Package 'ggh4x'

July 22, 2025

Title Hacks for 'ggplot2'

Version 0.3.1

Description A 'ggplot2' extension that does a variety of little helpful things. The package extends 'ggplot2' facets through customisation, by setting individual scales per panel, resizing panels and providing nested facets. Also allows multiple colour and fill scales per plot. Also hosts a smaller collection of stats, geoms and axis guides.

License MIT + file LICENSE

URL https://github.com/teunbrand/ggh4x, https://teunbrand.github.io/ggh4x/

BugReports https://github.com/teunbrand/ggh4x/issues

Depends ggplot2 (>= 3.5.2)

Imports grid, gtable, scales, vctrs (>= 0.5.0), rlang (>= 1.1.0), lifecycle, stats, cli, S7

Suggests covr, fitdistrplus, ggdendro, vdiffr, knitr, MASS, rmarkdown, testthat (>= 3.0.0), utils

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

Collate 'at_panel.R' 'borrowed_ggplot2.R' 'conveniences.R' 'ggh4x_extensions.R' 'coord_axes_inside.R' 'deprecated.R' 'element_part_rect.R' 'facet_grid2.R' 'facet_wrap2.R' 'facet_manual.R' 'facet_nested.R' 'facet_nested_wrap.R' 'facetted_pos_scales.R' 'force_panelsize.R' 'geom_box.R' 'geom_outline_point.R' 'geom_pointpath.R' 'geom_polygonraster.R' 'geom_rectrug.R' 'geom_text_aimed.R' 'ggh4x-package.R' 'guide_stringlegend.R' 'help_secondary.R' 'position_disjoint_ranges.R' 'position_lineartrans.R' 'scale_facet.R' 'scale_listed.R' 'scale_manual.R'

'scale_multi.R' 'stat_difference.R' 'stat_funxy.R' 'stat_rle.R'
'stat_roll.R' 'stat_theodensity.R' 'strip_vanilla.R'
'strip_themed.R' 'strip_nested.R' 'strip_split.R' 'strip_tag.R'
'themes.R' 'utils.R' 'utils_grid.R' 'utils_gtable.R'

NeedsCompilation no

Author Teun van den Brand [aut, cre] (ORCID:
<https://orcid.org/0000-0002-9335-7468>)

Maintainer Teun van den Brand <tahvdbrand@gmail.com>

Repository CRAN

Date/Publication 2025-05-30 08:00:01 UTC

Contents

`at_panel` *Constrain layer to panels*

Description

This function limits the panels in which a layer is displayed. It can be used to make panel-specific annotations.

Usage

```
at_panel(layer, expr)
```

Arguments

`layer`	A `layer` as returned by `layer()`. Alternatively, a bare list of layers.
`expr`	An `expression` that, when evaluated in the facet's layout data.frame, yields a `logical` vector parallel to the rows.

Details

The `expr` argument's expression will be evaluated in the context of the plot's layout. This is an internal `data.frame` structure that isn't ordinarily exposed to users, so it will require some extra knowledge. For most facets, the layout describes the panels with one panel per row. It typically has `COL`, `ROW` and `PANEL` columns that keep track of where a panel goes in a grid-layout of cells. In addition, the layout contains the facetting variables provided to the `facets` or `rows` and `cols` arguments of the facets. For example, if we have a plot facetted on the `var` variable with the levels `A`, `B` and `C`, as 1 row and 3 columns, we might target the second `B` panel iwth any of these expressions: `var == "B"`, `PANEL == 2` or `COL == 2`. We can inspect the layout structure by using `ggplot_build(p)$layout$layout`, wherein `p` is a plot.

Value

A modified `layer` which will only show in some panels.

Examples

```
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_grid(year ~ drv)

anno <- annotate("text", x = 3, y = 40, label = "My text")

# Target specific panels
p + at_panel(anno, PANEL %in% c(2, 4))

# Target a variable
p + at_panel(anno, drv == "f")

# Or combine variable with position
p + at_panel(anno, drv == "f" & ROW == 2)
```

`center_limits` *Center limits*

Description

This a function factory that allows the centering of scales around a certain value while still including all values. Convenient for centering log2 fold change limits around zero.

Usage

```
center_limits(around = 0)
```

Arguments

`around`	A `numeric` of length 1 indicating around which value to center the limits.

Value

A `function` that takes limits and returns expanded limits centered at the `around` argument.

Examples

```
center_limits(5)(c(3,8))

g <- ggplot(iris,
            aes(Sepal.Width, Sepal.Length,
                colour = log2(Petal.Width / Petal.Length))) +
  geom_point() +
  scale_colour_gradient2(limits = center_limits())
```

coord_axes_inside *Cartesian coordinates with interior axes*

Description

This coordinate system places the plot axes at interior positions. Other than this, it behaves like `coord_cartesian()` or `coord_fixed()` (the latter if the `ratio` argument is set).

Usage

```
coord_axes_inside(
  xlim = NULL,
  ylim = NULL,
  xintercept = 0,
  yintercept = 0,
  labels_inside = FALSE,
  ratio = NULL,
  expand = TRUE,
  default = FALSE,
  clip = "on"
)
```

Arguments

`xlim`, `ylim`	Limits for the x and y axes.
`xintercept`, `yintercept`	A `numeric(1)` for the positions where the orthogonal axes should be placed. If these are outside the bounds of the limits, the axes are placed to the nearest extreme.
`labels_inside`	One of `"x"`, `"y"`, `"both"` or `"none"` specifying the axes where labels should be placed inside the panel along the axes. `TRUE` is translated as `"both"` and `FALSE` (default) is translated as `"none"`.
`ratio`	Either `NULL`, or a `numeric(1)` for a fixed aspect ratio, expressed as `y / x`.
`expand`	If `TRUE`, the default, adds a small expansion factor to the limits to ensure that data and axes don't overlap. If `FALSE`, limits are taken exactly from the data or `xlim`/`ylim`.
`default`	Is this the default coordinate system? If `FALSE` (the default), then replacing this coordinate system with another one creates a message alerting the user that the coordinate system is being replaced. If `TRUE`, that warning is suppressed.
`clip`	Should drawing be clipped to the extent of the plot panel? A setting of `"on"` (the default) means yes, and a setting of `"off"` means no. In most cases, the default of `"on"` should not be changed, as setting `clip = "off"` can cause unexpected results. It allows drawing of data points anywhere on the plot, including in the plot margins. If limits are set via `xlim` and `ylim` and some data points fall outside those limits, then those data points may show up in places such as the axes, the legend, the plot title, or the plot margins.

Value

A `CoordAxesInside` object, which can be added to a plot.

Examples

```
# A standard plot
p <- ggplot(mpg, aes(scale(displ), scale(hwy))) +
  geom_point() +
  theme(axis.line = element_line())

# By default, axis text is still placed outside the panel
p + coord_axes_inside()

# However, this can simply be changed
p + coord_axes_inside(labels_inside = TRUE)

# The place where the axes meet can be changed
p + coord_axes_inside(xintercept = 1, yintercept = -1)

# Axes snap to the nearest limit when out-of-bounds
p + coord_axes_inside(xintercept = -5, yintercept = Inf, clip = "off")

# Can be combined with other non-default axes
p + guides(x = guide_axis(minor.ticks = TRUE)) +
  coord_axes_inside()
```

---

deprecated *Deprecated functions*

Description

The functions listed here are deprecated and no longer work.

Usage

```
guide_axis_logticks(...)

guide_axis_manual(...)

guide_axis_minor(...)

guide_axis_nested(...)

guide_axis_scalebar(...)

guide_axis_truncated(...)

guide_axis_colour(...)
```

```
guide_axis_color(...)

guide_dendro(...)

ggsubset(...)

scale_x_dendrogram(...)

scale_y_dendrogram(...)
```

Arguments

...	Not used.

Value

None, raises deprecation signal

Examples

```
# None
```

`distribute_args` *Element list constructors*

Description

These functions take a vector of arguments and pass on the i^{th} item of the vector to an i^{th} call of a function. The `elem_list_text` and `elem_list_rect` are convenience functions for constructing lists of `element_text()` and `element_rect()` theme elements.

Usage

```
distribute_args(..., .fun = element_text, .cull = TRUE)

elem_list_text(...)

elem_list_rect(...)
```

Arguments

...	Vectorised arguments to pass on to functions.
.fun	A function to distribute arguments to.
.cull	A `logical(1)` determining if unknown arguments are being culled.

Details

NAs and NULLs will be silently dropped. If you want to pass on a transparent `fill` or `colour` argument, you should use the more verbose character `"transparent"` instead. However, you *can* use a NA to indicate that it's argument should not be passed to a function in that position.

Value

A `list` of outputs from `fun`.

Note

Whereas the `distribute_args` function might seem amenable for off-label uses elsewhere (besides constructing lists of theme elements), it is not intended as such. For example, because valid arguments will be deduced from the formals of a function, using certain functions can be troublesome. For example, the `distribute_args` function does not properly recognise the utility of a `...` argument in a function that it is supposed to distribute arguments to. This can be a problem for object-oriented functions: if the methods contain more arguments than the generic itself, these extra arguments will be silently dropped.

See Also

The `element_text()` and `element_rect()` theme elements for a description of their arguments.

Examples

```
# Providing arguments for `element_rect()`
elem_list_rect(
  # The first element_rect will have linetype 1, the second gets 3
  linetype = c(1, 3),
  # If an argument doesn't exist, it will be silently dropped
  nonsense_argument = c("I", "will", "be", "filtered", "out")
)

# Providing arguments for `element_text()`
elem_list_text(
  # `NA`s will be skipped
  family = c("mono", NA, "sans"),
  # Providing a list of more complex arguments. `NULL` will be skipped too.
  margin = list(NULL, margin(t = 5))
)

# Providing arguments to other functions
distribute_args(
  lineend = c("round", "butt", "square"),
  # If you want to pass a vector instead of a scalar, you can use a list
  colour = list(c("blue", "red"), "green"),
  .fun = element_line
)
```

element_part_rect *Partial rectangle theme element*

Description

The element_part_rect() function draws sides of a rectangle as theme elements. It can substitute element_rect() theme elements.

Usage

```
element_part_rect(
  side = "tlbr",
  fill = NULL,
  colour = NULL,
  linewidth = NULL,
  linetype = NULL,
  color = NULL,
  inherit.blank = FALSE
)
```

Arguments

side	A character of length one containing any of "t", "l", "b", "r". If these letters are present it will draw an edge at the top (t), left (l), bottom (b) or right (r) respectively. Including all or none of these letters will default to normal element_rect().
fill	Fill colour.
colour, color	Line/border colour. Color is an alias for colour.
linewidth	Line/border size in mm.
linetype	Line type. An integer (0:8), a name (blank, solid, dashed, dotted, dotdash, longdash, twodash), or a string with an even number (up to eight) of hexadecimal digits which give the lengths in consecutive positions in the string.
inherit.blank	Should this element inherit the existence of an element_blank among its parents? If TRUE the existence of a blank element among its parents will cause this element to be blank as well. If FALSE any blank parent element will be ignored when calculating final element state.

Value

An S3 object of class element_part_rect.

Examples

```
ggplot(iris, aes(Sepal.Width, Sepal.Length)) +
  geom_point() +
  facet_grid(Species ~.) +
```

```
theme(
  strip.background = element_part_rect(side = "tb", colour = "black"),
  panel.background = element_part_rect(side = "l", colour = "black")
)
```

`facetted_pos_scales` *Set individual scales in facets*

Description

This function allows the tweaking of the position scales (x and y) of individual facets. You can use it to fine-tune limits, breaks and other scale parameters for individual facets, provided the facet allows free scales.

Usage

```
facetted_pos_scales(x = NULL, y = NULL)
```

Arguments

x, y	A `list` wherein elements are either x/y position scales or `NULL`s. Alternatively, a list of formulae (see details).

Details

It is intended that this function works with both `ggplot2::facet_wrap()` and `ggplot2::facet_grid()`. For `facet_wrap`, the scales are used for each individual panel. For `facet_grid`, the scales are used for the rows and columns. Note that these facets must be used with `scales = "free"` or `"free_x"` or `"free_y"`, depending on what scales are added.

Axis titles are derived from the first scale in the list (or the default position scale when the first list element is `NULL`).

Scale transformations: It is allowed to use individual scale transformations for facets, but this functionality comes with the trade-off that the out of bounds (`oob`) argument for individual scales is ignored. Values that are out of bounds will be clipped. Whereas the `stat` part of a ggplot layer is typically calculated after scale transformations, the calculation of the `stat` happens before scale transformation with this function, which can lead to some awkward results. The suggested workaround is to pre-transform the data for layers with non-identity `stat` parts.

Scale list input: `NULL`s are valid list elements and signal that the default position scale should be used at the position in the list where the `NULL` occurs. Since transformations are applied before facet scales are initiated, it is not recommended to use a default position (either the first in the list, or defined outside `facetted_pos_scales()`) scale with a transformation other than `trans = "identity"` (the default).

Formula list input: The `x` and `y` arguments also accept a list of two-sided formulas. The left hand side of a formula should evaluate to a `logical` vector. The right hand side of the formula should evaluate to a position scale, wherein the `x` argument accepts x-position scales and the `y` argument accepts y-position scales. Notably, the left hand side of the formula is evaluated using the tidy evaluation framework, whereby the `data.frame` with the plot's layout is given priority over the environment in which the formula was created. As a consequence, variables (columns) that define faceting groups can be references directly.

Value

A *facetted_pos_scales* object, instructing a ggplot how to adjust the scales per facet.

See Also

`ggplot2::scale_x_continuous()` and `scale_x_discrete`.

Examples

```
plot <- ggplot(iris, aes(Sepal.Width, Sepal.Length)) +
  geom_point(aes(colour = Species)) +
  facet_wrap(Species ~ ., scales = "free_y")

# Reversing the y-axis in the second panel. When providing a list of scales,
# NULL indicates to use the default, global scale
plot +
  facetted_pos_scales(
    y = list(NULL, scale_y_continuous(trans = "reverse"))
  )

# Alternative for specifying scales with formula lists. The LHS can access
# columns in the plot's layout.
plot +
  facetted_pos_scales(
    y = list(
      Species == "virginica" ~ scale_y_continuous(breaks = c(6, 7)),
      Species == "versicolor" ~ scale_y_reverse()
    )
  )
```

`facet_grid2` *Extended grid facets*

Description

This function behaves like ggplot2::facet_grid with default arguments, but has a few extra options. It can draw partial or full axis guides at inner panels, and position scales can be independent.

Usage

```
facet_grid2(
  rows = NULL,
  cols = NULL,
  scales = "fixed",
  space = "fixed",
  axes = "margins",
  remove_labels = "none",
  independent = "none",
  shrink = TRUE,
  labeller = "label_value",
  as.table = TRUE,
  switch = NULL,
  drop = TRUE,
  margins = FALSE,
  render_empty = TRUE,
  strip = "vanilla"
)
```

Arguments

rows, cols	A set of variables or expressions quoted by vars() and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to labeller). For compatibility with the classic interface, rows can also be a formula with the rows (of the tabular display) on the LHS and the columns (of the tabular display) on the RHS; the dot in the formula is used to indicate there should be no faceting on this dimension (either row or column).
scales	A character(1) or logical(1) whether scales are shared across facets or allowed to vary. Interacts with the independent argument. One of the following: "fixed" **or** FALSE Scales are shared across all facets (default). "free_x" x-scales are allowed to vary across rows. "free_y" y-scales are allowed to vary across columns. "free" **or** TRUE Scales can vary across rows and columns.
space	A character(1) or logical(1) determining whether the size of panels are proportional to the length of the scales. When the independent argument allows for free scales in a dimension, the panel sizes cannot be proportional. Note that the scales argument must be free in the same dimension as the space argument to have an effect.One of the following: "fixed" **or** FALSE All panels have the same size (default). "free_x" Panel widths are proportional to the x-scales. "free_y" Panel heights are proportional to the y-scales. "free" **or** TRUE Both the widths and heights vary according to scales.
axes	A character(1) or logical(1) where axes should be drawn. One of the following:

"margins" **or** FALSE Only draw axes at the outer margins (default).

"x" Draw axes at the outer margins and all inner x-axes too.

"y" Draw axes at the outer margins and all inner y-axes too.

"all" **or** TRUE Draw the axes for every panel.

`remove_labels` A `character(1)` or `logical(1)` determining whether axis text is displayed at inner panels. One of the following:

"none" **or** FALSE Display axis text at all axes (default).

"x" Display axis text at outer margins and all inner y-axes.

"y" Display axis text at outer margins and all inner x-axes.

"all" **or** TRUE Only display axis text at the outer margins.

`independent` A `character(1)` or `logical(1)` determining whether scales can vary within a row or column of panels, like they can be in ggplot2::facet_wrap. The `scales` argument must be free for the same dimension before they can be set to independent. One of the following:

"none" **or** FALSE All y-scales should be fixed in a row and all x-scales are fixed in a column (default).

"x" x-scales are allowed to vary within a column.

"y" y-scales are allowed to vary within a row.

"all" **or** TRUE Both x- and y-scales are allowed to vary within a column or row respectively.

`shrink` If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.

`labeller` A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with `vars(cyl, am)`. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with `labeller()`. You can use different labeling functions for different kind of labels, for example use `label_parsed()` for formatting facet labels. `label_value()` is used by default, check it for more details and pointers to other options.

`as.table` If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.

`switch` By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".

`drop` If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.

`margins` Either a logical value or a character vector. Margins are additional facets which contain all the data for each of the possible values of the faceting variables. If FALSE, no additional facets are included (the default). If TRUE, margins are included for all faceting variables. If specified as a character vector, it is the names of variables for which margins are to be created.

render_empty — A logical(1): whether to draw panels without any data (TRUE, default) or display these as blanks (FALSE).

strip — A strip specification as one of the following:

- An object inheriting from <Strip>, such as an object created with strip_vanilla().
- A strip function, i.e. strip_vanilla.
- A string giving such function without the strip_-prefix, i.e. "vanilla".

An object created by a call to a strip function, such as strip_vanilla.

Details

Both the independent and space arguments only have an effect when the scales argument in a dimension is free. However, the independent and space arguments can *not* be used to simultaneously set an independent scale and have the panel size be proportional to that scale.

Value

A Facet ggproto object that can be added to a plot.

See Also

Other facetting functions: facet_manual(), facet_nested(), facet_nested_wrap(), facet_wrap2()

Examples

```
p <- ggplot(mpg, aes(displ, hwy)) + geom_point()

# Repeat all axes for every facet
p + facet_grid2(cyl ~ drv, axes = "all")

# Repeat only y-axes
p + facet_grid2(cyl ~  drv, axes = "y")

# Repeat axes without x-labels
p + facet_grid2(cyl ~ drv, axes = "all", remove_labels = "x")

# Grid facets with independent axes for every panel
p + facet_grid2(cyl ~ drv, scales = "free", independent = "all")
```

---

facet_manual *Manual layout for panels*

Description

In facet_manual() the layout for panels is determined by a custom design. Inspired by base-R graphics layout() function, this variant of facets offers more freedom in how panels are displayed, but comes with less guarantees that it looks right.

Usage

```
facet_manual(
  facets,
  design = NULL,
  widths = NULL,
  heights = NULL,
  respect = FALSE,
  drop = TRUE,
  strip.position = "top",
  scales = "fixed",
  axes = "margins",
  remove_labels = "none",
  labeller = "label_value",
  trim_blank = TRUE,
  strip = "vanilla"
)
```

Arguments

facets	A set of variables or expressions quoted by vars() and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to labeller). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, ~a + b, or a character vector, c("a", "b").
design	Specification of panel areas in the layout. Can either be specified as a character(1) string or as a matrix. See examples.
widths, heights	A numeric or unit vector setting the sizes of panels. A numeric vector is converted to relative "null" units. Alternatively, when NULL (default), the sizes are set per instructions of coord or theme aspect ratio. Note that these widths and heights apply to the cells where panels can be drawn. In between such cells, room will be made to fit plot decoration such as paddings, axes and strips.
respect	A logical(1). If TRUE, widths and heights specified in "null" units are proportional. If FALSE, "null" units in the x- and y-directions can vary independently. Alternatively, when NULL, the respect parameter takes instructions from the coord or theme.
drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
strip.position	By default, the labels are displayed on the top of the plot. Using strip.position it is possible to place the labels on either of the four sides by setting strip.position = c("top", "bottom", "left", "right")
scales	A character(1) or logical(1) whether scales are shared across facets or allowed to vary. One of the following: "fixed" **or** FALSE Scales are shared across all facets (default). "free_x" x-scales are allowed to vary.

"free_y" y-scales are allowed to vary.

"free" **or** TRUE Both scales can vary

axes A character(1) or logical(1) where axes should be drawn. One of the following:

"margins" **or** FALSE Only draw axes at the outer margins (default).

"x" Draw axes at the outer margins and all inner x-axes too.

"y" Draw axes at the outer margins and all inner y-axes too.

"all" **or** TRUE Draw the axes for every panel.

remove_labels A character(1) or logical(1) determining whether axis text is displayed at inner panels. One of the following:

"none" **or** FALSE Display axis text at all axes (default).

"x" Display axis text at outer margins and all inner y-axes.

"y" Display axis text at outer margins and all inner x-axes.

"all" **or** TRUE Only display axis text at the outer margins.

labeller A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with vars(cyl, am). Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). You can use different labeling functions for different kind of labels, for example use label_parsed() for formatting facet labels. label_value() is used by default, check it for more details and pointers to other options.

trim_blank A logical(1). When TRUE (default), the design will be trimmed to remove empty rows and columns.

strip A strip specification as one of the following:

- An object inheriting from <Strip>, such as an object created with strip_vanilla().
- A strip function, i.e. strip_vanilla.
- A string giving such function without the strip_-prefix, i.e. "vanilla".

Value

A Facet ggproto object that can be added to a plot.

See Also

Other facetting functions: facet_grid2(), facet_nested(), facet_nested_wrap(), facet_wrap2()

Examples

```
# A standard plot
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point()

# The `design` argument can be a character string.
# New rows are indicated by newline symbol (`\n`), which are added
# automatically for multi-line strings.
```

```
# The `#`-symbol indicates empty cells.
design <- "
 A##
 AB#
 #BC
 ##C
"
p + facet_manual(~ cyl, design)

# Alternatively, the `design` argument can be a matrix.
# Using `NA`s will leave the cell empty.
design <- matrix(c(1,2,3,3), 2, 2, byrow = TRUE)
p + facet_manual(~ cyl, design)

# The sizes of columns and rows can be adjusted with the `widths` and
# `heights`parameters respectively.
p + facet_manual(
  ~ cyl, t(design),
  widths = c(2, 1), heights = c(2, 1), respect = TRUE
)
```

---

facet_nested *Layout panels in a grid with nested strips*

Description

facet_nested() forms a matrix of panels defined by row and column faceting variables and nests grouped facets.

Usage

```
facet_nested(
  rows = NULL,
  cols = NULL,
  scales = "fixed",
  space = "fixed",
  axes = "margins",
  remove_labels = "none",
  independent = "none",
  shrink = TRUE,
  labeller = "label_value",
  as.table = TRUE,
  switch = NULL,
  drop = TRUE,
  margins = FALSE,
  nest_line = element_line(inherit.blank = TRUE),
  solo_line = FALSE,
  resect = unit(0, "mm"),
  render_empty = TRUE,
```

```
  strip = "nested",
  bleed = NULL
)
```

Arguments

rows, cols	A set of variables or expressions quoted by vars() and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to labeller). For compatibility with the classic interface, rows can also be a formula with the rows (of the tabular display) on the LHS and the columns (of the tabular display) on the RHS; the dot in the formula is used to indicate there should be no faceting on this dimension (either row or column).
scales	A character(1) or logical(1) whether scales are shared across facets or allowed to vary. Interacts with the independent argument. One of the following: "fixed" **or** FALSE Scales are shared across all facets (default). "free_x" x-scales are allowed to vary across rows. "free_y" y-scales are allowed to vary across columns. "free" **or** TRUE Scales can vary across rows and columns.
space	A character(1) or logical(1) determining whether the size of panels are proportional to the length of the scales. When the independent argument allows for free scales in a dimension, the panel sizes cannot be proportional. Note that the scales argument must be free in the same dimension as the space argument to have an effect.One of the following: "fixed" **or** FALSE All panels have the same size (default). "free_x" Panel widths are proportional to the x-scales. "free_y" Panel heights are proportional to the y-scales. "free" **or** TRUE Both the widths and heights vary according to scales.
axes	A character(1) or logical(1) where axes should be drawn. One of the following: "margins" **or** FALSE Only draw axes at the outer margins (default). "x" Draw axes at the outer margins and all inner x-axes too. "y" Draw axes at the outer margins and all inner y-axes too. "all" **or** TRUE Draw the axes for every panel.
remove_labels	A character(1) or logical(1) determining whether axis text is displayed at inner panels. One of the following: "none" **or** FALSE Display axis text at all axes (default). "x" Display axis text at outer margins and all inner y-axes. "y" Display axis text at outer margins and all inner x-axes. "all" **or** TRUE Only display axis text at the outer margins.
independent	A character(1) or logical(1) determining whether scales can vary within a row or column of panels, like they can be in ggplot2::facet_wrap. The scales argument must be free for the same dimension before they can be set to independent. One of the following:

"none" **or** FALSE All y-scales should be fixed in a row and all x-scales are fixed in a column (default).

"x" x-scales are allowed to vary within a column.

"y" y-scales are allowed to vary within a row.

"all" **or** TRUE Both x- and y-scales are allowed to vary within a column or row respectively.

`shrink` If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.

`labeller` A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with `vars(cyl, am)`. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with `labeller()`. You can use different labeling functions for different kind of labels, for example use `label_parsed()` for formatting facet labels. `label_value()` is used by default, check it for more details and pointers to other options.

`as.table` If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.

`switch` By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".

`drop` If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.

`margins` Either a logical value or a character vector. Margins are additional facets which contain all the data for each of the possible values of the faceting variables. If FALSE, no additional facets are included (the default). If TRUE, margins are included for all faceting variables. If specified as a character vector, it is the names of variables for which margins are to be created.

`nest_line` a theme element, either `element_blank()` or inheriting from `ggplot2::element_line()`. Lines are drawn between layers of strips indicating hierarchy. The element inherits from the `ggh4x.facet.nestline` element in the theme.

`solo_line` A `logical(1)` indicating whether parent strips with a single child should be drawn with a `nest_line` (TRUE) or the line only applies to parents with multiple children (FALSE, default). Only relevant when `nest_line` is drawn.

`resect` a `unit` vector of length 1, indicating how much the nesting line should be shortened.

`render_empty` A `logical(1)`: whether to draw panels without any data (TRUE, default) or display these as blanks (FALSE).

`strip` An object created by a call to a strip function, such as `strip_nested()`.

`bleed` **[Deprecated]** the `bleed` argument has moved to the `strip_nested()` function.

Details

This function inherits the capabilities of `facet_grid2()`.

Unlike `facet_grid()`, this function only automatically expands missing variables when they have no variables in that direction, to allow for unnested variables. It still requires at least one layer to have all faceting variables.

Hierarchies are inferred from the order of variables supplied to `rows` or `cols`. The first variable is interpreted to be the outermost variable, while the last variable is interpreted to be the innermost variable. They display order is always such that the outermost variable is placed the furthest away from the panels. For more information about the nesting of strips, please visit the documentation of `strip_nested()`.

Value

A *FacetNested* ggproto object that can be added to a plot.

See Also

See `strip_nested()` for nested strips. See `ggplot2::facet_grid()` for descriptions of the original arguments. See `grid::unit()` for the construction of a `unit` vector.

Other facetting functions: `facet_grid2()`, `facet_manual()`, `facet_nested_wrap()`, `facet_wrap2()`

Examples

```
# A standard plot
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point()

# Similar to `facet_grid2(..., strip = strip_nested())`
p + facet_nested(~ vs + cyl)

# The nest line inherits from the global theme
p + facet_nested(~ cyl + vs, nest_line = element_line(colour = "red")) +
  theme(ggh4x.facet.nestline = element_line(linetype = 3))
```

---

facet_nested_wrap *Ribbon of panels with nested strips.*

Description

`facet_nested_wrap()` wraps a sequence of panels onto a two-dimensional layout, and nests grouped facets where possible.

Usage

```
facet_nested_wrap(
  facets,
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  axes = "margins",
  remove_labels = "none",
  shrink = TRUE,
  labeller = "label_value",
  as.table = TRUE,
  drop = TRUE,
  dir = "h",
  strip.position = "top",
  nest_line = element_line(inherit.blank = TRUE),
  solo_line = FALSE,
  resect = unit(0, "mm"),
  trim_blank = TRUE,
  strip = "nested",
  bleed = NULL
)
```

Arguments

facets	A set of variables or expressions quoted by vars() and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to labeller). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, ~a + b, or a character vector, c("a", "b").
nrow, ncol	Number of rows and columns.
scales	A character(1) or logical(1) whether scales are shared across facets or allowed to vary. One of the following: "fixed" **or** FALSE Scales are shared across all facets (default). "free_x" x-scales are allowed to vary. "free_y" y-scales are allowed to vary. "free" **or** TRUE Both scales can vary
axes	A character(1) or logical(1) where axes should be drawn. One of the following: "margins" **or** FALSE Only draw axes at the outer margins (default). "x" Draw axes at the outer margins and all inner x-axes too. "y" Draw axes at the outer margins and all inner y-axes too. "all" **or** TRUE Draw the axes for every panel.
remove_labels	A character(1) or logical(1) determining whether axis text is displayed at inner panels. One of the following:

"none" **or** FALSE Display axis text at all axes (default).

"x" Display axis text at outer margins and all inner y-axes.

"y" Display axis text at outer margins and all inner x-axes.

"all" **or** TRUE Only display axis text at the outer margins.

`shrink` If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.

`labeller` A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with `vars(cyl, am)`. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with `labeller()`. You can use different labeling functions for different kind of labels, for example use `label_parsed()` for formatting facet labels. `label_value()` is used by default, check it for more details and pointers to other options.

`as.table` If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.

`drop` If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.

`dir` Direction: either "h" for horizontal, the default, or "v", for vertical.

`strip.position` By default, the labels are displayed on the top of the plot. Using `strip.position` it is possible to place the labels on either of the four sides by setting `strip.position = c("top", "bottom", "left", "right")`

`nest_line` a theme element, either `element_blank()` or inheriting from `ggplot2::element_line()`. Lines are drawn between layers of strips indicating hierarchy. The element inherits from the `ggh4x.facet.nestline` element in the theme.

`solo_line` A `logical(1)` indicating whether parent strips with a single child should be drawn with a `nest_line` (TRUE) or the line only applies to parents with multiple children (FALSE, default). Only relevant when `nest_line` is drawn.

`resect` a `unit` vector of length 1, indicating how much the nesting line should be shortened.

`trim_blank` A `logical(1)`. When TRUE (default), does not draw rows and columns containing no panels. When FALSE, the `nrow` and `ncol` arguments are taken literally, even when there are more than needed to fit all panels.

`strip` An object created by a call to a strip function, such as `strip_nested()`.

`bleed` **[Deprecated]** the `bleed` argument has moved to the `strip_nested()` function.

Details

This function inherits the capabilities of `facet_wrap2()`.

This function only merges strips in the same row or column as they appear through regular `facet_wrap()` layout behaviour.

Hierarchies are inferred from the order of variables supplied to `facets`. The first variable is interpreted to be the outermost variable, while the last variable is interpreted to be the innermost variable. They display order is always such that the outermost variable is placed the furthest away from the panels. For more information about the nesting of strips, please visit the documentation of `strip_nested()`.

Value

A `FacetNestedWrap` ggproto object that can be added to a plot.

See Also

See `strip_nested()` for nested strips. See `ggplot2::facet_wrap()` for descriptions of the original arguments. See `grid::unit()` for the construction of a `unit` vector.

Other facetting functions: `facet_grid2()`, `facet_manual()`, `facet_nested()`, `facet_wrap2()`

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()

# Similar to `facet_wrap2(..., strip = strip_nested())`.
p + facet_nested_wrap(vars(cyl, drv))

# A nest line inherits from the global theme
p + facet_nested_wrap(vars(cyl, drv),
                      nest_line = element_line(colour = "red")) +
  theme(ggh4x.facet.nestline = element_line(linetype = 3))
```

---

facet_wrap2 *Extended wrapped facets*

Description

This function behaves like `ggplot2::facet_wrap()`, but has a few extra options on axis drawing when scales are fixed.

Usage

```
facet_wrap2(
  facets,
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  axes = "margins",
  remove_labels = "none",
  shrink = TRUE,
```

```
  labeller = "label_value",
  as.table = TRUE,
  drop = TRUE,
  dir = "h",
  strip.position = "top",
  trim_blank = TRUE,
  strip = "vanilla"
)
```

Arguments

facets	A set of variables or expressions quoted by vars() and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to labeller). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, ~a + b, or a character vector, c("a", "b").
nrow, ncol	Number of rows and columns.
scales	A character(1) or logical(1) whether scales are shared across facets or allowed to vary. One of the following: "fixed" **or** FALSE Scales are shared across all facets (default). "free_x" x-scales are allowed to vary. "free_y" y-scales are allowed to vary. "free" **or** TRUE Both scales can vary
axes	A character(1) or logical(1) where axes should be drawn. One of the following: "margins" **or** FALSE Only draw axes at the outer margins (default). "x" Draw axes at the outer margins and all inner x-axes too. "y" Draw axes at the outer margins and all inner y-axes too. "all" **or** TRUE Draw the axes for every panel.
remove_labels	A character(1) or logical(1) determining whether axis text is displayed at inner panels. One of the following: "none" **or** FALSE Display axis text at all axes (default). "x" Display axis text at outer margins and all inner y-axes. "y" Display axis text at outer margins and all inner x-axes. "all" **or** TRUE Only display axis text at the outer margins.
shrink	If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
labeller	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with vars(cyl, am). Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). You can use different labeling functions for different kind of labels, for example use label_parsed() for formatting facet labels. label_value() is used by default, check it for more details and pointers to other options.

as.table	If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.
drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
dir	Direction: either "h" for horizontal, the default, or "v", for vertical.
strip.position	By default, the labels are displayed on the top of the plot. Using strip.position it is possible to place the labels on either of the four sides by setting strip.position = c("top", "bottom", "left", "right")
trim_blank	A logical(1). When TRUE (default), does not draw rows and columns containing no panels. When FALSE, the nrow and ncol arguments are taken literally, even when there are more than needed to fit all panels.
strip	A strip specification as one of the following: • An object inheriting from <Strip>, such as an object created with strip_vanilla(). • A strip function, i.e. strip_vanilla. • A string giving such function without the strip_-prefix, i.e. "vanilla".

Value

A Facet ggproto object that can be added to a plot.

See Also

Other facetting functions: facet_grid2(), facet_manual(), facet_nested(), facet_nested_wrap()

Examples

```
p <- ggplot(mpg, aes(displ, hwy)) + geom_point()

# Repeat all axes for every facet
p + facet_wrap2(vars(class), axes = "all")

# Repeat only y-axes
p + facet_wrap2(vars(class), axes = "y")

# Repeat axes without labels
p + facet_wrap2(vars(class), axes = "all", remove_labels = "all")

# Repeat axes without x-axis labels
p + facet_wrap2(vars(class), axes = "all", remove_labels = "x")
```

`force_panelsizes` *Force a facetted plot to have specified panel sizes*

Description

Takes a ggplot and modifies its facet drawing behaviour such that the widths and heights of panels are set by the user.

Usage

```
force_panelsizes(
  rows = NULL,
  cols = NULL,
  respect = NULL,
  total_width = NULL,
  total_height = NULL
)
```

Arguments

`rows`, `cols`	a `numeric` or `unit` vector for setting panel heights (rows) or panel widths (cols).
`respect`	a `logical` value. If `TRUE`, widths and heights specified in `"null" units` are proportional. If `FALSE`, `"null" units` in x- and y-direction vary independently.
`total_width`, `total_height`	an absolute `unit` of length 1 setting the total width or height of all panels and the decoration between panels. If not `NULL`, `rows` and `cols` should be `numeric` and not `units`.

Details

Forcing the panel sizes should in theory work regardless of what facetting choice was made, as long as this function is called after the facet specification. Even when no facets are specified, ggplot2 defaults to the `ggplot2::facet_null()` specification; a single panel. `force_panelsizes` works by wrapping the original panel drawing function inside a function that modifies the widths and heights of panel grobs in the original function's output gtable.

When `rows` or `cols` are `numeric` vectors, panel sizes are defined as ratios i.e. relative `"null" units`. `rows` and `cols` vectors are repeated or shortened to fit the number of panels in their direction. When `rows` or `cols` are `NULL`, no changes are made in that direction.

When `respect = NULL`, default behaviour specified elsewhere is inherited.

No attempt is made to guarantee that the plot fits the output device. The `space` argument in `ggplot2::facet_grid()` will be overruled. When individual panels span multiple rows or columns, this function may not work as intended.

Value

A `forcedsize` S3 object that can be added to a plot.

See Also

ggplot2::facet_grid() ggplot2::facet_wrap() ggplot2::facet_null() grid::unit()

Examples

```
ggplot(mtcars, aes(disp, mpg)) +
  geom_point() +
  facet_grid(vs ~ am) +
  force_panelsizes(rows = c(2, 1),
                   cols = c(2, 1))
```

---

geom_box *Flexible rectangles*

Description

The geom_box() function offers a more flexible variant of geom_rect() and geom_tile(). Instead of exclusively working with the (x/y)min/(x/y)max *or* (x/y)/(width/height) aesthetics, any two out of these four aesthetics suffice to define a rectangle.

Usage

```
geom_box(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  radius = 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 the 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.

linejoin — Line join style (round, mitre, bevel).

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.
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. borders().
radius	A grid::unit object of length 1 or numeric(1) to set rounded corners. If NULL (default), no rounded corners are applied. If numeric(1), it is interpreted as millimetres. Does not work under non-linear coordinates.

Value

A ggplot2 Layer object that can be added to a plot.

Examples

```
# Combine any two position aesthetics
df <- data.frame(
  x = c(1.5, 3.5), xmin = c(1, 2),
  y = c(1.5, 2.5), ymin = c(1, 2)
)
ggplot(df) +
  geom_box(aes(x = x, xmin = xmin, y = y, ymin = ymin))

# Works with partial information for position, as long as two aesthetics
# are complete for any observation.
df <- data.frame(
  x = c(1.5, NA, 4),   xmin = c(1, 2, NA), width = c(NA, 3, 2),
  y = c(1.5, 2.5, NA), ymin = c(NA, 2, 3), height = c(1, NA, 3)
)
ggplot(df) +
  geom_box(aes(x = x, xmin = xmin, y = y, ymin = ymin,
               width = width, height = height))

# Set radius for rounded corners
ggplot() +
  geom_box(
    aes(x = 1:3, width = rep(1, 3),
        y = 1:3, height = 3:1),
    radius = 5
  )
```

---

geom_outline_point *Points with outline*

Description

This is a variant of the point geom, wherein overlapping points are given a shared outline. It works by drawing an additional layer of points below a regular layer of points with a thicker stroke.

Usage

```
geom_outline_point(
  mapping = NULL,
  data = NULL,
  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 the 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.

`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.

`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. `borders()`.

Details

Due to the way this geom is implemented, it handles the `alpha` aesthetic pretty ungracefully.

Value

A ggplot `Layer`

Aesthetics

`geom_outline_point()` understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- colour
- fill
- group
- shape
- size
- stroke
- stroke_colour

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

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, cty, colour = factor(cyl))) +
  geom_outline_point(size = 10, stroke = 3)
p

# The colour of the stroke can be mapped to a scale by setting the
# aesthetics to `"stroke_colour"`.
p +
  aes(stroke_colour = factor(cyl)) +
  scale_colour_hue(
    aesthetics = "stroke_colour",
    l = 50
  )
```

`geom_pointpath` *Point Paths*

Description

The point path geom is used to make a scatterplot wherein the points are connected with lines in some order. This geom intends to mimic the `type = 'b'` style of base R line plots.

Usage

```
geom_pointpath(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
```

```
  show.legend = NA,
  arrow = NULL,
  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 the 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.

`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.

`arrow` Arrow specification as created by `grid::arrow()`.

`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. `borders()`.

Details

The `mult` is a numeric value to scale the proportion of gaps in the line around points.

While the need for this geom is not very apparent, since it can be approximated in a variety of ways, the trick up its sleeve is that it dynamically adapts the inter-point segments so these don't deform under different aspect ratios or device sizes.

Value

A *Layer* ggproto object.

Aesthetics

`geom_pointpath()` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `alpha`
- `colour`
- `group`
- `shape`

- size
- stroke
- linewidth
- linetype
- mult

Examples

```
ggplot(pressure, aes(temperature, pressure)) +
  geom_pointpath()

# Using geom_pointpath as annotation
ggplot() +
  annotate(
    "pointpath",
    x = c(1, 0.32, 0.31, -0.12, -0.81, -0.4, -0.81, -0.12, 0.31, 0.32, 1),
    y = c(0, 0.24, 0.95, 0.38, 0.59, 0, -0.59, -0.38, -0.95, -0.24, 0)
  )
```

---

geom_polygonraster *Polygon parameterisation for rasters*

Description

geom_polygonraster takes data that describes a raster with pixels of the same size and reparametrises the data as a polygon. This allows for more flexible transformations of the data, but comes at an efficiency cost.

Usage

```
geom_polygonraster(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = position_lineartrans(),
  ...,
  hjust = 0.5,
  vjust = 0.5,
  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 the 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.

`hjust, vjust` horizontal and vertical justification of the grob. Each justification value should be a number between 0 and 1. Defaults to 0.5 for both, centering each pixel over its data location.

`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.

`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. `borders()`.

Details

For each pixel in a raster, makes a vertex for each of the four corner points. These coordinates can then by transformed by `coord`-functions such as `ggplot2::coord_polar()` or `position`-functions such as `position_lineartrans()`. Currently substitutes group aesthetics right before drawing in favour of pixel identifiers.

Value

A *Layer* ggproto object.

Aesthetics

`geom_raster()` understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- fill
- alpha
- group

See Also

`geom_raster()`

Examples

```
# Combining with coord_polar()
ggplot(faithfuld, aes(waiting, eruptions)) +
  geom_polygonraster(aes(fill = density)) +
  coord_polar()

# Combining with linear transformations
df <- data.frame(x = row(volcano)[TRUE],
                 y = col(volcano)[TRUE],
                 z = volcano[TRUE])

ggplot(df, aes(x, y, fill = z)) +
  geom_polygonraster(position = position_lineartrans(angle = 30,
                                                     shear = c(1, 0)))
```

---

geom_rectmargin *Rectangular rugs in the margins*

Description

Like rug plots display data points of a 2D plot as lines in the margins, this function plots rectangles in the margins. Rectangular rugs are convenient for displaying one-dimensional, ranged annotations for two-dimensional plots.

Usage

```
geom_rectmargin(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  outside = FALSE,
  sides = "bl",
  length = unit(0.03, "npc"),
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_tilemargin(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
```

```
  outside = FALSE,
  sides = "bl",
  length = unit(0.03, "npc"),
  linejoin = "mitre",
  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 the 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.

`outside` `logical` of length 1 that controls whether to move the rectangles outside of the plot area. For the best results, it is probably best to set `coord_cartesian(clip = "off")` and avoid overlap with the default axes by changing the sides argument to `"tr"`.

`sides` A `string` of length 1 that controls which sides of the plot the rug-rectangles appear on. A string containing any letters in `"trbl"` will set it to top, right, bottom and left respectively.

`length` A `grid::unit()` object that sets the width and height of the rectangles in the x- and y-directions respectively. Note that scale expansion can affect the look of this.

`linejoin` Line join style (round, mitre, bevel).

`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.

`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. `borders()`.

Details

By default, scales are expanded 5\ whereas the rug rectangles will occupy 3\ default. The `geom_rectmargin()` and `geom_tilemargin()` versions do the same thing, but are parametrised differently; see `geom_rect()`.

These functions do not have hard-coded required aesthetics, since the x and y directions can be omitted by not choosing a side in the corresponding direction, i.e. y-direction variables are omitted when plotting the rug only on the top and/or bottom. This can result in errors when the aesthetics are not specified appropriately, so some caution is advised.

Value

A *Layer* ggproto object.

Aesthetics

`geom_rectmargin()` requires either one of the following sets of aesthetics, but also can use both:

- **xmin**
- **xmax**

and/or:

- **ymin**
- **ymax**

`geom_tilemargin()` requires either one of the following sets of aesthetics, but can also use both:

- **x**
- **width**

and/or:

- **y**
- **height**

Furthermore, `geom_rectmargin()` and `geom_tilemargin()` also understand these shared aesthetics:

- alpha
- colour
- fill
- group
- linetype
- size

See Also

`ggplot2::geom_rug()`, `geom_rect()`, `ggplot2::geom_tile()`

Examples

```
# geom_rectmargin() is parameterised by the four corners
df <- data.frame(
  xmin = c(1, 5),
  xmax = c(2, 7),
  ymin = c(1, 2),
  ymax = c(2, 4),
  fill = c("A", "B")
)


ggplot(df, aes(xmin = xmin, xmax = xmax,
               ymin = ymin, ymax = ymax,
               fill = fill)) +
  geom_rect() +
  geom_rectmargin()

# geom_tilemargin() is parameterised by center and size
df <- data.frame(
  x = c(1, 4),
  y = c(1, 2),
  width = c(2, 1),
  height = c(1, 2),
  fill = c("A", "B")
)

ggplot(df, aes(x, y,
               width = width, height = height,
               fill = fill)) +
  geom_tile() +
  geom_tilemargin()
```

---

geom_text_aimed *Aimed text*

Description

Similar to `geom_text()`, this geom also generates text but places the text at an angle so that the text seems aimed towards a point defined by `[xend, yend]`.

Usage

```
geom_text_aimed(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
```

```
  nudge_x = 0,
  nudge_y = 0,
  flip_upsidedown = TRUE,
  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 the 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. Cannot be jointy specified with nudge_x or nudge_y. 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(). • A string nameing 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`.

`nudge_x, nudge_y` Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales. Cannot be jointly specified with `position`.

`flip_upsidedown` A `logical(1)`. If `TRUE` (default), the angle of text placed at angles between 90 and 270 degrees is flipped so that it is more comfortable to read. If `FALSE`, will take calculated angles literally.

`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.

`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. `borders()`.

Details

The calculated angle is such that the text will be parallel to a line passing through the coordinates `[x, y]` and `[xend, yend]`. The calculated angle is added to the `angle` angle aesthetic, so that you can set text perpendicular to that line by setting `angle = 90`. These angles are calculated in absolute coordinates, meaning that resizing the plot will retain the same appearance.

Value

A ggplot2 `Layer` that can be added to a plot.

Aesthetics

`geom_text_aimed()` understands the following aesthetics (required aesthetics are in bold):

- x
- y
- label
- alpha
- angle
- colour
- family
- fontface
- group
- hjust
- lineheight
- size
- vjust
- xend
- yend

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

Note

When using this geom to aim text at the centre of a polar plot, make sure the radius range does not have close to zero width.

Examples

```
# Point all labels to upper right corner
ggplot(mtcars, aes(mpg, wt)) +
  geom_text_aimed(aes(label = rownames(mtcars)),
                  xend = Inf, yend = Inf)

# Point all labels to center of polar plot
ggplot(mpg, aes(manufacturer)) +
  geom_bar(width = 1, aes(fill = manufacturer), show.legend = FALSE) +
  geom_text_aimed(aes(label = manufacturer), hjust = 0,
                  stat = "count", nudge_y = 2) +
  scale_x_discrete(labels = NULL) +
  coord_polar()
```

ggh4x_extensions *ggh4x extensions to ggplot2*

Description

ggh4x relies on the extension mechanism of ggplot2 through ggproto class objects, which allows cross-package inheritance of objects such as geoms, stats, facets, scales and coordinate systems. These objects can be ignored by users for the purpose of making plots, since interacting with these objects is preferred through various geom_, stat_, facet_, coord_ and scale_ functions.

See Also

ggproto

guide_stringlegend *String legend*

Description

This type of legend shows colour and fill mappings as coloured text. It does not draw keys as `guide_legend()` does.

Usage

```
guide_stringlegend(
  title = waiver(),
  theme = NULL,
  position = NULL,
  direction = NULL,
  nrow = NULL,
  ncol = NULL,
  reverse = FALSE,
  order = 0
)
```

Arguments

`title`	A character string or expression indicating a title of guide. If `NULL`, the title is not shown. By default (`waiver()`), the name of the scale object or the name specified in `labs()` is used for the title.
`theme`	A `theme` object to style the guide individually or differently from the plot's theme settings. The `theme` argument in the guide overrides, and is combined with, the plot's theme.
`position`	A character string indicating where the legend should be placed relative to the plot panels.

direction	A character string indicating the direction of the guide. One of "horizontal" or "vertical."
nrow, ncol	The desired number of rows and column of legends respectively.
reverse	logical. If TRUE the order of legends is reversed.
order	positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.

Value

A GuideStringlegend object.

Examples

```
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = manufacturer))

# String legend can be set in the `guides()` function
p + guides(colour = guide_stringlegend(ncol = 2))

# The string legend can also be set as argument to the scale
p + scale_colour_viridis_d(guide = "stringlegend")
```

---

help_secondary *Secondary axis helper*

Description

The purpose of this function is to construct a secondary axis with a projection function.

Usage

```
help_secondary(
  data = NULL,
  primary = c(0, 1),
  secondary = c(0, 1),
  method = c("range", "max", "fit", "ccf", "sortfit"),
  na.rm = TRUE,
  ...
)
```

Arguments

`data`	A `data.frame` object.
`primary, secondary`	An expression that is evaluated in the context of the `data` argument. These can be symbols for column names or plain expressions.
`method`	One of the following: `"range"` Causes the ranges of `primary` and `secondary` data to overlap completely. `"max"` Causes the maxima of `primary` and `secondary` data to coincide. `"fit"` Uses the coefficients of `lm(primary ~ secondary)` to make the axes fit. `"ccf"` Uses the lag at which maximum cross-correlation occurs to then align the data by truncation. The aligned data is then passed to the `"fit"` method. `"sortfit"` Sorts the both `primary` and `secondary` independently before passing these on to the `"fit"` method.
`na.rm`	A `logical(1)`: whether to remove missing values (`TRUE`) or propagate missing values (`FALSE`). Applies to the `method = "range"` and `method = "max"` methods.
`...`	Arguments passed on to `ggplot2::sec_axis` `transform` A formula or function of a strictly monotonic transformation `name` The name of the secondary axis `breaks` One of: • `NULL` for no breaks • `waiver()` for the default breaks computed by the transformation object • A numeric vector of positions • A function that takes the limits as input and returns breaks as output `labels` One of: • `NULL` for no labels • `waiver()` for the default labels computed by the transformation object • A character vector giving labels (must be same length as `breaks`) • A function that takes the breaks as input and returns labels as output `guide` A position guide that will be used to render the axis on the plot. Usually this is `guide_axis()`.

Details

The intent is to run this function before starting a plot. The output of the function is a secondary axis wherein the `trans` argument of `sec_axis()` is populated by an appropriate transformation. In addition, the output also contains a `output$proj()` function that helps transform the secondary data.

Value

An `AxisSecondary` ggproto object with a `proj` method for projecting secondary data.

Examples

```
# Run the secondary axis helper
sec <- help_secondary(economics, primary = unemploy, secondary = psavert)

# Making primary plot
p <- ggplot(economics, aes(date)) +
  geom_line(aes(y = unemploy), colour = "blue")

# For the secondary data, later we use the `proj` function from the helper
p <- p + geom_line(aes(y = sec$proj(psavert)), colour = "red")

# We feed the scale the secondary axis
p + scale_y_continuous(sec.axis = sec)

# Setup cross-correlated data
set.seed(42)
n <- 100
lag <- 20
dat <- cumsum(rnorm(n + lag))
df <- data.frame(
  x = seq_len(n),
  y1 = head(dat, n),
  y2 = 10 + tail(dat, n) * 5 # offset and scale y2
)
# Choosing the cross-correlation function method.
sec <- help_secondary(df, y1, y2, method = "ccf")

ggplot(df, aes(x)) +
  geom_line(aes(y = y1), colour = "blue") +
  geom_line(aes(y = sec$proj(y2)), colour = "red") +
  scale_y_continuous(sec.axis = sec)
```

---

position_disjoint_ranges

Segregating overlapping ranges

Description

One-dimensional ranged data in the x-direction is segregated in the y-direction such that no overlap in two-dimensional space occurs. This positioning works best when no relevant information is plotted in the y-direction.

Usage

```
position_disjoint_ranges(extend = 1, stepsize = 1)
```

Arguments

extend	a numeric of length 1 indicating how far a range should be extended in total for calculating overlaps. Setting this argument to a positive number leaves some space between ranges in the same bin.
stepsize	a numeric of length 1 that determines how much space is added between bins in the y-direction. A positive value grows the bins from bottom to top, while a negative value grows the bins from top to bottom.

Details

An object is considered disjoint from a second object when the range between their xmin and xmax coordinates don't overlap. Objects that overlap are assigned to different bins in the y-direction, whereby lower bins are filled first. This way, information in the x-direction is preserved and different objects can be discerned.

Note that this positioning is only particularly useful when y-coordinates do not encode relevant information. Geoms that pair well with this positioning are geom_rect() and ggplot2::geom_tile().

This positioning function was inspired by the disjointBins() function in the IRanges package, but has been written such that it accepts any numeric input next to solely integer input.

Value

A *PositionDisjointRanges* object.

See Also

The disjointBins function the Bioconductor IRanges package.

Examples

```
# Even though geom_tile() is parametrised by middle-x values, it is
# internally converted to xmin, xmax, ymin, ymax parametrisation so the
# positioning still works.

ggplot() +
  geom_tile(aes(x = rnorm(200), y = 0),
            width = 0.2, height = 0.9,
            position = position_disjoint_ranges(extend = 0.1))
```

---

position_lineartrans *Linearly transform coordinates*

Description

Transforms coordinates in two dimensions in a linear manner for layers that have an x and y parametrisation.

Usage

```
position_lineartrans(scale = c(1, 1), shear = c(0, 0), angle = 0, M = NULL)
```

Arguments

scale	A numeric of length two describing relative units with which to multiply the x and y coordinates respectively.
shear	A numeric of length two giving relative units by which to shear the output. The first number is for vertical shearing whereas the second is for horizontal shearing.
angle	A numeric noting an angle in degrees by which to rotate the input clockwise.
M	A 2 x 2 real matrix: the transformation matrix for linear mapping. Overrides other arguments if provided.

Details

Linear transformation matrices are 2 x 2 real matrices. The 'scale', 'shear' and 'rotation' arguments are convenience arguments to construct a transformation matrix. These operations occur in the order: scaling - shearing - rotating. To apply the transformations in another order, build a custom 'M' argument.

For some common transformations, you can find appropriate matrices for the 'M' argument below.

Value

A *PositionLinearTrans* ggproto object.

Common transformations

Identity transformations: An identity transformation, or returning the original coordinates, can be performed by using the following transformation matrix:

```
| 1 0 |
| 0 1 |
```

or

```
M <- matrix(c(1, 0, 0, 1), 2)
```

Scaling: A scaling transformation multiplies the dimension of an object by some amount. An example transformation matrix for scaling everything by a factor 2:

```
| 2 0 |
| 0 2 |
```

or

```
M <- matrix(c(2, 0, 0, 2), 2)
```

Squeezing: Similar to scaling, squeezing multiplies the dimensions by some amount that is unequal for the x and y coordinates. For example, squeezing y by half and expanding x by two:

```
|     2     0       |
|     0     0.5     |
```

or

```
M <- matrix(c(2, 0, 0, 0.5), 2)
```

Reflection: Mirroring the coordinates around one of the axes. Reflecting around the x-axis:

```
|     1      0     |
|     0     -1     |
```

or

```
M <- matrix(c(1, 0, 0, -1), 2)
```

Reflecting around the y-axis:

```
|     -1     0     |
|      0     1     |
```

or

```
M <- matrix(c(-1, 0, 0, 1), 2)
```

Projection: For projecting the coordinates on one of the axes, while collapsing everything from the other axis. Projecting onto the y-axis:

```
|     0     0     |
|     0     1     |
```

or

```
M <- matrix(c(0, 0, 0, 1), 2)
```

Projecting onto the x-axis:

```
|     1     0     |
|     0     0     |
```

or

```
M <- matrix(c(1, 0, 0, 0), 2)
```

Shearing: Tilting the coordinates horizontally or vertically. Shearing vertically by 10\

```
|     1       0     |
|     0.1     1     |
```

or

```
M <- matrix(c(1, 0.1, 0, 1), 2)
```

Shearing horizontally by 200\

```
|     1     2     |
|     0     1     |
```

or

```
M <- matrix(c(1, 0, 2, 1), 2)
```

Rotation: A rotation performs a motion around a fixed point, typically the origin the coordinate system. To rotate the coordinates by 90 degrees counter-clockwise:

```
|     0     -1     |
|     1      0     |
```

or

```
M <- matrix(c(0, 1, -1, 0), 2)
```

For a rotation around any angle θ :

$$\begin{vmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{vmatrix}$$

or

```
M <- matrix(c(cos(theta), sin(theta), -sin(theta), cos(theta)), 2)
```
with 'theta' defined in radians.

Examples

```
df <- data.frame(x = c(0, 1, 1, 0),
                 y = c(0, 0, 1, 1))
ggplot(df, aes(x, y)) +
  geom_polygon(position = position_lineartrans(angle = 30))
```

```
# Custom transformation matrices
# Rotation
theta <- -30 * pi / 180
rot <- matrix(c(cos(theta), sin(theta), -sin(theta), cos(theta)), 2)
# Shear
shear <- matrix(c(1, 0, 1, 1), 2)

# Shear and then rotate
M <- rot %*% shear
ggplot(df, aes(x, y)) +
  geom_polygon(position = position_lineartrans(M = M))
# Alternative shear and then rotate
ggplot(df, aes(x, y)) +
  geom_polygon(position = position_lineartrans(shear = c(0, 1), angle = 30))

# Rotate and then shear
M <- shear %*% rot
ggplot(df, aes(x, y)) +
  geom_polygon(position = position_lineartrans(M = M))
```

`scale_facet` *Position scales for individual panels in facets*

Description

This function adds position scales (x and y) of individual panels. These can be used to fine-tune limits, breaks and other scale parameters for individual panels, provided the facet allows free scales.

Usage

```
scale_x_facet(expr, ..., type = "continuous")

scale_y_facet(expr, ..., type = "continuous")
```

Arguments

`expr`	An `expression` that, when evaluated in the facet's layout data.frame, yields a `logical` vector. See details.
`...`	Other arguments passed to the scale.
`type`	A `character(1)` indicating the type of scale, such that `scale_(x/y)_{type}` spells a scale function. Defaults to `"continuous"`.

Details

These scale functions work through the mechanism of the `facetted_pos_scales()` function, and the same limitations apply: scale transformations are applied after `stat` transformations, and the `oob` argument of scales is ignored.

For the expr argument, the expression will be evaluated in the context of the plot's layout. This is an internal data.frame structure that isn't normally exposed, so it requires some extra knowledge. For most facets, the layout describes the panels, with one panel per row. It typically has COL, ROW and PANEL columns that keep track of what panel goes where in a grid of cells. In addition, it contains the facetting variables provided to the facets or rows and cols arguments of the facets. For example, if we have a plot facetted on the var variable with the levels A, B and C, as 1 row and 3 columns, we might target the second B panel with any of these expressions: var == "B", PANEL == 2 or COL == 2. We can inspect the layout structure by using ggplot_build(p)$layout$layout, wherein p is a plot.

When using multiple scale_(x/y)_facet(), the expr argument can target the same panels. In such case, the scales added to the plot first overrule the scales that were added later.

Value

A scale_facet object that can be added to a plot.

See Also

The facetted_pos_scales() function.

Examples

```
# A standard plot with continuous scales
p <- ggplot(mtcars, aes(disp, mpg)) +
  geom_point() +
  facet_wrap(~ cyl, scales = "free")

# Adding a scale for a value for a facetting variable
p + scale_x_facet(cyl == 8, limits = c(200, 600))

# Adding a scale by position in the layout
p + scale_x_facet(COL == 3, limits = c(200, 600))

# Setting the default scale and making an exception for one panel
p + scale_y_continuous(limits = c(0, 40)) +
  scale_y_facet(PANEL == 1, limits = c(10, 50))

# Using multiple panel-specific scales
p + scale_y_facet(PANEL == 1, limits = c(10, 50)) +
  scale_y_facet(cyl == 6, breaks = scales::breaks_width(0.5))

# When multiple scales target the same panel, the scale added first gets
# priority over scales added later.
p + scale_y_facet(COL == 2, limits = c(10, 40)) +
  scale_y_facet(cyl %in% c(4, 6), breaks = scales::breaks_width(1))

# A standard plot with discrete x scales
p <- ggplot(mtcars, aes(factor(cyl), mpg)) +
  geom_boxplot() +
  facet_wrap(~ vs, scales = "free")
```

```
# Expanding limits to show every level
p + scale_x_facet(vs == 1, limits = factor(c(4, 6, 8)), type = "discrete")

# Shrinking limits to hide a level
p + scale_x_facet(vs == 0, limits = factor(c(4, 6)), type = "discrete")
```

`scale_fill_multi` *Multiple gradient colour scales*

Description

Maps multiple aesthetics to multiple colour fill gradient scales. It takes in listed arguments for each aesthetic and disseminates these to `ggplot2::continuous_scale()`.

Usage

```
scale_fill_multi(
  ...,
  colours,
  values = NULL,
  na.value = "transparent",
  guide = "colourbar",
  aesthetics = "fill",
  colors
)

scale_colour_multi(
  ...,
  colours,
  values = NULL,
  na.value = "transparent",
  guide = "colourbar",
  aesthetics = "colour",
  colors
)
```

Arguments

`..., colours, values, na.value, guide, colors`	listed arguments in `scale_colour_gradientn()` (e.g. `colours = list(c("white", "red"), c("black", "blue"))`).
`aesthetics`	a `character` vector with names of aesthetic mapping.

Details

This function should only be called after all layers that this function affects are added to the plot.

The list elements of the listed arguments are assumed to follow the `aesthetics` order, i.e. the n*th* list element belongs to the n*th* aesthetic. When there are more list elements than n aesthetics, only

the first n*th* list elements are taken. When there are more `aesthetics` than list elements, the first list element is used for the remaining aesthethics.

In contrast to other `scale_*_continous`-family functions, the `guide` argument is interpreted before adding it to the plot instead of at the time of plot building. This behaviour ensures that the `available_aes` argument of the guides are set correctly, but may interfere with the `ggplot2::guides()` function.

Value

A nested list-like structure of the class `MultiScale`.

Examples

```
# Setup dummy data
df <- rbind(data.frame(x = 1:3, y = 1, v = NA, w = 1:3, z = NA),
            data.frame(x = 1:3, y = 2, v = 1:3, w = NA, z = NA),
            data.frame(x = 1:3, y = 3, v = NA, w = NA, z = 1:3))

ggplot(df, aes(x, y)) +
  geom_raster(aes(fill1 = v)) +
  geom_raster(aes(fill2 = w)) +
  geom_raster(aes(fill3 = z)) +
  scale_fill_multi(aesthetics = c("fill1", "fill2", "fill3"),
                   colours = list(c("white", "red"),
                                  c("black", "blue"),
                                  c("grey50", "green")))
```

---

scale_listed *Add a list of scales for non-standard aesthetics*

Description

This function should only be called after all layers that the non-standard aesthetic scales affects have been added to the plot.

Inside a layer, the non-standard aesthetic should be part of the call to `aes` mapping.

May return a warning that the plot is ignoring unknown aesthetics.

Usage

```
scale_listed(scalelist, replaces = NULL)
```

Arguments

`scalelist`	A `list` wherein elements are the results of calls to a scale function with a non-standard aesthetic set as the `aesthetic` argument.
`replaces`	A `character` vector of the same length as- and parallel to- `scalelist`, indicating what standard aesthetic to replace with the non-standard aesthetic. Typically `"colour"` or `"fill"`.

Details

Distributes a list of non-standard aesthetics scales to the plot, substituting geom and scale settings as necessary to display the non-standard aesthetics. Useful for mapping different geoms to different scales for example.

Value

A `list` of which the elements are of the class `MultiScale`.

Examples

```
# Annotation of heatmap
iriscor <- cor(t(iris[, 1:4]))

df <- data.frame(
  x = as.vector(row(iriscor)),
  y = as.vector(col(iriscor)),
  value = as.vector(iriscor)
)

annotation <- data.frame(
  z = seq_len(nrow(iris)),
  Species = iris$Species,
  Leaves = ifelse(iris$Species == "setosa", "Short", "Long")
)

ggplot(df, aes(x, y)) +
  geom_raster(aes(fill = value)) +
  geom_tile(data = annotation,
            aes(x = z, y = -5, spec = Species), height = 5) +
  geom_tile(data = annotation,
            aes(y = z, x = -5, leav = Leaves), width = 5) +
  scale_listed(
    list(scale_fill_brewer(palette = "Set1", aesthetics = "spec"),
         scale_fill_brewer(palette = "Dark2", aesthetics = "leav")),
    replaces = c("fill", "fill")
  )
```

`scale_x_manual` *Manual position scales*

Description

[Experimental]

`scale_x_manual()` and `scale_y_manual()` are hybrid discrete and continuous position scales for the `x` and `y` aesthetics. These accept input like discrete scales, but may map these discrete values to continuous values that needn't be equally spaced.

Usage

```
scale_x_manual(values, c_limits = NULL, position = "bottom", ...)

scale_y_manual(values, c_limits = NULL, position = "left", ...)
```

Arguments

values	A numeric vector with the same length as unique values. Alternatively, a function that accepts the limits (unique values) as determined from the data and returns a numeric vector parallel to the input.
c_limits	Either NULL (default) to accept the range of values as the continuous limits, or a numeric(2) to set custom continuous limits.
position	For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.
...	Arguments passed on to ggplot2::discrete_scale

scale_name **[Deprecated]** The name of the scale that should be used for error messages associated with this scale.

name The name of the scale. Used as the axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

breaks One of:

- NULL for no breaks
- waiver() for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

labels One of:

- NULL for no labels
- waiver() for the default labels computed by the transformation object
- A character vector giving labels (must be same length as breaks)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

limits One of:

- NULL to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

expand For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function expansion() to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`na.translate` Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

`na.value` If `na.translate = TRUE`, what aesthetic value should the missing values be displayed as? Does not apply to position scales where `NA` is always placed at the far right.

`drop` Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` includes the levels in the factor. Please note that to display every level in a legend, the layer should use `show.legend = TRUE`.

`guide` A function used to create a guide or its name. See `guides()` for more information.

`call` The `call` used to construct the scale for reporting messages.

`super` The super class to use for the constructed scale

Details

Many thanks to Constantin Ahlmann-Eltze for discussion and suggesting the adoption of this functionality in ggh4x.

Value

A `<ScaleManualPosition>` object that can be added to a plot.

Note

There currently is a known bug wherein a `c_limits` cannot be applied correctly when that range is within the range of the discrete limits.

See Also

`sep_discrete()`

Examples

```
# A boxplot with interactions
p <- ggplot(mpg, aes(interaction(year, cyl), displ)) +
  geom_boxplot()

# Manually setting positions
p + scale_x_manual(values = c(1, 2, 4, 6, 7, 9, 10))

# Using a function to separate grouped labels
p + scale_x_manual(values = sep_discrete())

# Expanding the continuous limits
p + scale_x_manual(values = sep_discrete(), c_limits = c(NA, 15))
```

sep_discrete *Separator for discrete grouped labels*

Description

This is a function factory that provides a function to split grouped discrete labels into numerical positions.

Usage

```
sep_discrete(sep = ".", inv = FALSE)
```

Arguments

sep	A `character(1)` separator to use for splitting. May not contain regular expressions.
inv	A `logical(1)` whether to invert the layering of groups.

Value

A `function` that accepts `character` input and returns `numeric` output.

Examples

```
# Here, 'bar.qux' belongs to the second group, so has +1 value
sep_discrete()(c("foo.bar", "bar.bar", "bar.qux"))

# Now, the values are grouped by the groups before the separator
sep_discrete(inv = TRUE)(c("foo.bar", "bar.bar", "bar.qux"))
```

stat_difference *Difference ribbon*

Description

This makes a ribbon that is filled depending on whether the `max` is higher than `min`. This can be useful for displaying differences between two series.

Usage

```
stat_difference(
  mapping = NULL,
  data = NULL,
  geom = "ribbon",
  position = "identity",
  ...,
  levels = c("+", "-"),
  na.rm = FALSE,
  orientation = NA,
  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)).
geom	Use to override the default connection between geom_ribbon() and stat_difference().
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.

`levels` A `character(2)` indicating factor levels for the `fill` aesthetic for the cases where (1) `max > min` and where (2) `max < min`.

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

`orientation` The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.

`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.

`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. `borders()`.

Details

The stat may reorder the `group` aesthetic to accommodate two different fills for the signs of differences. The stat takes care to interpolate a series whenever a crossover between `max` and `min` series happens. This makes the ribbon not look stumpy at these crossovers.

Value

A `Layer` object that can be added to a plot.

Aesthetics

`geom_ribbon()` understands the following aesthetics (required aesthetics are in bold):

- `x` ***or*** `y`
- `ymin` ***or*** `xmin`
- `ymax` ***or*** `xmax`

- alpha
- colour
- fill
- group
- linetype
- linewidth

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

Computed variables

`sign` A `factor` with the `levels` attribute set to the `levels` argument.

Note

When there is a run of more than two 0-difference values, the inner values will be ignored.

Examples

```
set.seed(2021)
df <- data.frame(
  x = 1:100,
  y = cumsum(rnorm(100)),
  z = cumsum(rnorm(100))
)

ggplot(df, aes(x = x)) +
  stat_difference(aes(ymin = y, ymax = z), alpha = 0.3) +
  geom_line(aes(y = y, colour = "min")) +
  geom_line(aes(y = z, colour = "max"))
```

---

stat_funxy *Apply function to position coordinates*

Description

The function xy stat applies a function to the x- and y-coordinates of a layers positions by group. The `stat_centroid()` and `stat_midpoint()` functions are convenience wrappers for calculating centroids and midpoints. `stat_funxy()` by default leaves the data as-is, but can be supplied functions and arguments.

Usage

```
stat_funxy(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  funx = force,
  funy = force,
  argx = list(),
  argy = list(),
  crop_other = TRUE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_centroid(
  ...,
  funx = mean,
  funy = mean,
  argx = list(na.rm = TRUE),
  argy = list(na.rm = TRUE)
)

stat_midpoint(..., argx = list(na.rm = TRUE), argy = list(na.rm = 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)).
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.

... — 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.

funx, funy — A function to call on the layer's x and y positions respectively.

argx, argy — A named list containing arguments to the funx, and funy function calls.

crop_other — A logical of length one; whether the other data should be fitted to the length of x and y (default: TRUE). Useful to set to FALSE when funx or funy calculate summaries of length one that need to be recycled.

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.

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. borders().

Details

This statistic only makes a minimal attempt at ensuring that the results from calling both functions are of equal length. Results of length 1 are recycled to match the longest length result.

Value

A `StatFunxy` ggproto object, that can be added to a plot.

Examples

```
p <- ggplot(iris, aes(Sepal.Width, Sepal.Length, colour = Species))

# Labelling group midpoints
p + geom_point() +
  stat_midpoint(aes(label = Species, group = Species),
                geom = "text", colour = "black")

# Drawing segments to centroids
p + geom_point() +
  stat_centroid(aes(xend = Sepal.Width, yend = Sepal.Length),
                geom = "segment", crop_other = FALSE)

# Drawing intervals
ggplot(iris, aes(Sepal.Width, Sepal.Length, colour = Species)) +
  geom_point() +
  stat_funxy(geom = "path",
             funx = median, funy = quantile,
             argy = list(probs = c(0.1, 0.9)))
```

---

stat_rle *Run length encoding*

Description

Run length encoding takes a vector of values and calculates the lengths of consecutive repeated values.

Usage

```
stat_rle(
  mapping = NULL,
  data = NULL,
  geom = "rect",
  position = "identity",
  ...,
  align = "none",
  na.rm = FALSE,
  orientation = "x",
```

```
  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)).
geom	Use to override the default connection between geom_rect() and stat_rle().
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.

`align` A `character` of length one that effect the computed `start` and `end` variables. One of the following:

`"none"` Take exact start and end `x` values.

`"center"` Return start and end `x` values in between an end and the subsequent start.

`"start"` Align start values with previous end values.

`"end"` Align end values with next start values.

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

`orientation` The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.

`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.

`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. `borders()`.

Details

The data is first ordered on the `x` aesthetic before run lengths are calculated for the `label` aesthetic. In contrast to `base::rle()`, `NA`s are considered equivalent values, not different values.

Value

A `ggplot2` layer

Aesthetics

`stat_rle()` understands the following aesthetics (required aesthetics are in bold)

- **x**
- **label**
- group

Computed variables

start The `x` values at the start of every run.

end The `x` values at the end of every run.

start_id The index where a run starts.

end_id The index where a run ends.

run_id The index of a run.

runlength The length of a run.

runvalue The value associated with a run.

Examples

```
df <- data.frame(
  x = seq(0, 10, length.out = 100),
  y = sin(seq(0, 10, length.out = 100)*2)
)

# Label every run of increasing values
ggplot(df) +
  stat_rle(aes(x, label = diff(c(0, y)) > 0),
           align = "end") +
  geom_point(aes(x, y))

# Label every run above some threshold
ggplot(df) +
  stat_rle(aes(x, label = y > 0),
           align = "center") +
  geom_point(aes(x, y))

# Categorising runs, more complicated usage
ggplot(df) +
  stat_rle(aes(stage(x, after_stat = run_id),
               after_stat(runlength),
               label = cut(y, c(-1, -0.6, 0.6, 1)),
               fill = after_stat(runvalue)),
           geom = "col")
```

---

stat_rollingkernel *Rolling Kernel*

Description

A rolling kernel moves along one of the axes and assigns weights to datapoints depending on the distance to the kernel's location. It then calculates a weighted average on the y-values of the datapoints, creating a trendline. In contrast to (weighted) rolling averages, the interval between datapoints do not need to be constant.

Usage

```
stat_rollingkernel(
  mapping = NULL,
  data = NULL,
```

```
  geom = "line",
  position = "identity",
  ...,
  bw = "nrd",
  kernel = "gaussian",
  n = 256,
  expand = 0.1,
  na.rm = FALSE,
  orientation = "x",
  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)).
geom	Use to override the default geom ("line").
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.

`bw` A bandwidth, which can be one of the following:

- A `numeric` of length one indicating a measure of kernel width, in data units.
- A `rel` object of length one constructed for setting a bandwidth relative to the group data range. Can be constructed with the `rel()` function.
- A `character` of length one, naming one of the functions documented in `bw.nrd()`.

`kernel` One of the following:

- A `function` that takes a vector of distances as first argument, a numeric bandwidth as second argument and returns relative weights.
- A `character` of length one that can take one of the following values:

  `"gaussian"` **or** `"norm"` A kernel that follows a normal distribution with 0 mean and bandwidth as standard deviation.

  `"mean"` **or** `"unif"` A kernel that follows a uniform distribution with $bandwidth * -0.5$ and $bandwidth * 0.5$ as minimum and maximum. This is similar to a simple, unweighted moving average.

  `"cauchy"` A kernel that follows a Cauchy distribution with 0 as location and bandwidth as scale parameters. The Cauchy distribution has fatter tails than the normal distribution.

`n` An `integer` of length one: how many points to return per group.

`expand` A `numeric` of length one: how much to expand the range for which the rolling kernel is calculated beyond the most extreme datapoints.

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

`orientation` A `character` of length one, either `"x"` (default) or `"y"`, setting the axis along which the rolling should occur.

`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.

`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. `borders()`.

Value

A *Layer* ggproto object.

Aesthetics

`stat_rollingkernel()` understands the following aesthetics (required aesthetics are in bold)

- **x**
- **y**
- group

Computed variables

`x` A sequence of ordered x positions.

`y` The weighted value of the rolling kernel.

`weight` The sum of weight strengths at a position.

`scaled` The fraction of weight strengths at a position. This is the same as `weight / sum(weight)` by group.

Examples

```
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() +
  stat_rollingkernel()

# The (scaled) weights can be used to emphasise data-dense areas
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() +
  stat_rollingkernel(aes(alpha = after_stat(scaled)))
```

`stat_theodensity` *Fitted theoretical density*

Description

Estimates the parameters of a given distribution and evaluates the probability density function with these parameters. This can be useful for comparing histograms or kernel density estimates against a theoretical distribution.

Usage

```
stat_theodensity(
  mapping = NULL,
  data = NULL,
  geom = "line",
  position = "identity",
```

```
  ...,
  distri = "norm",
  n = 512,
  fix.arg = NULL,
  start.arg = NULL,
  na.rm = TRUE,
  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)`).
geom	Use to override the default geom for `stat_theodensity`.
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.

`distri` A `character` of length 1 naming a distribution without prefix. See details.

`n` An `integer` of length 1 with the number of equally spaced points at which the density function is evaluated. Ignored if distribution is discrete.

`fix.arg` An optional named list giving values of fixed parameters of the named distribution. Parameters with fixed value are not estimated by maximum likelihood procedures.

`start.arg` A named list giving initial values of parameters for the named distribution. This argument may be omitted (default) for some distributions for which reasonable starting values are computed.

`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.

`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. `borders()`.

Details

Valid `distri` arguments are the names of distributions for which there exists a density function. The names should be given without a prefix (typically 'd', 'r', 'q' and 'r'). For example: `"norm"` for the normal distribution and `"nbinom"` for the negative binomial distribution. Take a look at `distributions()` in the **stats** package for an overview.

There are a couple of distribution for which there exist no reasonable starting values, such as the Student t-distribution and the F-distribution. In these cases, it would probably be wise to provide reasonable starting values as a named list to the `start.arg` argument. When estimating a binomial distribution, it would be best to supply the `size` to the `fix.arg` argument.

By default, the y values are such that the integral of the distribution is 1, which scales well with the defaults of kernel density estimates. When comparing distributions with absolute count histograms, a sensible choice for aesthetic mapping would be `aes(y = stat(count) * binwidth)`, wherein `binwidth` is matched with the bin width of the histogram.

For discrete distributions, the input data are expected to be integers, or doubles that can be divided by 1 without remainders.

Parameters are estimated using the `fitdistrplus::fitdist()` function in the **fitdistrplus** package using maximum likelihood estimation. Hypergeometric and multinomial distributions from the **stats** package are not supported.

Value

A *Layer* ggproto object.

Computed variables

density probability density

count density * number of observations - useful for comparing to histograms

scaled density scaled to a maximum of 1

See Also

`stats::Distributions()` `fitdistrplus::fitdist()` `ggplot2::geom_density()` `ggplot2::geom_histogram()`

Examples

```
# A mixture of normal distributions where the standard deviation is
# inverse gamma distributed resembles a cauchy distribution.
x <- rnorm(2000, 10, 1/rgamma(2000, 2, 0.5))
df <- data.frame(x = x)

ggplot(df, aes(x)) +
  geom_histogram(binwidth = 0.1,
                 alpha = 0.3, position = "identity") +
  stat_theodensity(aes(y = stat(count) * 0.1, colour = "Normal"),
                   distri = "norm", geom = "line") +
  stat_theodensity(aes(y = stat(count) * 0.1, colour = "Cauchy"),
                   distri = "cauchy", geom = "line") +
  coord_cartesian(xlim = c(5, 15))

# A negative binomial can be understood as a Poisson-gamma mixture
df <- data.frame(x = c(rpois(500, 25),
                       rpois(500, rgamma(500, 5, 0.2))),
                 cat = rep(c("Poisson", "Poisson-gamma"), each = 500))

ggplot(df, aes(x)) +
  geom_histogram(binwidth = 1, aes(fill = cat),
                 alpha = 0.3, position = "identity") +
  stat_theodensity(aes(y = stat(count), colour = cat), distri = "nbinom",
                   geom = "step", position = position_nudge(x = -0.5)) +
  stat_summary(aes(y = x, colour = cat, x = 1),
               fun.data = function(x){data.frame(xintercept = mean(x))},
               geom = "vline")
```

`strip_nested` *Nested strips*

Description

This strip style groups strips on the same layer that share a label. It is the default strip for `facet_nested()` and `facet_nested_wrap()`.

Usage

```
strip_nested(
  clip = "inherit",
  size = "constant",
  bleed = FALSE,
  text_x = NULL,
  text_y = NULL,
  background_x = NULL,
  background_y = NULL,
  by_layer_x = FALSE,
  by_layer_y = FALSE
)
```

Arguments

`clip`	A `character(1)` that controls whether text labels are clipped to the background boxes. Can be either `"inherit"` (default), `"on"` or `"off"`.
`size`	A `character(1)` stating that the strip margins in different layers remain `"constant"` or are `"variable"`.
`bleed`	A `logical(1)` indicating whether merging of lower-layer variables is allowed when the higher-layer variables are separate. See details.
`text_x, text_y`	A `list()` with `element_text()` elements. See the details section in `strip_themed()`.
`background_x, background_y`	A `list()` with `element_rect()` elements. See the details section in `strip_themed()`.
`by_layer_x, by_layer_y`	A `logical(1)` that when `TRUE`, maps the different elements to different layers of the strip. When `FALSE`, maps the different elements to individual strips, possibly repeating the elements to match the number of strips through `rep_len()`.

Details

The display order is always such that the outermost variable is placed the furthest away from the panels. Strips are automatically grouped when they span a nested variable.

The `bleed` argument controls whether lower-layer strips are allowed to be merged when higher-layer strips are different, i.e. they can bleed over hierarchies. Suppose the `strip_vanilla()` behaviour would be the following for strips:

```
[_1_][_2_][_2_]
[_3_][_3_][_4_]
```

In such case, the default `bleed = FALSE` argument would result in the following:

```
[_1_][___2____]
[_3_][_3_][_4_]
```

Whereas `bleed = TRUE` would allow the following:

```
[_1_][___2____]
[___3____][_4_]
```

Value

A `StripNested` ggproto object that can be given as an argument to facets in ggh4x.

See Also

Other strips: `strip_split()`, `strip_tag()`, `strip_themed()`, `strip_vanilla()`

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()

# Combine the strips
p + facet_wrap2(vars(cyl, drv), strip = strip_nested())

# The facet_nested and facet_nested_wrap functions have nested strips
# automatically
p + facet_nested_wrap(vars(cyl, drv))

# Changing the bleed argument merges the "f" labels in the top-right
p + facet_wrap2(vars(cyl, drv), strip = strip_nested(bleed = TRUE))
```

---

strip_split *Split strips*

Description

[Experimental]

This strip style allows a greater control over where a strip is placed relative to the panel. Different facetting variables are allowed to be placed on different sides.

Usage

```
strip_split(
  position = c("top", "left"),
  clip = "inherit",
  size = "constant",
  bleed = FALSE,
  text_x = NULL,
  text_y = NULL,
  background_x = NULL,
  background_y = NULL,
  by_layer_x = FALSE,
  by_layer_y = FALSE
)
```

Arguments

position	A character vector stating where the strips of faceting variables should be placed. Can be some of the following: "top", "bottom", "left" or "right". The length of the position argument must match the length of variables provided to the facets argument in wrap/manual layouts, or those provided to the rows and cols arguments in the grid layout.
clip	A character(1) that controls whether text labels are clipped to the background boxes. Can be either "inherit" (default), "on" or "off".
size	A character(1) stating that the strip margins in different layers remain "constant" or are "variable".
bleed	A logical(1) indicating whether merging of lower-layer variables is allowed when the higher-layer variables are separate. See the details of strip_nested for more info. Note that currently, strip_split() cannot recognise collisions between strips, so changing to bleed = TRUE can have unexpected results.
text_x, text_y	A list() with element_text() elements. See the details section in strip_themed().
background_x, background_y	A list() with element_rect() elements. See the details section in strip_themed().
by_layer_x, by_layer_y	A logical(1) that when TRUE, maps the different elements to different layers of the strip. When FALSE, maps the different elements to individual strips, possibly repeating the elements to match the number of strips through rep_len().

Details

Using this style of strip completely overrules the strip.position and switch arguments.

Value

A StripSplit ggproto object that can be given as an argument to facets in ggh4x.

See Also

Other strips: strip_nested(), strip_tag(), strip_themed(), strip_vanilla()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()

# --- Wrap examples ------

# Defaults to 1st (cyl) at top, 2nd (drv) on left
p + facet_wrap2(vars(cyl, drv), strip = strip_split())

# Change cyl to left, drv to bottom
p + facet_wrap2(vars(cyl, drv), strip = strip_split(c("left", "bottom")))

# --- Grid examples -----

# Display both strips levels on the left
p + facet_grid2(vars(drv), vars(cyl),
                strip = strip_split(c("left", "left")))

# Separate the strips again
p + facet_grid2(vars(cyl, year),
                strip = strip_split(c("bottom", "left")))

# Using a dummy variable as a title strip
p + facet_grid2(vars(cyl, "year", year),
                strip = strip_split(c("bottom", "left", "left")))
```

---

strip_tag *Strips as tags*

Description

This strip style renders the strips as text with fitted boxes onto the panels of the plot. This is in contrast to strips that match the panel size and are located outside the panels.

Usage

```
strip_tag(
  clip = "inherit",
  order = c("x", "y"),
  just = c(0, 1),
  text_x = NULL,
  text_y = element_text(angle = 0),
  background_x = NULL,
  background_y = NULL,
  by_layer_x = FALSE,
  by_layer_y = FALSE
)
```

Arguments

clip	A character(1) that controls whether text labels are clipped to the background boxes. Can be either "inherit" (default), "on" or "off".
order	Either c("x", "y") or c("y", "x"), setting the top-to-bottom order of horizontal versus "vertical" labels in facets with a grid layout.
just	A <numeric[2]> setting the horizontal and vertical justification of placing the textbox.
text_x, text_y	A list() with element_text() elements. See details.
background_x, background_y	A list() with element_rect() elements. See details.
by_layer_x, by_layer_y	A logical(1) that when TRUE, maps the different elements to different layers of the strip. When FALSE, maps the different elements to individual strips, possibly repeating the elements to match the number of strips through rep_len().

Value

A StripTag ggproto object that can be given as an argument to facets in ggh4x.

See Also

Other strips: strip_nested(), strip_split(), strip_themed(), strip_vanilla()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()

# Typical use
p + facet_wrap2(
  ~ class,
  strip = strip_tag()
)

# Adjusting justification
p + facet_wrap2(
  ~ class,
  strip = strip_tag(just = c(1, 0))
)

p + facet_wrap2(
  ~ drv + year,
  strip = strip_tag()
)

# With a grid layout, you can control in which order the labels are drawn
p + facet_grid2(
  "vertical" ~ "horizontal",
```

```
  strip = strip_tag(order = c("x", "y")) # default
)

p +facet_grid2(
  "vertical" ~ "horizontal",
  strip = strip_tag(order = c("y", "x")) # invert order
)
```

`strip_themed` *Strip with themed boxes and texts*

Description

A style of strips with individually themed strips.

Usage

```
strip_themed(
  clip = "inherit",
  size = "constant",
  text_x = NULL,
  text_y = NULL,
  background_x = NULL,
  background_y = NULL,
  by_layer_x = FALSE,
  by_layer_y = FALSE
)
```

Arguments

`clip`	A `character(1)` that controls whether text labels are clipped to the background boxes. Can be either `"inherit"` (default), `"on"` or `"off"`.
`size`	A `character(1)` stating that the strip margins in different layers remain `"constant"` or are `"variable"`.
`text_x, text_y`	A `list()` with `element_text()` elements. See details.
`background_x, background_y`	A `list()` with `element_rect()` elements. See details.
`by_layer_x, by_layer_y`	A `logical(1)` that when `TRUE`, maps the different elements to different layers of the strip. When `FALSE`, maps the different elements to individual strips, possibly repeating the elements to match the number of strips through `rep_len()`.

Details

With respect to the `text_*` and `background_*` arguments, they can be a list with (a mix of) the following objects:

- NULL indicates that the global plot theme applies.
- element_blank() omits drawing the background or text.
- An element class object inheriting from the element_text or element_rect classes.

For constructing homogeneous lists of elements, the elem_list_text() and elem_list_rect() are provided for convenience.

Value

A StripThemed ggproto object that can be given as an argument to facets in ggh4x.

See Also

Other strips: strip_nested(), strip_split(), strip_tag(), strip_vanilla()

Examples

```
# Some simple plot
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()

# Set some theming options, we can use `element_blank()`
backgrounds <- list(element_blank(), element_rect(fill = "dodgerblue"))
# Or we could use `NULL` to use the global theme
texts <- list(element_text(colour = "red"), NULL, element_text(face = "bold"))

# Elements are repeated until the fit the number of facets
p + facet_wrap2(
  vars(drv, year),
  strip = strip_themed(
    background_x = backgrounds,
    text_x = texts
  )
)

# Except when applied to each layer instead of every strip
p + facet_wrap2(
  vars(drv, year),
  strip = strip_themed(
    background_x = backgrounds,
    text_x = texts,
    by_layer_x = TRUE
  )
)

# To conveniently distribute arguments over a list of the same elements,
# you can use the following wrappers:
p + facet_wrap2(
  vars(drv, year),
  strip = strip_themed(
    text_x = elem_list_text(colour = c("blue", "red")),
    background_x = elem_list_rect(fill = c("white", "grey80")),
```

```
    by_layer_x = TRUE
  )
)
```

`strip_vanilla` *Default strips*

Description

Strips with the style of vanilla ggplot2.

Usage

```
strip_vanilla(clip = "inherit", size = "constant")
```

Arguments

`clip`	A `character(1)` that controls whether text labels are clipped to the background boxes. Can be either `"inherit"` (default), `"on"` or `"off"`.
`size`	A `character(1)` stating that the strip margins in different layers remain `"constant"` or are `"variable"`.

Value

A `Strip` ggproto object that can be used ggh4x facets.

See Also

Other strips: `strip_nested()`, `strip_split()`, `strip_tag()`, `strip_themed()`

Examples

```
# Some dummy data with a long string
df <- data.frame(
  short = "X",
  long  = "A very long string that takes up a lot of space",
  value = 1
)
# Simple plot
p <- ggplot(df, aes(value, value)) +
  geom_point() +
  theme(strip.text.y.right = element_text(angle = 0))

# Short titles take up as much space as long titles
p + facet_grid2(
  vars(short, long),
  strip = strip_vanilla(size = "constant")
)
```

```
# Short titles take up less space
p + facet_grid2(
  vars(short, long),
  strip = strip_vanilla(size = "variable")
)
```

`theme_extensions` *Theme extensions*

Description

Some functions in **ggh4x** are using extensions to the theme system. These extended theme argument are listed below, along with what elements they are expected to be, and in what function(s) they are used.

Arguments

`ggh4x.facet.nestline`
: An `element_line()` used as the parent for the `nest_line` argument in `facet_nested()` and `facet_nested_wrap()`. Inherits directly from the '`line`' theme element.

`ggh4x.axis.nestline`, `ggh4x.axis.nestline.x`, `ggh4x.axis.nestline.y`
: An `element_line()` used as the line to separate different layers of labels in `guide_axis_nested()`. Inherits from the '`axis.ticks`' theme element.

`ggh4x.axis.nesttext.x`, `ggh4x.axis.nesttext.y`
: An `element_text()` used to differentiate text higher in the hierarchy from the axis labels directly next to the axis line in `guide_axis_nested()`. Inherits from the '`axis.text.x`' and '`axis.text.y`' theme elements respectively.

`ggh4x.axis.ticks.length.minor`
: A `rel()` object used to set the size of minor tick marks relative to the regular tick marks. This is used in the `guide_axis_minor()` and `guide_axis_logticks()` functions. Defaults to `rel(2/3)`.

`ggh4x.axis.ticks.length.mini`
: A `rel()` object used to set the size of the smallest tick marks relative to regular tick marks. This is only used in the `guide_axis_logticks()` function. Defaults to `rel(1/3)`.

`weave_factors` *Bind together factors*

Description

Computes a new factor out of combinations of input factors.

Usage

```
weave_factors(..., drop = TRUE, sep = ".", replaceNA = TRUE)
```

Arguments

...	The vectors
drop	A logical of length 1 which when TRUE will remove combinations of factors not occurring in the input data.
sep	A character of length 1 with a string to delimit the new level labels.
replaceNA	A logical of length 1: replace NA values with empty strings?

Details

weave_factors() broadly resembles interaction(..., lex.order = TRUE), with a slightly altered approach to non-factor inputs. In other words, this function orders the new levels such that the levels of the first input variable in ... is given priority over the second input, the second input has priority over the third, etc.

This function treats non-factor inputs as if their levels were unique(as.character(x)), wherein x represents an input.

Value

A factor representing combinations of input factors.

See Also

interaction()

Examples

```
f1 <- c("banana", "apple", "apple", "kiwi")
f2 <- factor(c(1, 1:3), labels = c("house", "cat", "dog"))

# Notice the difference in level ordering between the following:
interaction(f1, f2, drop = TRUE, lex.order = TRUE)
interaction(f1, f2, drop = TRUE, lex.order = FALSE)
weave_factors(f1, f2)

# The difference is in how characters are interpreted
# The following are equivalent
interaction(f1, f2, drop = TRUE, lex.order = TRUE)
weave_factors(as.factor(f1), f2)
```

Index