption = "not flipped")

# nothing flipped
p0 <- p %>% add_links(emale_ava)

# flip manually
p1 <- p %>%
  add_links(emale_ava) %>%
  flip(4:6) + labs(caption = "manually")

# flip automatically based on genome-genome links
p2 <- p %>%
  add_links(emale_ava) %>%
  sync() + labs(caption = "genome alignments")

# flip automatically based on protein-protein links
p3 <- p %>%
  add_sublinks(emale_prot_ava) %>%
  sync() + labs(caption = "protein alignments")
```

```
# flip automatically based on genes linked implicitly by belonging
# to the same clusters of orthologs (or any grouping of your choice)
p4 <- p %>%
  add_clusters(emale_cogs) %>%
  sync() + labs(caption = "shared orthologs")

p0 + p1 + p2 + p3 + p4 + plot_layout(nrow = 1, guides = "collect")
```

`flip_strand` *Flip strand*

Description

Flip strand

Usage

```
flip_strand(strand, na = NA)
```

Arguments

`strand`	some representation for strandedness
`na`	what to use for `NA`

Value

the strand flipped

`focus` *Show features and regions of interest*

Description

Show loci containing features of interest. Loci can either be provided as predefined regions directly (`loci=`), or are constructed automatically based on pre-selected features (via `...`). Features within `max_dist` are greedily combined into the same locus. `locate()` adds these loci as new track so that they can be easily visualized. `focus()` extracts those loci from their parent sequences making them the new sequence set. These sequences will have their `locus_id` as their new `seq_id`.

Usage

```
focus(
  x,
  ...,
  .track_id = 2,
  .max_dist = 10000,
  .expand = 5000,
  .overhang = c("drop", "trim", "keep"),
  .locus_id = str_glue("{seq_id}_lc{row_number()}"),
  .locus_id_group = seq_id,
  .locus_bin = c("bin", "seq", "locus"),
  .locus_score = n(),
  .locus_filter = TRUE,
  .loci = NULL
)

locate(
  x,
  ...,
  .track_id = 2,
  .max_dist = 10000,
  .expand = 5000,
  .locus_id = str_glue("{seq_id}_lc{row_number()}"),
  .locus_id_group = .data$seq_id,
  .locus_bin = c("bin", "seq", "locus"),
  .locus_score = n(),
  .locus_filter = TRUE,
  .locus_track = "loci"
)
```

Arguments

`x`	A gggenomes object
`...`	Logical predicates defined in terms of the variables in the track given by `.track_id`. Multiple conditions are combined with '&'. Only rows where the condition evaluates to 'TRUE' are kept. The arguments in '...' are automatically quoted and evaluated in the context of the data frame. They support unquoting and splicing. See 'vignette("programming")' for an introduction to these concepts.
`.track_id`	the track to filter from - defaults to first feature track, usually "genes". Can be a quoted or unquoted string or a positional argument giving the index of a track among all tracks (seqs, feats & links).
`.max_dist`	Maximum distance between adjacent features to be included into the same locus, default 10kb.
`.expand`	The amount to nucleotides to expand the focus around the target features. Default 2kb. Give two values for different up- and downstream expansions.

`.overhang` How to handle features overlapping the locus boundaries (including expand). Options are to "keep" them, "trim" them exactly at the boundaries, or "drop" all features not fully included within the boundaries.

`.locus_id`, `.locus_id_group` How to generate the ids for the new loci which will eventually become their new `seq_ids`.

`.locus_bin` What bin to assign new locus to. Defaults to keeping the original binning, but can be set to the "seq" to bin all loci originating from the same parent sequence, or to "locus" to separate all loci into individual bins.

`.locus_score` An expression evaluated in the context of all features that are combined into a new locus. Results are stored in the column `locus_score`. Defaults to the `n()`, i.e. the number of features per locus. Set, for example, to `sum(bitscore)` to sum over all blast hit bitscore of per locus. Usually used in conjunction with `.locus_filter`.

`.locus_filter` An predicate expression used to post-filter identified loci. Set `.locus_filter=locus_score >= 3` to only return loci comprising at least 3 target features.

`.loci` A data.frame specifying loci directly. Required columns are `seq_id,start,end`. Supersedes `....`

`.locus_track` The name of the new track containing the identified loci.

Value

A gggenomes object focused on the desired loci

A gggenomes object with the new loci track added

Functions

- `focus()`: Identify regions of interest and zoom in on them
- `locate()`: Identify regions of interest and add them as new feature track

Examples

```
# Let's hunt some defense systems in marine SAGs
# read the genomes
s0 <- read_seqs(ex("gorg/gorg.fna.fai"))
s1 <- s0 %>%
  # strip trailing number from contigs to get bins
  dplyr::mutate(bin_id = stringr::str_remove(seq_id, "_\\d+$"))
# gene annotations from prokka
g0 <- read_feats(ex("gorg/gorg.gff.xz"))

# best hits to the PADS Arsenal database of prokaryotic defense-system genes
# $ mmseqs easy-search gorg.fna pads-arsenal-v1-prf gorg-pads-defense.o6 /tmp \
#     --greedy-best-hits
f0 <- read_feats(ex("gorg/gorg-pads-defense.o6"))
f1 <- f0 %>%
  # parser system/gene info
  tidyr::separate(seq_id2, into = c("seq_id2", "system", "gene"), sep = ",") %>%
```

```
  dplyr::filter(
    evalue < 1e-10, # get rid of some spurious hits
    # and let's focus just on a few systems for this example
    system %in% c("CRISPR-CAS", "DISARM", "GABIJA", "LAMASSU", "THOERIS")
  )

# plot the distribution of hits across full genomes
gggenomes(g0, s1, f1, wrap = 2e5) +
  geom_seq() + geom_bin_label() +
  scale_color_brewer(palette = "Dark2") +
  geom_point(aes(x = x, y = y, color = system), data = feats())

# hilight the regions containing hits
gggenomes(g0, s1, f1, wrap = 2e5) %>%
  locate(.track_id = feats) %>%
  identity() +
  geom_seq() + geom_bin_label() +
  scale_color_brewer(palette = "Dark2") +
  geom_feat(data = feats(loci), color = "plum3") +
  geom_point(aes(x = x, y = y, color = system), data = feats())

# zoom in on loci
gggenomes(g0, s1, f1, wrap = 5e4) %>%
  focus(.track_id = feats) +
  geom_seq() + geom_bin_label() +
  geom_gene() +
  geom_feat(aes(color = system)) +
  geom_feat_tag(aes(label = gene)) +
  scale_color_brewer(palette = "Dark2")
```

`GeomFeatText` *Geom for feature text*

Description

Geom for feature text

Usage

```
GeomFeatText
```

Format

An object of class `GeomFeatText` (inherits from `Geom`, `ggproto`, gg) of length 6.

geom_bin_label *Draw bin labels*

Description

Put bin labels left of the sequences. `nudge_left` adds space relative to the total bin width between the label and the seqs, by default 5%. `expand_left` expands the plot to the left by 20% to make labels visible.

Usage

```
geom_bin_label(
  mapping = NULL,
  data = bins(),
  hjust = 1,
  size = 3,
  nudge_left = 0.05,
  expand_left = 0.2,
  expand_x = NULL,
  expand_aes = NULL,
  yjust = 0,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`hjust`	Moves the text horizontally
`size`	of the label
`nudge_left`	by this much relative to the widest bin
`expand_left`	by this much relative to the widest bin
`expand_x`	expand the plot to include this absolute x value
`expand_aes`	provide custom aes mappings for the expansion (advanced)

`yjust` for multiline bins set to 0.5 to center labels on bins, and 1 to align labels to the bottom.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Details

Set `x` and `expand_x` to an absolute position to align all labels at a specific location

Value

Bin labels are added as a text layer/component to the plot.

Examples

```
s0 <- read_seqs(list.files(ex("cafeteria"), "Cr.*\\.fa.fai$", full.names = TRUE))
s1 <- s0 %>% dplyr::filter(length > 5e5)

gggenomes(emale_genes) + geom_seq() + geom_gene() +
  geom_bin_label()

# make larger labels and extra room on the canvas
gggenomes(emale_genes) + geom_seq() + geom_gene() +
  geom_bin_label(size = 7, expand_left = .4)

# align labels for wrapped bins:
# top
gggenomes(seqs = s1, infer_bin_id = file_id, wrap = 5e6) +
  geom_seq() + geom_bin_label() + geom_seq_label()
```

```
# center
gggenomes(seqs = s1, infer_bin_id = file_id, wrap = 5e6) +
  geom_seq() + geom_bin_label(yjust = .5) + geom_seq_label()

# bottom
gggenomes(seqs = s1, infer_bin_id = file_id, wrap = 5e6) +
  geom_seq() + geom_bin_label(yjust = 1) + geom_seq_label()
```

---

geom_coverage *Draw wiggle ribbons or lines*

Description

Visualize data that varies along sequences as ribbons, lines, lineranges, etc.

Usage

```
geom_coverage(
  mapping = NULL,
  data = feats(),
  stat = "coverage",
  geom = "ribbon",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  offset = 0,
  height = 0.2,
  max = base::max,
  ...
)

geom_wiggle(
  mapping = NULL,
  data = feats(),
  stat = "wiggle",
  geom = "ribbon",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  offset = 0,
  height = 0.8,
  bounds = Hmisc::smedian.hilow,
  ...
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`geom` The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

offset — distance between seq center and wiggle mid/start.

height — distance in plot between lowest and highest point of the wiggle data.

max — geom_coverage uses the function base::max by default, which plots data in positive direction. (base::min Can also be called here when the input data)

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

bounds — geom_wiggle uses mid, low and high boundary values for plotting wiggle data. Can be both a function or a vector returning those three values. Defaults to Hmisc::smedian.hilow.

Details

Geom_wiggle plots the wiggle data in both directions around the median. Geom_coverage plots the data only in positive direction. Both functions use data from the feats' track.

Value

A ggplot2 layer with coverage information.

Aesthetics

geom_wiggle() and geom_coverage() understand aesthetics depending on the chosen underlying ggplot geom, by default ggplot2::geom_ribbon(). Other options that play well are for example ggplot2::geom_line(), ggplot2::geom_linerange(), ggplot2::geom_point(). The only required aesthetic is:

- **z**

Examples

```
# Plotting data with geom_coverage with increased height.
gggenomes(seqs = emale_seqs, feats = emale_gc) +
  geom_coverage(aes(z = score), height = 0.5) +
  geom_seq()

# In opposite direction by calling base::min and taking the negative values of "score"
gggenomes(seqs = emale_seqs, feats = emale_gc) +
  geom_coverage(aes(z = -score), max = base::min, height = 0.5) +
  geom_seq()

# GC-content plotted as points with variable color in geom_coverage
gggenomes(seqs = emale_seqs, feats = emale_gc) +
  geom_coverage(aes(z = score, color = score), height = 0.5, geom = "point") +
  geom_seq()

# wiggle's default bounds function requires Hmisc
if (requireNamespace("Hmisc", quietly = TRUE)) {

# Plot varying GC-content along sequences as ribbon
gggenomes(seqs = emale_seqs, feats = emale_gc) +
  geom_wiggle(aes(z = score)) +
  geom_seq()

# customize color and position
gggenomes(genes = emale_genes, seqs = emale_seqs, feats = emale_gc) +
  geom_wiggle(aes(z = score), fill = "lavenderblush3", offset = -.3, height = .5) +
  geom_seq() + geom_gene()

# GC-content as line and with variable color
gggenomes(seqs = emale_seqs, feats = emale_gc) +
  geom_wiggle(aes(z = score, color = score), geom = "line", bounds = c(.5, 0, 1)) +
  geom_seq() +
  scale_colour_viridis_b(option = "A")

# or as lineranges
gggenomes(seqs = emale_seqs, feats = emale_gc) +
  geom_wiggle(aes(z = score, color = score), geom = "linerange") +
  geom_seq() +
  scale_colour_viridis_b(option = "A")

}
```

geom_feat *Draw feats*

Description

geom_feat() allows the user to draw (additional) features to the plot/graph. For example, specific regions within a sequence (e.g. transposons, introns, mutation hotspots) can be highlighted by color, size, etc..

Usage

```
geom_feat(
  mapping = NULL,
  data = feats(),
  stat = "identity",
  position = "pile",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping — Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

data — feat_layout: Uses first data frame stored in the feats track by default.

stat — The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following:

- A Stat ggproto subclass, for example StatCount.
- A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count".
- For more information and other ways to specify the stat, see the layer stat documentation.

position — describes how the position of different plotted features are adjusted. By default it uses "pile", but different ggplot2 position adjustments, such as "identity or "jitter" can be used as well.

na.rm — If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Details

geom_feat uses ggplot2::geom_segment under the hood. As a result, different aesthetics such as *alpha*, *linewidth*, *color*, etc. can be called upon to modify the visualization of the data.

By default, the function uses the first feature track.

Value

A ggplot2 layer with features.

Examples

```
# Plotting data from the feats' track with adjusted linewidth and color
gggenomes(seqs = emale_seqs, feats = emale_ngaros) +
  geom_seq() +
  geom_feat(linewidth = 5, color = "darkred")
```

```
# Geom_feat can be called several times as well, when specified what data should be used
gggenomes(seqs = emale_seqs, feats = list(emale_ngaros, emale_tirs)) +
  geom_seq() +
  geom_feat(linewidth = 5, color = "darkred") + # uses first feature track
  geom_feat(data = feats(emale_tirs))

# Additional notes to feats can be added with functions such as: geom_feat_note / geom_feat_text
gggenomes(seqs = emale_seqs, feats = list(emale_ngaros, emale_tirs)) +
  geom_seq() +
  geom_feat(color = "darkred") +
  geom_feat(data = feats(emale_tirs), color = "darkblue") +
  geom_feat_note(data = feats(emale_ngaros), label = "repeat region", size = 4)

# Different position adjustments with a simple dataset
exampledata <- tibble::tibble(
  seq_id = c(rep("A", 3), rep("B", 3), rep("C", 3)),
  start = c(0, 30, 15, 40, 80, 20, 30, 50, 70),
  end = c(30, 90, 60, 60, 100, 80, 60, 90, 120)
)

gggenomes(feats = exampledata) +
  geom_feat(position = "identity", alpha = 0.5, linewidth = 0.5) +
  geom_bin_label()
```

`geom_feat_text` *Add text to genes, features, etc.*

Description

The functions below are useful for labeling features/genes in plots. Users have to call on `aes(label = ...)` or `(label = ...)` to define label's text Based on the function, the label will be placed at a specific location:

- `geom_..._text()` will plot **text in the middle of the feature**.
- `geom_..._tag()` will plot **text on top of the feature, with a 45 degree angle**.
- `geom_..._note()` will plot **text under the feature at the left side**.

The ... can be either replaced with `feat` *or* `gene` *depending on which track the user wants to label.*

With arguments such as `hjust`, `vjust`, `angle`, and `nudge_y`, the user can also manually change the position of the text.

Usage

```
geom_feat_text(
  mapping = NULL,
  data = feats(),
  stat = "identity",
  position = "identity",
```

```
    ...,
    parse = FALSE,
    check_overlap = FALSE,
    na.rm = FALSE,
    show.legend = NA,
    inherit.aes = TRUE
  )

  geom_feat_tag(
    mapping = NULL,
    data = feats(),
    stat = "identity",
    position = "identity",
    hjust = 0,
    vjust = 0,
    angle = 45,
    nudge_y = 0.03,
    xjust = 0.5,
    strandwise = TRUE,
    ...,
    parse = FALSE,
    check_overlap = FALSE,
    na.rm = FALSE,
    show.legend = NA,
    inherit.aes = TRUE
  )

  geom_feat_note(
    mapping = NULL,
    data = feats(),
    stat = "identity",
    position = "identity",
    hjust = 0,
    vjust = 1,
    nudge_y = -0.03,
    xjust = 0,
    strandwise = FALSE,
    ...,
    parse = FALSE,
    check_overlap = FALSE,
    na.rm = FALSE,
    show.legend = NA,
    inherit.aes = TRUE
  )

  geom_gene_text(
    mapping = NULL,
    data = genes(),
```

```
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_gene_tag(
  mapping = NULL,
  data = genes(),
  stat = "identity",
  position = "identity",
  hjust = 0,
  vjust = 0,
  angle = 45,
  nudge_y = 0.03,
  xjust = 0.5,
  strandwise = TRUE,
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_gene_note(
  mapping = NULL,
  data = genes(),
  stat = "identity",
  position = "identity",
  hjust = 0,
  vjust = 1,
  nudge_y = -0.03,
  xjust = 0,
  strandwise = FALSE,
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`parse` If `TRUE`, the labels will be parsed into expressions and displayed as described in `?plotmath`.

`check_overlap` If `TRUE`, text that overlaps previous text in the same layer will not be plotted. `check_overlap` happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

`hjust` Moves the text horizontally

`vjust` Moves the text vertically

`angle` Defines the angle in which the text will be placed. *Note

`nudge_y` Moves the text vertically an entire contig/sequence. (e.g. `nudge_y = 1` places the text to the contig above)

`xjust` Move text in x direction

`strandwise` plotting of feature tags

Details

These labeling functions use `ggplot2::geom_text()` under the hood. Any changes to the aesthetics of the text can be performed in a ggplot2 manner.

Value

A ggplot2 layer with gene text.

A ggplot2 layer with feature tags.

A ggplot2 layer with feature notes.

A ggplot2 layer with gene text.

A ggplot2 layer with gene tags.

A ggplot2 layer with gene notes.

Examples

```
# example data
genes <- tibble::tibble(
  seq_id = c("A", "A", "A", "B", "B", "C"),
  start = c(20, 40, 80, 30, 10, 60),
  end = c(30, 70, 85, 40, 15, 90),
  feat_id = c("A1", "A2", "A3", "B1", "B2", "C1"),
  type = c("CDS", "CDS", "CDS", "CDS", "CDS", "CDS"),
  name = c("geneA", "geneB", "geneC", "geneA", "geneC", "geneB")
)

seqs <- tibble::tibble(
  seq_id = c("A", "B", "C"),
  start = c(0, 0, 0),
  end = c(100, 100, 100),
  length = c(100, 100, 100)
)

# basic plot creation
plot <- gggenomes(seqs = seqs, genes = genes) +
  geom_bin_label() +
  geom_gene()

# geom_..._text
plot + geom_gene_text(aes(label = name))

# geom_..._tag
plot + geom_gene_tag(aes(label = name))

# geom_..._note
plot + geom_gene_note(aes(label = name))

# with horizontal adjustment (`hjust`), vertical adjustment (`vjust`)
plot + geom_gene_text(aes(label = name), vjust = -2, hjust = 1)

# using `nudge_y` and and `angle` adjustment
plot + geom_gene_text(aes(label = name), nudge_y = 1, angle = 10)

# labeling with manual input
plot + geom_gene_text(label = c("This", "is", "an", "example", "test", "test"))
```

---

geom_gene *Draw gene models*

Description

Draw coding sequences, mRNAs and other non-coding features. Supports multi-exon features. CDS and mRNAs in the same group are plotted together. They can therefore also be positioned as a single unit using the `position` argument.

Usage

```
geom_gene(
  mapping = NULL,
  data = genes(),
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  size = 2,
  rna_size = size,
  shape = size,
  rna_shape = shape,
  intron_shape = size,
  intron_types = c("CDS", "mRNA", "tRNA", "tmRNA", "ncRNA", "rRNA"),
  cds_aes = NULL,
  rna_aes = NULL,
  intron_aes = NULL,
  ...
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.

- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`na.rm` remove na values

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

`size, rna_size` the size of the gene model, aka the height of the polygons. `rna_size` only applies to non-coding parts of the gene model, defaults to size.

`shape, rna_shape`
vector of height and width of the arrow tip, defaults to size. If only one value is provided it is recycled. Set '0' to deactivates arrow-shaped tips. `rna_shape` only applies to non-coding parts of the gene model, defaults to shape.

`intron_shape` single value controlling the kink of the intron line. Defaults to size. Set 0 for straight lines between exons.

`intron_types` introns will only be computed/drawn for features with types listed here. Set to "CDS" to plot mRNAs as continuous features, and set to NA to completely ignore introns.

`cds_aes, rna_aes, intron_aes`
overwrite aesthetics for different model parts. Need to be wrapped in `ggplot2::aes()`. NOTE: These remappings are applied after the data has been transformed and mapped by the plot scales (see `ggplot2::after_scale()`). So you need to map between aesthetic names (not data columns) and with standardized names, i.e. British English spelling. These mappings can be used to dynamically change parts of the gene model. For example, to change the color of introns from a hard-coded "black" to the same color used to fill the CDS you could specify `intron_aes=aes(colour = fill)`. By default, `rna_aes` is remapped with `aes(fill=colorspace::lighten(fill, .5), colour=colorspace::lighten(colour,`

.5)) to give it a lighter appearance than the corresponding CDS but in the same color.

... passed to layer params

Value

A ggplot2 layer with genes.

Aesthetics

geom_gene() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- xend
- y
- alpha → 1
- colour → "black"
- fill → "cornsilk3"
- group → inferred
- introns → NULL
- linetype → 1
- stroke → 0.4
- type → "CDS"

Learn more about setting these aesthetics in vignette("ggplot2-specs").

'type' and 'group' (mapped to 'type' and 'geom_id' by default) power the proper recognition of CDS and their corresponding mRNAs so that they can be drawn as one composite object. Overwrite 'group' to plot CDS and mRNAs independently.

'introns' (mapped to 'introns') is used to compute intron/exon boundaries. Use the parameter intron_types if you want to disable introns.

Examples

```
gggenomes(genes = emale_genes) +
  geom_gene()

gggenomes(genes = emale_genes) +
  geom_gene(aes(fill = as.numeric(gc_content)), position = "strand") +
  scale_fill_viridis_b()

g0 <- read_gff3(ex("eden-utr.gff"))
gggenomes(genes = g0) +
  # all features in the "genes" regardless of type
  geom_feat(data = feats(genes)) +
  annotate("text", label = "geom_feat", x = -15, y = .9) + xlim(-20, NA) +
  # only features in the "genes" of geneish type (implicit `data=genes()`)
  geom_gene() +
```

```
  geom_gene_tag(aes(label = ifelse(is.na(type), "<NA>", type)), data = genes(.gene_types = NULL)) +
   annotate("text", label = "geom_gene", x = -15, y = 1) +
   # control which types are returned from the track
   geom_gene(aes(y = 1.1), data = genes(.gene_types = c("CDS", "misc_RNA"))) +
   annotate("text", label = "gene_types", x = -15, y = 1.1) +
   # control which types can have introns
   geom_gene(
     aes(y = 1.2, yend = 1.2),
     data = genes(.gene_types = c("CDS", "misc_RNA")),
     intron_types = "misc_RNA"
   ) +
   annotate("text", label = "intron_types", x = -15, y = 1.2)

# spliced genes
library(patchwork)
gg <- gggenomes(genes = g0)
gg + geom_gene(position = "pile") +
  gg + geom_gene(aes(fill = type),
    position = "pile",
    shape = 0, intron_shape = 0, color = "white"
  ) +
  # some fine-control on cds/rna/intron after_scale aesthetics
  gg + geom_gene(aes(fill = geom_id),
    position = "pile",
    size = 2, shape = c(4, 3), rna_size = 2, intron_shape = 4, stroke = 0,
    cds_aes = aes(fill = "black"), rna_aes = aes(fill = fill),
    intron_aes = aes(colour = fill, stroke = 2)
  ) +
  scale_fill_viridis_d() +
  # fun with introns
  gg + geom_gene(aes(fill = geom_id), position = "pile", size = 3, shape = c(4, 4)) +
  gg + geom_gene(aes(fill = geom_id),
    position = "pile", size = 3, shape = c(4, 4),
    intron_types = c()
  ) +
  gg + geom_gene(aes(fill = geom_id),
    position = "pile", size = 3, shape = c(4, 4),
    intron_types = "CDS"
  )
```

`geom_gene_label` *Draw feat/link labels*

Description

These `geom_..._label()` functions able the user to plot labels/text at individual features and/or links. Users have to indicate how to label the features/links by specifying `label = ...` or `aes(label = ...`

Position of labels can be adjusted with arguments such as `vjust`, `hjust`, `angle`, `nudge_y`, etc. Also check out `geom_bin_label()`, `geom_seq_label()` or `geom_feat_text()` given their resemblance.

Usage

```
geom_gene_label(
  mapping = NULL,
  data = genes(),
  angle = 45,
  hjust = 0,
  nudge_y = 0.1,
  size = 6,
  ...
)

geom_feat_label(
  mapping = NULL,
  data = feats(),
  angle = 45,
  hjust = 0,
  nudge_y = 0.1,
  size = 6,
  ...
)

geom_link_label(
  mapping = NULL,
  data = links(),
  angle = 0,
  hjust = 0.5,
  vjust = 0.5,
  size = 4,
  repel = FALSE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
angle	Defines the angle in which the text will be placed. *Note

`hjust` Moves the text horizontally

`nudge_y` Moves the text vertically an entire contig/sequence. (e.g. `nudge_y = 1` places the text to the contig above)

`size` of the label

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`vjust` Moves the text vertically

`repel` use ggrepel to avoid overlaps

Details

These labeling functions use `ggplot2::geom_text()` under the hood. Any changes to the aesthetics of the text can be performed in a ggplot2 manner.

Value

Gene labels are added as a text layer/component to the plot.

`geom_link` *Draw links between genomes*

Description

Draw connections between genomes, such as genome/gene/protein alignments and gene/protein clusters. `geom_link()` and `geom_link_curved()` create filled polygons between regions, `geom_link_line()` a single connecting line.

Note that by default only links between adjacent genomes are computed and shown. To compute and show all links between all genomes, set `gggenomes(..., adjacent_only=FALSE)`.

Usage

```
geom_link(
  mapping = NULL,
  data = links(),
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  offset = 0.15,
  curve = NA,
  ...
)

geom_link_curved(
  mapping = NULL,
  data = links(),
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  offset = 0.15,
  curve = 10,
  ...
)

geom_link_line(
  mapping = NULL,
  data = links(),
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

`offset` distance between seq center and link start. Use two values `c(<offset_top>, <offset_bottom>)` for different top and bottom offsets

curve — curvature of the link. If NA or 0, the link edges will be straight. For curved links, higher values lead to stronger curvature. Typical values are between 5 and 15.

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Details

The function calls upon the data stored within the link track. Data frames added to this track have seq_id and seq_id2 as required variables. Optional and recommended variables include start, start2, end, end2, bin_id, bin_id2 and strand.

Note, when start/end is not specified, links will be created between the entire contigs of seq_id and seq_id2.

Value

A ggplot2 layer with links.

A ggplot2 layer with links.

A ggplot2 layer with links.

Examples

```
p0 <- gggenomes(seqs = emale_seqs, links = emale_ava) + geom_seq()

# default links
p1 <- p0 + geom_link()

# change offset from seqs and color
p2 <- p0 + geom_link(aes(fill = de, color = de), offset = 0.05) +
```

```
  scale_fill_viridis_b() + scale_colour_viridis_b()

# combine with flip
p3 <- p0 |> flip(3, 4, 5) +
  geom_link()

# compute & show all links among all genomes
# usually not useful and not recommended for large dataset
p4 <- gggenomes(links = emale_ava, adjacent_only = FALSE) + geom_link()

library(patchwork) # combine plots in one figure
p1 + p2 + p3 + p4 + plot_layout(nrow = 1)

q0 <- gggenomes(emale_genes, emale_seqs) |>
  add_clusters(emale_cogs) +
  geom_seq() + geom_gene()

qq <-
# link gene clusters with polygon
q0 + geom_link(aes(fill = cluster_id)) +
# link with curved polygons (bezier-like)
q0 + geom_link_curved(aes(fill = cluster_id)) +
# link gene clusters with lines
q0 + geom_link_line(aes(color = cluster_id))

qq + plot_layout(nrow = 1, guides = "collect")
```

`geom_seq` *draw seqs*

Description

`geom_seq()` draws contigs for each sequence/chromosome supplied in the `seqs` track. Several sequences belonging to the same bin will be plotted next to one another.

If `seqs` track is empty, sequences are inferred from the `feats` or `links` track respectively.

(*The length of sequences can be deduced from the axis and is typically indicated in base pairs.*)

Usage

```
geom_seq(mapping = NULL, data = seqs(), arrow = NULL, ...)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	seq_layout: Uses the first data frame stored in the `seqs` track, by default.
`arrow`	set to non-NULL to generate default arrows

... Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through ....Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Details

geom_seq() uses ggplot2::geom_segment() under the hood. As a result, different aesthetics such as *alpha*, *linewidth*, *color*, etc. can be called upon to modify the visualization of the data.

Note: The seqs track indicates the length/region of the sequence/contigs that will be plotted. *Feats* or *links* data that falls outside of this region are ignored!

Value

Sequence data drawn as contigs is added as a layer/component to the plot.

Examples

```
# Simple example of geom_seq
gggenomes(seqs = emale_seqs) +
  geom_seq() + # creates contigs
  geom_bin_label() # labels bins/sequences

# No sequence information supplied, will inform/warn that seqs are inferred from feats.
gggenomes(genes = emale_genes) +
  geom_seq() + # creates contigs
  geom_gene() + # draws genes on top of contigs
  geom_bin_label() # labels bins/sequences

# Sequence data controls what sequences and/or regions will be plotted.
# Here one sequence is filtered out, Notice that the genes of the removed
```

```
# sequence are silently ignored and thus not plotted.
missing_seqs <- emale_seqs |>
  dplyr::filter(seq_id != "Cflag_017B") |>
  dplyr::arrange(seq_id) # `arrange` to restore alphabetical order.

gggenomes(seqs = missing_seqs, genes = emale_genes) +
  geom_seq() + # creates contigs
  geom_gene() + # draws genes on top of contigs
  geom_bin_label() # labels bins/sequences

# Several sequences belonging to the same *bin* are plotted next to one another
seqs <- tibble::tibble(
  bin_id = c("A", "A", "A", "B", "B", "B", "B", "C", "C"),
  seq_id = c("A1", "A2", "A3", "B1", "B2", "B3", "B4", "C1", "C2"),
  start = c(0, 100, 200, 0, 50, 150, 250, 0, 400),
  end = c(100, 200, 400, 50, 100, 250, 300, 300, 500),
  length = c(100, 100, 200, 50, 50, 100, 50, 300, 100)
)

gggenomes(seqs = seqs) +
  geom_seq() +
  geom_bin_label() + # label bins
  geom_seq_label() # label individual sequences

# Wrap bins uptill a certain amount.
gggenomes(seqs = seqs, wrap = 300) +
  geom_seq() +
  geom_bin_label() + # label bins
  geom_seq_label() # label individual sequences

# Change the space between sequences belonging to one bin
gggenomes(seqs = seqs, spacing = 100) +
  geom_seq() +
  geom_bin_label() + # label bins
  geom_seq_label() # label individual sequences
```

`geom_seq_break` *Decorate truncated sequences*

Description

`geom_seq_break()` adds decorations to the ends of truncated sequences. These could arise from zooming onto sequence loci with `focus()`, or manually annotating sequences with start > 1 and/or end < length.

Usage

```
geom_seq_break(
  mapping_start = NULL,
  mapping_end = NULL,
```

```
  data_start = seqs(start > 1),
  data_end = seqs(end < length),
  label = "/",
  size = 4,
  hjust = 0.75,
  family = "sans",
  stat = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping_start	optional start mapping
mapping_end	optional end mapping
data_start	seq_layout of sequences for which to decorate the start. default: `seqs(start >1)`
data_end	seq_layout of sequences for which to decorate the end. default: `seqs(end < length)`
label	the character to decorate ends with. Provide two values for different start and end decorations, e.g. `label=c("]", "[")`.
size	of the text
hjust	Moves the text horizontally
family	font family of the text
stat	The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following: • A `Stat` ggproto subclass, for example `StatCount`. • A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`. • For more information and other ways to specify the stat, see the layer stat documentation.
na.rm	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

... Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Value

A ggplot2 layer with sequence breaks.

Examples

```
# decorate breaks created with focus()
gggenomes(emale_genes, emale_seqs) |>
  focus(.expand = 1e3, .max_dist = 1e3) +
  geom_seq() + geom_gene() +
  geom_seq_break()

# customize decorations
gggenomes(emale_genes, emale_seqs) |>
  focus(.expand = 1e3, .max_dist = 1e3) +
  geom_seq() + geom_gene() +
  geom_seq_break(label = c("[", "]"), size = 3, color = "#1b9e77")

# decorate manually truncated sequences
s0 <- tibble::tribble(
  # start/end define regions, i.e. truncated contigs
  ~bin_id, ~seq_id, ~length, ~start, ~end,
  "complete_genome", "chromosome_1_long_trunc_2side", 1e5, 1e4, 2.1e4,
  "fragmented_assembly", "contig_1_trunc_1side", 1.3e4, .9e4, 1.3e4,
  "fragmented_assembly", "contig_2_short_complete", 0.3e4, 1, 0.3e4,
  "fragmented_assembly", "contig_3_trunc_2sides", 2e4, 1e4, 1.4e4
)
```

```
l0 <- tibble::tribble(
  ~seq_id, ~start, ~end, ~seq_id2, ~start2, ~end2,
  "chromosome_1_long_trunc_2side", 1.1e4, 1.4e4,
  "contig_1_trunc_1side", 1e4, 1.3e4,
  "chromosome_1_long_trunc_2side", 1.4e4, 1.7e4,
  "contig_2_short_complete", 1, 0.3e4,
  "chromosome_1_long_trunc_2side", 1.7e4, 2e4,
  "contig_3_trunc_2sides", 1e4, 1.3e4
)

gggenomes(seqs = s0, links = l0) +
  geom_seq() + geom_link() +
  geom_seq_label(nudge_y = -.05) +
  geom_seq_break()
```

`geom_seq_label` *Draw seq labels*

Description

This function will put labels at each individual sequence. By default it will plot the `seq_id` as label, but users are able to change this manually.

Position of the label/text can be adjusted with the different arguments (e.g. `vjust`, `hjust`, `angle`, etc.)

Usage

```
geom_seq_label(
  mapping = NULL,
  data = seqs(),
  hjust = 0,
  vjust = 1,
  nudge_y = -0.15,
  size = 2.5,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`hjust` Moves the text horizontally

`vjust` Moves the text vertically

`nudge_y` Moves the text vertically an entire contig/sequence. (e.g. `nudge_y = 1` places the text to the contig above)

`size` of the label

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Details

This labeling function uses `ggplot2::geom_text()` under the hood. Any changes to the aesthetics of the text can be performed in a ggplot2 manner.

Value

Sequence labels are added as a text layer/component to the plot.

Examples

```
# example data
seqs <- tibble::tibble(
```

```
  bin_id = c("A", "A", "A", "B", "B", "B", "B", "C", "C"),
  seq_id = c("A1", "A2", "A3", "B1", "B2", "B3", "B4", "C1", "C2"),
  start = c(0, 100, 200, 0, 50, 150, 250, 0, 400),
  end = c(100, 200, 400, 50, 100, 250, 300, 300, 500),
  length = c(100, 100, 200, 50, 50, 100, 50, 300, 100)
)

# example plot using geom_seq_label
gggenomes(seqs = seqs) +
  geom_seq() +
  geom_seq_label()

# changing default label to `length` column
gggenomes(seqs = seqs) +
  geom_seq() +
  geom_seq_label(aes(label = length))

# with horizontal adjustment
gggenomes(seqs = seqs) +
  geom_seq() +
  geom_seq_label(hjust = -5)

# with wrapping at 300
gggenomes(seqs = seqs, wrap = 300) +
  geom_seq() +
  geom_seq_label()
```

`geom_variant` *Draw place of mutation*

Description

geom_variant allows the user to draw points at locations where a mutation has occurred. Data on SNPs, Insertions, Deletions and more (often stored in a variant call format (VCF)) can easily be visualized this way.

Usage

```
geom_variant(
  mapping = NULL,
  data = feats(),
  stat = "identity",
  position = "identity",
  geom = "variant",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  offset = 0,
  ...
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` Data from the first feats track is used for this function by default. When several feats tracks are present within the gggenomes track system, make sure that the wanted data is used by calling `data = feats(*df*)` within the `geom_variant` function.

`stat` Describes what statistical transformation is used for this layer. By default it uses `"identity"`, indicating no statistical transformation.

`position` Describes how the position of different plotted features are adjusted. By default it uses `"identity"`, but different position adjustments, such as `position_variant()`, ggplot2' `"jitter"` or `"pile"` can be used as well.

`geom` Describes what geom is called upon by the function for plotting. By default the function uses `"variant"`, a modified geom_point object. For larger sequences with abundant mutations/variations, it is recommended to use `"ticks"` (a modified geom_point object with different default shape and alpha, which plots the points as small "ticks"), but in theory any other ggplot2 geom can be called here as well.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

`offset` Numeric value describing how far the points will be drawn from the base/sequence. By default it is set on `offset = 0`.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.

- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Details

geom_variant uses `ggplot2::geom_point` under the hood. As a result, different aesthetics such as `alpha`, `size`, `color`, etc. can be called upon to modify the data visualization.

#' the function `gggenomes::read_feats` is able to read VCF files and converts them into a format that is applicable within the gggenomes' track system. *Keep in mind: The function uses data from the feats' track.*

Value

A ggplot2 layer with variant information.

Examples

```
# Creation of example data.
# (Note: These are mere examples and do not fully resemble data from VCF-files)
## Small example data set
f1 <- tibble::tibble(
  seq_id = c(rep(c("A", "B"), 4)), start = c(1, 10, 15, 15, 30, 40, 40, 50),
  end = c(2, 11, 20, 16, 31, 41, 50, 51), length = end - start,
  type = c("SNP", "SNP", "Insertion", "Deletion", "Deletion", "SNP", "Insertion", "SNP"),
  ALT = c("A", "T", "CAT", ".", ".", "G", "GG", "G"),
  REF = c("C", "G", "C", "A", "A", "C", "G", "T")
)
s1 <- tibble::tibble(seq_id = c("A", "B"), start = c(0, 0), end = c(55, 55), length = end - start)

## larger example data set
f2 <- tibble::tibble(
  seq_id = c(rep("A", 667)),
  start = c(
    seq(from = 1, to = 500, by = 2),
    seq(from = 500, to = 2500, by = 50),
    seq(from = 2500, to = 4000, by = 4)
  ),
  end = start + 1, length = end - start,
  type = c(
    rep("SNP", 100),
    rep("Deletion", 20),
    rep("SNP", 180),
    rep("Deletion", 67),
    rep("SNP", 100),
    rep("Insertion", 50),
    rep("SNP", 150)
```

```
  ),
  ALT = c(
    sample(x = c("A", "C", "G", "T"), size = 100, replace = TRUE),
    rep(".", 20), sample(x = c("A", "C", "G", "T"), size = 180, replace = TRUE),
    rep(".", 67), sample(x = c("A", "C", "G", "T"), size = 100, replace = TRUE),
    sample(x = c(
      "AA", "AC", "AG", "AT", "CA", "CC", "CG", "CT", "GA", "GC",
      "GG", "GT", "TA", "TC", "TG", "TT"
    ), size = 50, replace = TRUE),
    sample(x = c("A", "C", "G", "T"), size = 150, replace = TRUE)
  )
)

# Basic example plot with geom_variant
gggenomes(seqs = s1, feats = f1) +
  geom_seq() +
  geom_variant()

# Improving plot elements, by changing shape and adding bin_label
gggenomes(seqs = s1, feats = f1) +
  geom_seq() +
  geom_variant(aes(shape = type), offset = -0.1) +
  scale_shape_variant() +
  geom_bin_label()

# Positional adjustment based on type of mutation: position_variant
gggenomes(seqs = s1, feats = f1) +
  geom_seq() +
  geom_variant(
    aes(shape = type),
    position = position_variant(offset = c(Insertion = -0.2, Deletion = -0.2, SNP = 0))
  ) +
  scale_shape_variant() +
  geom_bin_label()

# Plotting larger example data set with Changing default geom to
# `geom = "ticks"` using positional adjustment based on type (`position_variant`)
gggenomes(feats = f2) +
 geom_variant(aes(color = type), geom = "ticks", alpha = 0.4, position = position_variant()) +
  geom_bin_label()

# Changing geom to `"text"`, to plot ALT nucleotides
gggenomes(seqs = s1, feats = f1) +
  geom_seq() +
  geom_variant(aes(shape = type), offset = -0.1) +
  scale_shape_variant() +
  geom_variant(aes(label = ALT), geom = "text", offset = -0.25) +
  geom_bin_label()
```

`get_seqs` *Get/set the seqs track*

Description

Get/set the seqs track

Usage

```
get_seqs(x)

set_seqs(x, value)
```

Arguments

x	a gggenomes or gggenomes_layout object
value	to set for seqs

Value

a gggenomes_layout track tibble

---

gggenomes *Plot genomes, features and synteny maps*

Description

`gggenomes()` initializes a gggenomes-flavored ggplot object. It is used to declare the input data for gggenomes' track system.

(*See for more details on the track system, gggenomes vignette or the Details/Arguments section*)

Usage

```
gggenomes(
  genes = NULL,
  seqs = NULL,
  feats = NULL,
  links = NULL,
  .id = "file_id",
  spacing = 0.05,
  wrap = NULL,
  adjacent_only = TRUE,
  infer_bin_id = seq_id,
  infer_start = min(start, end),
  infer_end = max(start, end),
  infer_length = max(start, end),
  theme = c("clean", NULL),
  .layout = NULL,
  ...
)
```

Arguments

`genes`, `feats` A data.frame, a list of data.frames, or a character vector with paths to files containing gene data. Each item is added as feature track.

For a single data.frame the track_id will be "genes" and "feats", respectively. For a list, track_ids are parsed from the list names, or if names are missing from the name of the variable containing each data.frame. Data columns:

- required: `seq_id,start,end`
- recognized: `strand,bin_id,feat_id,introns`

`seqs` A data.frame or a character vector with paths to files containing sequence data. Data columns:

- required: `seq_id,length`
- recognized: `bin_id,start,end,strand`

`links` A data.frame or a character vector with paths to files containing link data. Each item is added as links track. Data columns:

- required: `seq_id,seq_id2`
- recognized: `start,end,bin_id,start2,end2,bin_id2,strand`

`.id` The name of the column for file labels that are created when reading directly from files. Defaults to "file_id". Set to "bin_id" if every file represents a different bin.

`spacing` between sequences in bases (>1) or relative to longest bin (<1)

`wrap` wrap bins into multiple lines with at most this many nucleotides per line.

`adjacent_only` Indicates whether links should be created between adjacent sequences/chromosomes only. By default it is set to `adjacent_only = TRUE`. If `FALSE`, links will be created between all sequences

(*not recommended for large data sets*)

`infer_length`, `infer_start`, `infer_end`, `infer_bin_id`

used to infer pseudo seqs if only feats or links are provided, or if no bin_id column was provided. The expressions are evaluated in the context of the first feat or link track.

By default subregions of sequences from the first to the last feat/link are generated. Set `infer_start` to 0 to show all sequences from their true beginning.

`theme` choose a gggenomes default theme, NULL to omit.

`.layout` a pre-computed layout from `layout_genomes()`. Useful for developmental purposes.

`...` additional parameters, passed to layout

Details

`gggenomes::gggenomes()` resembles the functionality of `ggplot2::ggplot()`. It is used to construct the initial plot object, and is often followed by "+" to add components to the plot (*e.g. "+ geom_gene()"*).

A big difference between the two is that gggenomes has a multi-track setup (`'seqs'`, `'feats'`, `'genes'` *and* `'links'`). `gggenomes()` pre-computes a layout and adds coordinates (`y,x,xend`) to each data frame prior to the actual plot construction. This has some implications for the usage of gggenomes:

- **Data frames for tracks have required variables.** These predefined variables are used during import to compute x/y coordinates (*see arguments*).
- **gggenomes' geoms can often be used without explicit** `aes()` **mappings** This works because we always know the names of the plot variables ahead of time: they originate from the pre-computed layout, and we can use that information to set sensible default aesthetic mappings for most cases.

Value

gggenomes-flavored ggplot object

Examples

```
# Compare the genomic organization of three viral elements
# EMALEs: endogenous mavirus-like elements (example data shipped with gggenomes)
gggenomes(emale_genes, emale_seqs, emale_tirs, emale_ava) +
  geom_seq() + geom_bin_label() + # chromosomes and labels
  geom_feat(linewidth= 8) + # terminal inverted repeats
  geom_gene(aes(fill = strand), position = "strand") + # genes
  geom_link(offset = 0.15) # synteny-blocks

# with some more information
gggenomes(emale_genes, emale_seqs, emale_tirs, emale_ava) %>%
  add_feats(emale_ngaros, emale_gc) %>%
  add_clusters(emale_cogs) %>%
  sync() +
  geom_link(offset = 0.15, color = "white") + # synteny-blocks
  geom_seq() + geom_bin_label() + # chromosomes and labels
  # thistle4, salmon4, burlywood4
  geom_feat(linewidth= 6, position = "identity") + # terminal inverted repeats
  geom_feat(
    data = feats(emale_ngaros), color = "turquoise4", alpha = .3,
    position = "strand", linewidth = 16
  ) +
  geom_feat_note(aes(label = type),
    data = feats(emale_ngaros),
    position = "strand", nudge_y = .3
  ) +
  geom_gene(aes(fill = cluster_id), position = "strand") + # genes
  geom_wiggle(aes(z = score, linetype = "GC-content"), feats(emale_gc),
    fill = "lavenderblush4", position = position_nudge(y = -.2), height = .2
  ) +
  scale_fill_brewer("Conserved genes", palette = "Dark2", na.value = "cornsilk3")

# initialize plot directly from files
gggenomes(
  ex("emales/emales.gff"),
  ex("emales/emales.gff"),
  ex("emales/emales-tirs.gff"),
  ex("emales/emales.paf")
) + geom_seq() + geom_gene() + geom_feat() + geom_link()
```

```
# multi-contig genomes wrap to fixed width
s0 <- read_seqs(list.files(ex("cafeteria"), "Cr.*\\.fa.fai$", full.names = TRUE))
s1 <- s0 %>% dplyr::filter(length > 5e5)
gggenomes(seqs = s1, infer_bin_id = file_id, wrap = 5e6) +
  geom_seq() + geom_bin_label() + geom_seq_label()
```

---

if_reverse *Vectorised if_else based on strandedness*

Description

Vectorised if_else based on strandedness

Usage

```
if_reverse(strand, reverse, forward)
```

Arguments

strand	vector with strandedness information
reverse	value to use for reverse elements
forward	value to use for forward elements

Value

vector with values based on strandedness

---

introduce *Introduce non-existing columns*

Description

Works like `dplyr::mutate()` but without changing existing columns, but only adding new ones. Useful to add possibly missing columns with default values.

Usage

```
introduce(.data, ...)
```

Arguments

`.data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.
`...`	<`data-masking`> Name-value pairs. The name gives the name of the column in the output. The value can be: • A vector of length 1, which will be recycled to the correct length. • A vector the same length as the current group (or the whole data frame if ungrouped). • `NULL`, to remove the column. • A data frame or tibble, to create multiple columns in the output.

Value

a tibble with new columns

Examples

```
# ensure columns "y" and "z" exist
tibble::tibble(x = 1:3) %>%
  introduce(y = "a", z = paste0(y, dplyr::row_number()))
# ensure columns "y" and "z" exist, but do not overwrite "y"
tibble::tibble(x = 1:3, y = c("c", "d", "e")) %>%
  introduce(y = "a", z = paste0(y, dplyr::row_number()))
```

`in_range` *Do numeric values fall into specified ranges?*

Description

Do numeric values fall into specified ranges?

Usage

```
in_range(x, left, right, closed = TRUE)
```

Arguments

`x`	a numeric vector of values
`left, right`	boundary values or vectors of same length as x
`closed`	whether to include (`TRUE`) or exclude (`FALSE`) the endpoints. Provide 2 values for different behaviors for lower and upper boundary, e.g. `c(TRUE, FALSE)` to include only the lower boundary.

Value

a logical vector of the same length as the input

Examples

```
in_range(1:5, 2, 4)
in_range(1:5, 2, 4, closed = c(FALSE, TRUE)) # left-open
in_range(1:5, 6:2, 3) # vector of boundaries, single values recycle


# plays nicely with dplyr
df <- tibble::tibble(x = rep(4, 5), left = 1:5, right = 3:7)
dplyr::mutate(df,
  closed = in_range(x, left, right, TRUE),
  open = in_range(x, left, right, FALSE)
)
```

`is_reverse` *Check whether strand is reverse*

Description

Check whether strand is reverse

Usage

```
is_reverse(strand, na = FALSE)
```

Arguments

`strand`	some representation for strandedness
`na`	what to use for `NA`

Value

logical vector indicating whether the strand is reverse

`layout` *Re-layout a genome layout*

Description

Re-layout the tracks and update the scales after seqs have been modified

Usage

```
layout(x, ...)
```

Arguments

x	layout
...	additional data

Value

layout with updated scales

---

layout_seqs *Layout sequences*

Description

Layout sequences

Usage

```
layout_seqs(
  x,
  spacing = 0.05,
  wrap = NULL,
  spacing_style = c("regular", "center", "spread"),
  keep = "strand"
)
```

Arguments

x	seq_layout
spacing	between sequences in bases (>1) or relative to longest bin (<1)
wrap	wrap bins into multiple lines with at most this many nucleotides per line.
spacing_style	one of "regular", "center", "spread"
keep	keys to keep (default: "strand")

Value

a tbl_df with plot coordinates

pick *Pick bins and seqs by name or position*

Description

Pick which bins and seqs to show and in what order. Uses `dplyr::select()`-like syntax, which means unquoted genome names, positional arguments and selection helpers, such as `tidyselect::starts_with()` are supported. Renaming is not supported.

Usage

```
pick(x, ...)

pick_seqs(x, ..., .bins = everything())

pick_seqs_within(x, ..., .bins = everything())

pick_by_tree(x, tree, infer_bin_id = .data$label)
```

Arguments

`x`	gggenomes object
`...`	bins/seqs to pick, select-like expression.
`.bins`	scope for positional arguments, select-like expression, enclose multiple arguments with `c()`!
`tree`	a phylogenetic tree in ggtree::ggtree or `ape::ape-package`-"phylo" format.
`infer_bin_id`	an expression to extract bin_ids from the tree data.

Details

Use the dots to select bins or sequences (depending on function suffix), and the `.bins` argument to set the scope for positional arguments. For example, `pick_seqs(1)` will pick the first sequence from the first bin, while `pick_seqs(1, .bins=3)` will pick the first sequence from the third bin.

Value

gggenomes object with selected bins and seqs.

gggenomes object with selected seqs.

gggenomes object with selected seqs.

gggenomes object with seqs selected by tree order.

Functions

- `pick()`: pick bins by bin_id, positional argument (start at top) or select-helper.
- `pick_seqs()`: pick individual seqs seq_id, positional argument (start at top left) or select-helper.
- `pick_seqs_within()`: pick individual seqs but only modify bins containing those seqs, keep rest as is.
- `pick_by_tree()`: align bins with the leaves in a given phylogenetic tree.

Examples

```
s0 <- tibble::tibble(
  bin_id = c("A", "B", "B", "B", "C", "C", "C"),
  seq_id = c("a1", "b1", "b2", "b3", "c1", "c2", "c3"),
  length = c(1e4, 6e3, 2e3, 1e3, 3e3, 3e3, 3e3)
)

p <- gggenomes(seqs = s0) + geom_seq(aes(color = bin_id), linewidth = 3) +
  geom_bin_label() + geom_seq_label() +
  expand_limits(color = c("A", "B", "C"))
p

# remove
p %>% pick(-B)

# select and reorder, by ID and position
p %>% pick(C, 1)

# use helper function
p %>% pick(starts_with("B"))

# pick just some seqs
p %>% pick_seqs(1, c3)

# pick with .bin scope
p %>% pick_seqs(3:1, .bins = C)

# change seqs in some bins, but keep rest as is
p %>% pick_seqs_within(3:1, .bins = B)

# same w/o scope, unaffected bins remain as is
p %>% pick_seqs_within(b3, b2, b1)

try({ # can fail on older systems with older ggtree versions

# Align sequences with and plot next to a phylogenetic tree
library(patchwork) # arrange multiple plots
library(ggtree) # plot phylogenetic trees

# load and plot a phylogenetic tree
emale_mcp_tree <- read.tree(ex("emales/emales-MCP.nwk"))
t <- ggtree(emale_mcp_tree) + geom_tiplab(align = TRUE, size = 3) +
```

```
  xlim(0, 0.05) # make room for labels

p <- gggenomes(seqs = emale_seqs, genes = emale_genes) +
  geom_seq() + geom_seq() + geom_bin_label()

# plot next to each other, but with
# different order in tree and genomes
t + p + plot_layout(widths = c(1, 5))

# reorder genomes to match tree order
# with a warning caused by mismatch in y-scale expansions
t + p %>% pick_by_tree(t) + plot_layout(widths = c(1, 5))

# extra genomes are dropped with a notification
emale_seqs_more <- emale_seqs
emale_seqs_more[7, ] <- emale_seqs_more[6, ]
emale_seqs_more$seq_id[7] <- "One more genome"
p <- gggenomes(seqs = emale_seqs_more, genes = emale_genes) +
  geom_seq() + geom_seq() + geom_bin_label()
t + p %>% pick_by_tree(t) + plot_layout(widths = c(1, 5))

# no shared ids will cause an error
p <- gggenomes(seqs = tibble::tibble(seq_id = "foo", length = 1)) +
  geom_seq() + geom_seq() + geom_bin_label()
t + p %>% pick_by_tree(t) + plot_layout(widths = c(1, 5))

# extra leafs in tree will cause an error
emale_seqs_fewer <- slice_head(emale_seqs, n = 4)
p <- gggenomes(seqs = emale_seqs_fewer, genes = emale_genes) +
  geom_seq() + geom_seq() + geom_bin_label()
t + p %>% pick_by_tree(t) + plot_layout(widths = c(1, 5))

})
```

`position_strand` *Stack features*

Description

`position_strand()` offsets forward feats upward and reverse feats downward. `position_pile()` stacks overlapping feats upward. `position_strandpile()` stacks overlapping feats up-/downward based on their strand. `position_sixframe()` offsets the feats based on their strand and reading frame.

Usage

```
position_strand(offset = 0.1, flip = FALSE, grouped = NULL, base = offset/2)

position_pile(offset = 0.1, gap = 1, flip = FALSE, grouped = NULL, base = 0)
```

```
position_strandpile(
  offset = 0.1,
  gap = 1,
  flip = FALSE,
  grouped = NULL,
  base = offset * 1.5
)

position_sixframe(offset = 0.1, flip = FALSE, grouped = NULL, base = offset/2)
```

Arguments

offset	Shift overlapping feats up/down this much on the y-axis. The y-axis distance between two sequences is 1, so this is usually a small fraction, such as 0.1.
flip	stack downward, and for stranded versions reverse upward.
grouped	if TRUE feats in the same group are stacked as a single feature. Useful to move CDS and mRNA as one unit. If NULL (default) set to TRUE if data appears to contain gene-ish features.
base	How to align the stack relative to the sequence. 0 to center the lowest stack level on the sequence, 1 to put forward/reverse sequence one half offset above/below the sequence line.
gap	If two feats are closer together than this, they will be stacked. Can be negative to allow small overlaps. NA disables stacking.

Value

A ggproto object to be used in geom_gene().

Examples

```
library(patchwork)
p <- gggenomes(emale_genes) %>%
  pick(3:4) + geom_seq()

f0 <- tibble::tibble(
  seq_id = pull_seqs(p)$seq_id[1],
  start = 1:20 * 1000,
  end = start + 2500,
  strand = rep(c("+", "-"), length(start) / 2)
)

sixframe <- function(x, strand) as.character((x %% 3 + 1) * strand_int(strand))

p1 <- p + geom_gene()
p2 <- p + geom_gene(aes(fill = strand), position = "strand")
p3 <- p + geom_gene(aes(fill = strand), position = position_strand(flip = TRUE, base = 0.2))
p4 <- p + geom_gene(aes(fill = sixframe(x, strand)), position = "sixframe")
p5 <- p %>% add_feats(f0) + geom_gene() + geom_feat(aes(color = strand))
p6 <- p %>% add_feats(f0) + geom_gene() + geom_feat(aes(color = strand), position = "strandpile")
```

```
p1 + p2 + p3 + p4 + p5 + p6 + plot_layout(ncol = 3, guides = "collect") & ylim(2.5, 0.5)
```

---

position_variant *Plot types of mutations with different offsets*

Description

position_variant() allows the user to plot the different mutation types (e.g. del, ins, snps) at different offsets from the base. This can especially be useful to highlight in which regions certain types of mutations have higher prevalence. This position adjustment is most relevant for the analysis/visualization of VCF files with the function geom_variant().

Usage

```
position_variant(offset = c(del = 0.1, snp = 0, ins = -0.1), base = 0)
```

Arguments

offset	Shifts the data up/down based on the type of mutation. By default offset = c(del=0.1, snp=0, ins=-0.1). The user can supply an own vector to offset to indicate at which offsets the different mutation types should be plotted. *Types of mutations that have not been specified within the vector, will be plotted with an offset of 0.*
base	How to align the offsets relative to the sequence. At base = 0, plotting of the offsets starts from the sequence. base thus moves the entire feature up/down.

Value

A ggproto object to be used in geom_variant().

Examples

```
# Creation of example data.
testposition <- tibble::tibble(
  type = c("ins", "snp", "snp", "del", "del", "snp", "snp", "ins", "snp", "ins", "snp"),
  start = c(10, 20, 30, 35, 40, 60, 65, 90, 90, 100, 120),
  end = start + 1,
  seq_id = c(rep("A", 11))
)
testseq <- tibble::tibble(
  seq_id = "A",
  start = 0,
  end = 150,
  length = end - start
)

p <- gggenomes(seqs = testseq, feats = testposition)
```

```
# This first plot shows what is being plotted when only geom_variant is called
p + geom_variant()

# Next lets use position_variant, and change the shape aesthetic by column `type`
p + geom_variant(aes(shape = type), position = position_variant())

# Now lets create a plot with different offsets by inserting a self-created vector.
p + geom_variant(
  aes(shape = type),
  position = position_variant(c(del = 0.4, ins = -0.4))
) + scale_shape_variant()

# Changing the base will shift all points up/down relatively from the sequence.
p + geom_variant(
  aes(shape = type),
  position = position_variant(base = 0.5)
) + geom_seq()
```

`read_alitv` *Read AliTV .json file*

Description

this file contains sequences, links and (optionally) genes

Usage

```
read_alitv(file)
```

Arguments

`file`	path to json

Value

list with seqs, genes, and links

Examples

```
ali <- read_alitv("https://alitvteam.github.io/AliTV/d3/data/chloroplasts.json")
gggenomes(ali$genes, ali$seqs, links = ali$links) +
  geom_seq() +
  geom_bin_label() +
  geom_gene(aes(fill = class)) +
  geom_link()
p <- gggenomes(ali$genes, ali$seqs, links = ali$links) +
  geom_seq() +
  geom_bin_label() +
  geom_gene(aes(color = class)) +
```

```
  geom_link(aes(fill = identity)) +
  scale_fill_distiller(palette = "RdYlGn", direction = 1)
p %>%
  flip_seqs(5) %>%
  pick_seqs(1, 3, 2, 4, 5, 6, 7, 8)
```

---

read_bed *Read a BED file*

Description

BED files use 0-based coordinate starts, while gggenomes uses 1-based start coordinates. BED file coordinates are therefore transformed into 1-based coordinates during import.

Usage

```
read_bed(file, col_names = def_names("bed"), col_types = def_types("bed"), ...)
```

Arguments

`file` Either a path to a file, a connection, or literal data (either a single string or a raw vector). `file` can also be a character vector containing multiple filepaths or a list containing multiple connections.

Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically decompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote compressed files (`.gz`, `.bz2`, `.xz`, `.zip`) will be automatically downloaded and decompressed.

Literal data is most useful for examples and tests. To be recognised as literal data, wrap the input with `I()`.

`col_names` column names to use. Defaults to `def_names("bed")` compatible with canonical bed files. `def_names()` can easily be combined with extra columns: `col_names = c(def_names("bed"), "more", "things")`.

`col_types` One of `NULL`, a `cols()` specification, or a string. See `vignette("readr")` for more details.

If `NULL`, all column types will be inferred from `guess_max` rows of the input, interspersed throughout the file. This is convenient (and fast), but not robust. If the guessed types are wrong, you'll need to increase `guess_max` or supply the correct types yourself.

Column specifications created by `list()` or `cols()` must contain one column specification for each column. If you only want to read a subset of the columns, use `cols_only()`.

Alternatively, you can use a compact string representation where each character represents one column:

- c = character
- i = integer

- n = number
- d = double
- l = logical
- f = factor
- D = date
- T = date time
- t = time
- ? = guess
- _ or - = skip

By default, reading a file without a column specification will print a message showing what `readr` guessed they were. To remove this message, set `show_col_types = FALSE` or set `options(readr.show_col_types = FALSE)`.

`...` additional parameters, passed to `read_tsv`

Value

tibble

`read_blast` *Read BLAST tab-separated output*

Description

Read BLAST tab-separated output

Usage

```
read_blast(
  file,
  col_names = def_names("blast"),
  col_types = def_types("blast"),
  comment = "#",
  swap_query = FALSE,
  ...
)
```

Arguments

`file` Either a path to a file, a connection, or literal data (either a single string or a raw vector). `file` can also be a character vector containing multiple filepaths or a list containing multiple connections.

Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically decompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote compressed files (`.gz`, `.bz2`, `.xz`, `.zip`) will be automatically downloaded and decompressed.

Literal data is most useful for examples and tests. To be recognised as literal data, wrap the input with `I()`.

col_names	column names to use. Defaults to `def_names("blast")` compatible with blast tabular output (`--outfmt 6/7` in blast++ and `-m8` in blast-legacy). `def_names()` can easily be combined with extra columns: `col_names = c(def_names("blast"), "more", "things")`.
col_types	column types to use. Defaults to `def_types("gff3")` (see `def_types`).
comment	character
swap_query	if TRUE swap query and subject columns using `swap_query()` on import.
...	additional parameters, passed to `read_tsv`

Value

a tibble with the BLAST output

`read_context` *Read files in different contexts*

Description

Powers `read_seqs()`, `read_feats()`, `read_links()`

Usage

```
read_context(
  files,
  context,
  .id = "file_id",
  format = NULL,
  parser = NULL,
  ...
)
```

Arguments

files	files to reads. Should all be of same format. In many cases, compressed files (`.gz`, `.bz2`, `.xz`, or `.zip`) are supported. Similarly, automatic download of remote files starting with `http(s)://` or `ftp(s)://` works in most cases.
context	the context ("seqs", "feats", "links") in which a given format should be read.
.id	the column with the name of the file a record was read from. Defaults to "file_id". Set to "bin_id" if every file represents a different bin.
format	specify a format known to gggenomes, such as `gff3`, `gbk`, ... to overwrite automatic determination based on the file extension (see `def_formats()` for full list).
parser	specify the name of an R function to overwrite automatic determination based on format, e.g. `parser="read_tsv"`.
...	additional arguments passed on to the format-specific read function called down the line.

Value

a tibble with the combined data from all files

Functions

- `read_context()`: bla keywords internal

---

read_gbk *Read genbank files*

Description

Genbank flat files (.gb/.gbk/.gbff) and their ENA and DDBJ equivalents have a particularly gruesome format. That's why `read_gbk()` is just a wrapper around a Perl-based `gb2gff` converter and `read_gff3()`.

Usage

```
read_gbk(file, sources = NULL, types = NULL, infer_cds_parents = TRUE)
```

Arguments

`file` Either a path to a file, a connection, or literal data (either a single string or a raw vector). `file` can also be a character vector containing multiple filepaths or a list containing multiple connections.

Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically decompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote compressed files (`.gz`, `.bz2`, `.xz`, `.zip`) will be automatically downloaded and decompressed.

Literal data is most useful for examples and tests. To be recognised as literal data, wrap the input with `I()`.

`sources` only return features from these sources

`types` only return features of these types, e.g. gene, CDS, ...

`infer_cds_parents` infer the mRNA parent for CDS features based on overlapping coordinates. Default TRUE for gff2/gtf, FALSE for gff3. In most GFFs this is properly set, but sometimes this information is missing. Generally, this is not a problem, however, geom_gene calls parse the parent information to determine which CDS and mRNAs are part of the same gene model. Without the parent info, mRNA and CDS are plotted as individual features.

Value

tibble

read_gff3 *Read features from GFF3 (and with some limitations GFF2/GTF) files*

Description

Files with ##FASTA section work but result in parsing problems for all lines of the fasta section. Just ignore those warnings, or strip the fasta section ahead of time from the file.

Usage

```
read_gff3(
  file,
  sources = NULL,
  types = NULL,
  infer_cds_parents = is_gff2,
  sort_exons = TRUE,
  col_names = def_names("gff3"),
  col_types = def_types("gff3"),
  keep_attr = FALSE,
  fix_augustus_cds = TRUE,
  is_gff2 = NULL
)
```

Arguments

`file` Either a path to a file, a connection, or literal data (either a single string or a raw vector). `file` can also be a character vector containing multiple filepaths or a list containing multiple connections.

Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically decompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote compressed files (`.gz`, `.bz2`, `.xz`, `.zip`) will be automatically downloaded and decompressed.

Literal data is most useful for examples and tests. To be recognised as literal data, wrap the input with `I()`.

`sources` only return features from these sources

`types` only return features of these types, e.g. gene, CDS, ...

`infer_cds_parents` infer the mRNA parent for CDS features based on overlapping coordinates. Default TRUE for gff2/gtf, FALSE for gff3. In most GFFs this is properly set, but sometimes this information is missing. Generally, this is not a problem, however, geom_gene calls parse the parent information to determine which CDS and mRNAs are part of the same gene model. Without the parent info, mRNA and CDS are plotted as individual features.

`sort_exons` make sure that exons/introns appear sorted. Default TRUE. Set to FALSE to read CDS/exon order exactly as present in the file, which is less robust, but faster and allows non-canonical splicing (exon1-exon3-exon2).

col_names	column names to use. Defaults to `def_names("gff3")` (see `def_names`).
col_types	column types to use. Defaults to `def_types("gff3")` (see `def_types`).
keep_attr	keep the original attributes column also after parsing tag=value pairs into tidy columns.
fix_augustus_cds	If true, assume Augustus gff with bad CDS IDs that need fixing
is_gff2	set if file is in gff2 format

Value

tibble

`read_paf` *Read a .paf file (minimap/minimap2).*

Description

Read a minimap/minimap2 .paf file including optional tagged extra fields. The optional fields will be parsed into a tidy format, one column per tag.

Usage

```
read_paf(
  file,
  max_tags = 20,
  col_names = def_names("paf"),
  col_types = def_types("paf"),
  ...
)
```

Arguments

file	Either a path to a file, a connection, or literal data (either a single string or a raw vector). `file` can also be a character vector containing multiple filepaths or a list containing multiple connections. Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically decompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote compressed files (`.gz`, `.bz2`, `.xz`, `.zip`) will be automatically downloaded and decompressed. Literal data is most useful for examples and tests. To be recognised as literal data, wrap the input with `I()`.
max_tags	maximum number of optional fields to include
col_names	column names to use. Defaults to `def_names("gff3")` (see `def_names`).
col_types	column types to use. Defaults to `def_types("gff3")` (see `def_types`).
...	additional parameters, passed to `read_tsv`

Details

Because `readr::read_tsv` expects a fixed number of columns, but in .paf the number of optional fields can differ among records, `read_paf` tries to read at least as many columns as the longest record has (`max_tags`). The resulting warnings for each record with fewer fields of the form "32 columns expected, only 22 seen" should thus be ignored.

From the minimap2 manual

+—-+———+————————————————————————+ |Col | Type | Description | +—-+——+————————————————————————+ | 1 | string | Query sequence name | | 2 | int | Query sequence length | | 3 | int | Query start coordinate (0-based) | | 4 | int | Query end coordinate (0-based) | | 5 | char | '+' if query/target on the same strand; '-' if opposite | | 6 | string | Target sequence name | | 7 | int | Target sequence length | | 8 | int | Target start coordinate on the original strand | | 9 | int | Target end coordinate on the original strand | | 10 | int | Number of matching bases in the mapping | | 11 | int | Number bases, including gaps, in the mapping | | 12 | int | Mapping quality (0-255 with 255 for missing) | +—-+———+————————————————————————+

+—-+———+————————————————————————-+ |Tag | Type | Description | +—-+———+————————————————————————-+ | tp | A | Type of aln: P/primary, S/secondary and I,i/inversion | | cm | i | Number of minimizers on the chain | | s1 | i | Chaining score | | s2 | i | Chaining score of the best secondary chain | | NM | i | Total number of mismatches and gaps in the alignment | | MD | Z | To generate the ref sequence in the alignment | | AS | i | DP alignment score | | ms | i | DP score of the max scoring segment in the alignment | | nn | i | Number of ambiguous bases in the alignment | | ts | A | Transcript strand (splice mode only) | | cg | Z | CIGAR string (only in PAF) | | cs | Z | Difference string | | dv | f | Approximate per-base sequence divergence | +—-+———+————————————————————————-+

From https://samtools.github.io/hts-specs/SAMtags.pdf type may be one of A (character), B (general array), f (real number), H (hexadecimal array), i (integer), or Z (string).

Value

tibble

`read_seq_len` *Read sequence index*

Description

Read sequence index

Usage

```
read_seq_len(file)

read_fai(file, col_names = def_names("fai"), col_types = def_types("fai"), ...)
```

Arguments

`file` with sequence length information

`col_names` Either `TRUE`, `FALSE` or a character vector of column names.

If `TRUE`, the first row of the input will be used as the column names, and will not be included in the data frame. If `FALSE`, column names will be generated automatically: X1, X2, X3 etc.

If `col_names` is a character vector, the values will be used as the names of the columns, and the first row of the input will be read into the first row of the output data frame.

Missing (`NA`) column names will generate a warning, and be filled in with dummy names `...1`, `...2` etc. Duplicate column names will generate a warning and be made unique, see `name_repair` to control how this is done.

`col_types` One of `NULL`, a `cols()` specification, or a string. See `vignette("readr")` for more details.

If `NULL`, all column types will be inferred from `guess_max` rows of the input, interspersed throughout the file. This is convenient (and fast), but not robust. If the guessed types are wrong, you'll need to increase `guess_max` or supply the correct types yourself.

Column specifications created by `list()` or `cols()` must contain one column specification for each column. If you only want to read a subset of the columns, use `cols_only()`.

Alternatively, you can use a compact string representation where each character represents one column:

- c = character
- i = integer
- n = number
- d = double
- l = logical
- f = factor
- D = date
- T = date time
- t = time
- ? = guess
- _ or - = skip

By default, reading a file without a column specification will print a message showing what `readr` guessed they were. To remove this message, set `show_col_types = FALSE` or set `options(readr.show_col_types = FALSE)`.

`...` additional parameters, passed to `read_tsv`

Value

tibble with sequence information

tibble with sequence information

Functions

- `read_seq_len()`: read seqs from a single file_name in fasta, gbk or gff3 format.
- `read_fai()`: read seqs from a single file in seqkit/samtools fai format.

`read_tracks` *Read files in various standard formats (FASTA, GFF3, GBK, BED, BLAST, ...) into track tables*

Description

Convenience functions to read sequences, features or links from various bioinformatics file formats, such as FASTA, GFF3, Genbank, BLAST tabular output, etc. See `def_formats()` for full list. File formats and the corresponding read-functions are automatically determined based on file extensions. All these functions can read multiple files in the same format at once, and combine them into a single table - useful, for example, to read a folder of gff-files with each file containing genes of a different genome.

Usage

```
read_feats(files, .id = "file_id", format = NULL, parser = NULL, ...)

read_subfeats(files, .id = "file_id", format = NULL, parser = NULL, ...)

read_links(files, .id = "file_id", format = NULL, parser = NULL, ...)

read_sublinks(files, .id = "file_id", format = NULL, parser = NULL, ...)

read_seqs(
  files,
  .id = "file_id",
  format = NULL,
  parser = NULL,
  parse_desc = TRUE,
  ...
)
```

Arguments

`files`	files to reads. Should all be of same format. In many cases, compressed files (`.gz`, `.bz2`, `.xz`, or `.zip`) are supported. Similarly, automatic download of remote files starting with `http(s)://` or `ftp(s)://` works in most cases.
`.id`	the column with the name of the file a record was read from. Defaults to "file_id". Set to "bin_id" if every file represents a different bin.
`format`	specify a format known to gggenomes, such as `gff3`, `gbk`, ... to overwrite automatic determination based on the file extension (see `def_formats()` for full list).

parser	specify the name of an R function to overwrite automatic determination based on format, e.g. parser="read_tsv".
...	additional arguments passed on to the format-specific read function called down the line.
parse_desc	turn key=some value pairs from seq_desc into key-named columns and remove them from seq_desc.

Value

A gggenomes-compatible sequence, feature or link tibble

tibble with features

tibble with features

tibble with links

tibble with links

tibble with sequence information

Functions

- read_feats(): read files as features mapping onto sequences.
- read_subfeats(): read files as subfeatures mapping onto other features
- read_links(): read files as links connecting sequences
- read_sublinks(): read files as sublinks connecting features
- read_seqs(): read sequence ID, description and length.

Examples

```
# read genes/features from a gff file
read_feats(ex("eden-utr.gff"))


# read all gff files from a directory
read_feats(list.files(ex("emales/"), "*.gff$", full.names = TRUE))


# read remote files

gbk_phages <- c(
  PSSP7 = paste0(
    "https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/",
    "000/858/745/GCF_000858745.1_ViralProj15134/",
    "GCF_000858745.1_ViralProj15134_genomic.gff.gz"
  ),
  PSSP3 = paste0(
    "https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/",
    "000/904/555/GCF_000904555.1_ViralProj195517/",
    "GCF_000904555.1_ViralProj195517_genomic.gff.gz"
  )
```

```
)
read_feats(gbk_phages)


# read sequences from a fasta file.
read_seqs(ex("emales/emales.fna"), parse_desc = FALSE)

# read sequence info from a fasta file with `parse_desc=TRUE` (default). `key=value`
# pairs are removed from `seq_desc` and parsed into columns with `key` as name
read_seqs(ex("emales/emales.fna"))

# read sequence info from samtools/seqkit style index
read_seqs(ex("emales/emales.fna.seqkit.fai"))

# read sequence info from multiple gff file
read_seqs(c(ex("emales/emales.gff"), ex("emales/emales-tirs.gff")))
```

`read_vcf` *Read a VCF file*

Description

VCF (Variant Call Format) file format is used to store variation data and its metadata. Based on the used analysis program (e.g. GATK, freebayes, etc...), details within the VCF file can slightly differ. For example, type of mutation is not mentioned as output for certain variant analysis programs. the "read_vcf" function, ignores the first header/metadata lines and directly converts the data into a tidy dataframe. The function will extract the type of mutation. By absence, it will derive the type of mutation from the "ref" and "alt" column.

Usage

```
read_vcf(
  file,
  parse_info = FALSE,
  col_names = def_names("vcf"),
  col_types = def_types("vcf")
)
```

Arguments

`file` Either a path to a file, a connection, or literal data (either a single string or a raw vector). `file` can also be a character vector containing multiple filepaths or a list containing multiple connections.

Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically decompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote compressed files (`.gz`, `.bz2`, `.xz`, `.zip`) will be automatically downloaded and decompressed.

Literal data is most useful for examples and tests. To be recognised as literal data, wrap the input with `I()`.

`parse_info` if set to 'TRUE', the read_vcf function will split all the metadata stored in the "info" column and stores it into separate columns. By default it is set to 'FALSE'.

`col_names` column names to use. Defaults to `def_names("vcf")` (see `def_names`).

`col_types` column types to use. Defaults to `def_types("vcf")` (see `def_types`).

Value

dataframe

`require_vars` *Require variables in an object*

Description

Require variables in an object

Usage

```
require_vars(x, vars, warn_only = FALSE)
```

Arguments

`x` object

`vars` required variables

`warn_only` don't die on missing vars

Value

the original tibble if all vars are present or warning only

scale_color_variant *Default colors and shapes for mutation types.*

Description

The user can call upon an convenient function called `scale_color_variant`, which changes the color of (SNP) points, based on their nucleotides (A, C, G, T). By default the function uses a colorblind friendly palette, but users can manually overwrite these colors. *(Within the plotting function (e.g.* `geom_variant`*), coloring of the column should still be mentioned (*`aes(color = ...)`*).*

The function `scale_shape_variant` changes the shape of plotted points based on the type of mutation. The user can also manually decide which shape, each specific type of mutation should have. By default, SNPs are diamond shaped, Deletions triangle downwards and Insertions triangle upwards. (These default settings make most sense when using `geom_variant(offset = -0.2)`). *(User should still manually call which column is used for the shape aesthetic)*

Usage

```
scale_color_variant(
  values = c(A = "#e66101", C = "#b2abd2", G = "#5e3c99", T = "#fdb863"),
  na.value = "white",
  ...
)

scale_shape_variant(
  values = c(SNP = 23, Deletion = 25, Insertion = 24),
  na.value = 1,
  characters = FALSE,
  ...
)
```

Arguments

`values`	A vector indicating how to color/shape different variables. The functions `scale_color_variant()` and `scale_shape_variant()` have a default setting, which can be overwritten.
`na.value`	The aesthetic value (color/shape/etc.) to use for non matching values.
`...`	Additional parameters, passed to scale_color_manual
`characters`	When `TRUE`, it changes the default shapes of `scale_shape_variant()` to become the letters of the nucleotides.

Value

A ggplot2 scale object for color or shape.

Examples

```
# Creation of example data.
testposition <- tibble::tibble(
  type = c(
    "Insertion", "SNP", "SNP", "Deletion",
    "Deletion", "SNP", "SNP", "Insertion", "SNP", "Insertion", "SNP"
  ),
  start = c(10, 20, 30, 35, 40, 60, 65, 90, 90, 100, 120),
  ALT = c("AT", "G", "C", ".", ".", "T", "C", "CAT", "G", "TC", "A"),
  REF = c("A", "T", "G", "A", "A", "G", "A", "C", "A", "T", "G"),
  end = start + 1,
  seq_id = c(rep("A", 11))
)

testseq <- tibble::tibble(
  seq_id = "A",
  start = 0,
  end = 150,
  length = end - start
)

p1 <- gggenomes(seqs = testseq, feats = testposition)
p2 <- p1 + geom_seq()

## Scale_color_variant()
# Changing the color aesthetics in geom_variant: colors all mutations
# (In this example, All ALT (alternative) nucleotides are being colored)
p1 + geom_variant(aes(color = ALT))

# Color all SNPs with default colors using scale_color_variant().
# (SNPs are 1 nucleotide long, other mutations such as Insertions
# and Deletions have either more ore less nucleotides within the
# ALT column and are thus not plotted)
p1 + geom_variant(aes(color = ALT)) +
  scale_color_variant()

# Manually changing colors with scale_color_variant()
p1 + geom_variant(aes(color = ALT)) +
  scale_color_variant(values = c(A = "purple", T = "darkred", TC = "black", AT = "pink"))

## Scale_shape_variant()
# Changing the `shape` aesthetics in geom_variant
p2 + geom_variant(aes(shape = type), offset = -0.1)

# Calling upon scale_shape_variant() to change shapes
p2 + geom_variant(aes(shape = type), offset = -0.1) +
  scale_shape_variant()

# Manually changing shapes with scale_shape_variant()
p2 + geom_variant(aes(shape = type), offset = -0.1) +
  scale_shape_variant(values = c(SNP = 14, Deletion = 18, Insertion = 21))
```

```
# Plotting (nucleotides) characters instead of shapes
p2 + geom_variant(aes(shape = ALT), offset = -0.1, size = 3) +
  scale_shape_variant(characters = TRUE)

# Alternative way to plot nucleotides (of ALT) by using `geom=text` within `geom_variant()`
gggenomes(seqs = testseq, feats = testposition) +
  geom_seq() +
  geom_variant(aes(shape = type), offset = -0.1) +
  scale_shape_variant() +
  geom_variant(aes(label = ALT), geom = "text", offset = -0.25) +
  geom_bin_label()

# Combining scale_color_variant() and scale_shape_variant()
p2 + geom_variant(aes(shape = ALT, color = ALT), offset = -0.1, size = 3, show.legend = FALSE) +
  geom_variant(aes(color = ALT)) +
  scale_color_variant(na.value = "black") +
  scale_shape_variant(characters = TRUE)
```

`scale_x_bp` *X-scale for genomic data*

Description

`scale_x_bp()` is the default scale for genomic x-axis. It wraps `ggplot2::scale_x_continuous()` using `label_bp()` as default labeller.

Usage

```
scale_x_bp(..., suffix = "", sep = "", accuracy = 1)

label_bp(suffix = "", sep = "", accuracy = 1)
```

Arguments

`...`	Arguments passed on to `ggplot2::scale_x_continuous()`
`suffix`	unit suffix e.g. "bp"
`sep`	between number and unit prefix+suffix
`accuracy`	A number to round to. Use (e.g.) `0.01` to show 2 decimal places of precision. If `NULL`, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.

Value

A ggplot2 scale object with bp labels

A labeller function for genomic data

Examples

```
# scale_x_bp invoked by default
gggenomes(emale_genes) + geom_gene()

# customize labels
gggenomes(emale_genes) + geom_gene() +
  scale_x_bp(suffix = "bp", sep = " ")

# Note: xlim will overwrite scale_x_bp() with ggplot2::scale_x_continuous()
gggenomes(emale_genes) + geom_gene() +
  xlim(0, 3e4)

# set limits explicitly with scale_x_bp() to avoid overwrite
gggenomes(emale_genes) + geom_gene() +
  scale_x_bp(limits = c(0, 3e4))
```

`set_class` *Modify object class attributes*

Description

Set class of an object. Optionally append or prepend to exiting class attributes. `add_class` is short for `set_class(x, class, "prepend")`. `strip_class` removes matching class strings from the class attribute vector.

Usage

```
set_class(x, class, add = c("overwrite", "prepend", "append"))

add_class(x, class)

strip_class(x, class)
```

Arguments

`x`	Object to assign new class to.
`class`	Class value to add/strip.
`add`	Possible values: "overwrite", "prepend", "append"

Value

Object x as class value.

shift *Shift bins left/right*

Description

Shift bins along the x-axis, i.e. left or right in the default plot layout. This is useful to align feats of interest in different bins.

Usage

```
shift(x, bins = everything(), by = 0, center = FALSE)
```

Arguments

x	gggenomes object
bins	to shift left/right, select-like expression
by	shift each bin by this many bases. Single value or vector of the same length as bins.
center	horizontal centering

Value

gggenomes object with shifted seqs

Examples

```
p0 <- gggenomes(emale_genes, emale_seqs) +
  geom_seq() + geom_gene()

# Slide one bin left and one bin right
p1 <- p0 |> shift(2:3, by = c(-8000, 10000))

# align all bins to a target gene
mcp <- emale_genes |>
  dplyr::filter(name == "MCP") |>
  dplyr::group_by(seq_id) |>
  dplyr::slice_head(n = 1) # some have fragmented MCP gene, keep only first

p2 <- p0 |> shift(all_of(mcp$seq_id), by = -mcp$start) +
  geom_gene(data = genes(name == "MCP"), fill = "#01b9af")

library(patchwork)
p0 + p1 + p2
```

strand_chr — *Convert strand to character*

Description

Convert strand to character

Usage

```
strand_chr(strand, na = NA)
```

Arguments

strand	some representation for strandedness
na	what to use for NA

Value

strand vector as character

strand_int — *Convert strand to integer*

Description

Convert strand to integer

Usage

```
strand_int(strand, na = NA)
```

Arguments

strand	some representation for strandedness
na	what to use for NA

Value

strand vector as integer

strand_lgl — *Convert strand to logical*

Description

Convert strand to logical

Usage

```
strand_lgl(strand, na = NA)
```

Arguments

strand	some representation for strandedness
na	what to use for NA

Value

strand vector as logical

swap_if — *Swap values of two columns based on a condition*

Description

Swap values of two columns based on a condition

Usage

```
swap_if(x, condition, ...)
```

Arguments

x	a tibble
condition	an expression to be evaluated in data context returning a TRUE/FALSE vector
...	the two columns between which values are to be swapped in dplyr::select-like syntax

Value

a tibble with conditionally swapped start and end

Examples

```
x <- tibble::tibble(start = c(10, 100), end = c(30, 50))
# ensure start of a range is always smaller than the end
swap_if(x, start > end, start, end)
```

swap_query *Swap query and subject in blast-like feature tables*

Description

Swap query and subject columns in a table read with `read_feats()` or `read_links()`, for example, from blast searches. Swaps columns with name/name2, such as 'seq_id/seq_id2', 'start/start2', ...

Usage

```
swap_query(x)
```

Arguments

x	tibble with query and subject columns

Value

tibble with swapped query/subject columns

Examples

```
feats <- tibble::tribble(
  ~seq_id, ~seq_id2, ~start, ~end, ~strand, ~start2, ~end2, ~evalue,
  "A", "B", 100, 200, "+", 10000, 10200, 1e-5
)
# make B the query
swap_query(feats)
```

theme_gggenomes_clean *gggenomes default theme*

Description

gggenomes default theme

Usage

```
theme_gggenomes_clean(
  base_size = 12,
  base_family = "",
  base_line_size = base_size/30,
  base_rect_size = base_size/30
)
```

Arguments

base_size	base font size, given in pts.
base_family	base font family
base_line_size	base size for line elements
base_rect_size	base size for rect elements

Value

ggplot2 theme with gggenomes defaults

---

track_ids *Named vector of track ids and types*

Description

Named vector of track ids and types

Usage

```
track_ids(x, track_type, ...)
```

Arguments

x	A gggenomes or gggenomes_layout object
track_type	restrict to any combination of "seqs", "feats" and "links".
...	unused

Value

a named vector of track ids and types

---

track_info *Basic info on tracks in a gggenomes object*

Description

Use `track_info()` to call on a gggenomes or gggenomes_layout object to return a short tibble with ids, types, index and size of the loaded tracks.

Usage

```
track_info(x, ...)
```

Arguments

x	A gggenomes or gggenomes_layout object
...	unused

Details

The short tibble contains basic information on the tracks within the entered gggenomes object.

- **id** : Shows original name of inputted data frame (only when more than one data frames are present in a track).
- **type** : The track in which the data frame is present.
- **i** (index) : The chronological order of data frames in a specific track.
- **n** (size) : Amount of objects **plotted** from the data frame. (**not** the amount of objects *in* the inputted data frame)

Value

Short tibble with ids, types, index and size of loaded tracks.

Examples

```
gggenomes(
  seqs = emale_seqs,
  feats = list(emale_genes, emale_tirs, emale_ngaros),
  links = emale_ava
) |>
  track_info()
```

---

unnest_exons *Unnest exons*

Description

Unnest exons

Usage

```
unnest_exons(x)
```

Arguments

x	data

Value

data with unnested exons

vars_track — *Tidyselect track variables*

Description

Based on `tidyselect::vars_pull`. Powers track selection in `pull_track()`. Catches and modifies errors from vars_pull to track-relevant info.

Usage

```
vars_track(
  x,
  track_id,
  track_type = c("seqs", "feats", "links"),
  ignore = NULL
)
```

Arguments

`x`	A gggenomes or gggenomes_layout object
`track_id`	a quoted or unquoted name or as positive/negative integer giving the position from the left/right.
`track_type`	restrict to these types of tracks - affects position-based selection
`ignore`	names of tracks to ignore when selecting by position.

Value

The selected track_id as an unnamed string

width — *The width of a range*

Description

Always returns a positive value, even if start > end. `width0` is a short handle for `width(..., base=0)`

Usage

```
width(start, end, base = 1)

width0(start, end, base = 0)
```

Arguments

start, end	start and end of the range
base	the base of the coordinate system, usually 1 or 0.

Value

a numeric vector

---

write_gff3 *Write a gff3 file from a tidy table*

Description

Write a gff3 file from a tidy table

Usage

```
write_gff3(
  feats,
  file,
  seqs = NULL,
  type = NULL,
  source = ".",
  score = ".",
  strand = ".",
  phase = ".",
  id_var = "feat_id",
  parent_var = "parent_ids",
  head = "##gff-version 3",
  ignore_attr = c("introns", "geom_id")
)
```

Arguments

feats	tidy feat table
file	name of output file
seqs	a tidy sequence table to generate optional `##sequence-region` directives in the header
type	if no type column exists, use this as the default type
source	if no source column exists, use this as the default source
score	if no score column exists, use this as the default score
strand	if no strand column exists, use this as the default strand
phase	if no phase column exists, use this as the default phase
id_var	the name of the column to use as the GFF3 `ID` tag

parent_var	the name of the column to use as GFF3 Parent tag
head	additional information to add to the header section
ignore_attr	attributes not to be included in GFF3 tag list. Defaults to internals: introns, geom_id

Value

No return value, writes to file

Examples

```
filename <- tempfile(fileext = ".gff")
write_gff3(emale_genes, filename, emale_seqs, id_var = "feat_id")
```

Index

∗ **datasets**
emale_ava, 12
emale_cogs, 12
emale_gc, 13
emale_genes, 14
emale_ngaros, 15
emale_prot_ava, 16
emale_seqs, 16
emale_tirs, 17
GeomFeatText, 25
position_strand, 69

add_class *(*set_class*)*, 88
add_clusters *(*add_feats*)*, 3
add_feats, 3
add_links *(*add_feats*)*, 3
add_seqs, 5
add_subfeats *(*add_feats*)*, 3
add_sublinks *(*add_feats*)*, 3
add_tracks *(*add_feats*)*, 3
aes(), *26*, *29*, *32*, *37*, *40*, *44*, *47*, *49*, *54*, *57*
align, 5
alpha, *42*
annotation_borders(), *30*, *33*, *38*, *41*, *47*, *52*, *57*

base::max, *30*
base::min, *30*
bins *(*feats*)*, 18

check_strand, 7
colour, *42*
cols(), *73*, *80*
cols_only(), *73*, *80*
combine_strands, 7

def_formats, 8
def_formats(), *75*, *81*
def_names, 9, *78*, *84*
def_names(), *73*, *75*
def_types, *75*, *78*, *84*
def_types *(*def_names*)*, 9
dplyr::filter, *19*
dplyr::filter(), *18*
dplyr::mutate(), *63*
dplyr::pull(), *18*
dplyr::select(), *67*
drop_feat_layout, 10
drop_layout, 10
drop_link_layout, 11
drop_seq_layout, 11

emale_ava, 12
emale_cogs, 12
emale_gc, 13
emale_genes, 14
emale_ngaros, 15
emale_prot_ava, 16
emale_seqs, 16
emale_tirs, 17
ex, 18

feats, 18
feats0 *(*feats*)*, 18
fill, *42*
flip, 20
flip_seqs *(*flip*)*, 20
flip_strand, 22
focus, 22
fortify(), *26*, *29*, *37*, *40*, *44*, *47*, *55*

genes *(*feats*)*, 18
geom_bin_label, 26
geom_bin_label(), *43*
geom_coverage, 28
geom_feat, 32
geom_feat_label *(*geom_gene_label*)*, 43
geom_feat_note *(*geom_feat_text*)*, 34
geom_feat_tag *(*geom_feat_text*)*, 34
geom_feat_text, 34

geom_feat_text(), *43*
geom_gene, 39
geom_gene_label, 43
geom_gene_note *(geom_feat_text)*, 34
geom_gene_tag *(geom_feat_text)*, 34
geom_gene_text *(geom_feat_text)*, 34
geom_link, 45
geom_link_curved *(geom_link)*, 45
geom_link_label *(geom_gene_label)*, 43
geom_link_line *(geom_link)*, 45
geom_seq, 49
geom_seq_break, 51
geom_seq_label, 54
geom_seq_label(), *43*
geom_variant, 56
geom_wiggle *(geom_coverage)*, 28
GeomFeatText, 25
get_seqs, 59
gggenomes, 60
ggplot(), *26*, *29*, *37*, *40*, *44*, *47*, *54*
ggplot2::aes(), *41*
ggplot2::after_scale(), *41*
ggplot2::geom_line(), *31*
ggplot2::geom_linerange(), *31*
ggplot2::geom_point(), *31*
ggplot2::geom_ribbon(), *31*
ggplot2::geom_text(), *45*, *55*
ggplot2::scale_x_continuous(), *87*
ggtree::ggtree, *67*
group, *42*

Hmisc::smedian.hilow, *30*

if_reverse, 63
in_range, 64
introduce, 63
is_reverse, 65

key glyphs, *27*, *30*, *33*, *38*, *45*, *48*, *50*, *53*, *55*, *58*

label_bp *(scale_x_bp)*, 87
layer geom, *29*
layer position, *29*, *37*, *41*, *47*
layer stat, *29*, *32*, *37*, *41*, *47*, *52*
layer(), *27*, *30*, *33*, *37*, *38*, *45*, *48*, *50*, *53*, *55*, *57*, *58*
layout, 65
layout_genomes(), *61*
layout_seqs, 66
linetype, *42*
links *(feats)*, 18
list(), *73*, *80*
locate *(focus)*, 22

pick, 67
pick_by_tree *(pick)*, 67
pick_seqs *(pick)*, 67
pick_seqs_within *(pick)*, 67
position_pile *(position_strand)*, 69
position_sixframe *(position_strand)*, 69
position_strand, 69
position_strandpile *(position_strand)*, 69
position_variant, 71
PositionPile *(position_strand)*, 69
PositionSixframe *(position_strand)*, 69
PositionStrand *(position_strand)*, 69
PositionStrandpile *(position_strand)*, 69
pull_bins *(feats)*, 18
pull_feats *(feats)*, 18
pull_genes *(feats)*, 18
pull_links *(feats)*, 18
pull_seqs *(feats)*, 18
pull_track *(feats)*, 18

read_alitv, 72
read_bed, 73
read_blast, 74
read_context, 75
read_fai *(read_seq_len)*, 79
read_feats *(read_tracks)*, 81
read_feats(), *75*, *92*
read_gbk, 76
read_gbk(), *76*
read_gff3, 77
read_gff3(), *76*
read_links *(read_tracks)*, 81
read_links(), *75*, *92*
read_paf, 78
read_seq_len, 79
read_seqs *(read_tracks)*, 81
read_seqs(), *75*
read_subfeats *(read_tracks)*, 81
read_sublinks *(read_tracks)*, 81
read_tracks, 81
read_vcf, 83
readr::read_tsv(), *9*

require_vars, 84

scale_color_variant, 85
scale_shape_variant
(scale_color_variant), 85
scale_x_bp, 87
seqs (feats), 18
set_class, 88
set_seqs (get_seqs), 60
shift, 89
strand_chr, 90
strand_int, 90
strand_lgl, 91
strip_class (set_class), 88
swap_if, 91
swap_query, 92
swap_query(), *75*
sync (flip), 20

theme_gggenomes_clean, 92
tidyselect::starts_with(), *67*
tidyselect::where(), *21*
track (feats), 18
track_ids, 93
track_info, 93

unnest_exons, 94

vars_track, 95

width, 95
width0 (width), 95
write_gff3, 96

x, *42*
xend, *42*

y, *42*