

# Package: ggh4x (via r-universe)

September 7, 2024

**Title** Hacks for 'ggplot2'

**Version** 0.2.8.9000

**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 (>= 3.5.0)

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

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

**VignetteBuilder** knitr

**Encoding** UTF-8

**RoxygenNote** 7.3.1

**Config/testthat/edition** 3

**Collate** 'at\_panel.R' 'borrowed\_ggplot2.R' 'conveniences.R'  
'ggh4x\_extensions.R' 'coord\_axes\_inside.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\_axis\_logticks.R' 'guide\_axis\_manual.R'  
'guide\_axis\_minor.R' 'guide\_axis\_nested.R'  
'guide\_axis\_scalebar.R' 'guide\_axis\_truncated.R'

'guide\_axis\_utils.R' 'guide\_dendrogram.R'  
 'guide\_stringlegend.R' 'help\_secondary.R'  
 'position\_disjoint\_ranges.R' 'position\_lineartrans.R'  
 'scale\_dendrogram.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' 'themes.R' 'utils.R' 'utils\_grid.R'  
 'utils\_gtable.R'

**Roxygen** list(markdown = TRUE)

**Repository** <https://teunbrand.r-universe.dev>

**RemoteUrl** <https://github.com/teunbrand/ggh4x>

**RemoteRef** HEAD

**RemoteSha** e797c7cbb006aa852318a73240de9926e8364aa

## Contents

at_panel . . . . .	3
center_limits . . . . .	4
coord_axes_inside . . . . .	5
distribute_args . . . . .	6
element_part_rect . . . . .	8
faceted_pos_scales . . . . .	9
facet_grid2 . . . . .	11
facet_manual . . . . .	14
facet_nested . . . . .	16
facet_nested_wrap . . . . .	19
facet_wrap2 . . . . .	22
force_panelsizes . . . . .	24
geom_box . . . . .	26
geom_outline_point . . . . .	28
geom_pointpath . . . . .	31
geom_polygonraster . . . . .	34
geom_rectmargin . . . . .	37
geom_text_aimed . . . . .	41
ggh4x_extensions . . . . .	44
ggsubset . . . . .	45
guide_axis_logticks . . . . .	46
guide_axis_manual . . . . .	48
guide_axis_minor . . . . .	50
guide_axis_nested . . . . .	52
guide_axis_scalebar . . . . .	55
guide_axis_truncated . . . . .	56
guide_dendro . . . . .	59
guide_stringlegend . . . . .	61
help_secondary . . . . .	63

position_disjoint_ranges . . . . .	65
position_lineartrans . . . . .	66
scale_dendrogram . . . . .	70
scale_facet . . . . .	72
scale_fill_multi . . . . .	74
scale_listed . . . . .	75
scale_x_manual . . . . .	76
sep_discrete . . . . .	79
stat_difference . . . . .	79
stat_funxy . . . . .	82
stat_rle . . . . .	85
stat_rollingkernel . . . . .	88
stat_theodensity . . . . .	91
strip_nested . . . . .	95
strip_split . . . . .	96
strip_themed . . . . .	98
strip_vanilla . . . . .	100
theme_extensions . . . . .	101
weave_factors . . . . .	102

**Index****104**


---

at_panel	<i>Constrain layer to panels</i>
----------	----------------------------------

---

**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 <code>layer()</code> .
expr	An expression that, when evaluated in the facet's layout <code>data.frame</code> , 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 faceting variables provided to the `facets` or `rows` and `cols` arguments of the facets. For example, if we have a plot faceted 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**

<code>xlim, ylim</code>	Limits for the x and y axes.
<code>xintercept, yintercept</code>	A <code>numeric(1)</code> 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.
<code>labels_inside</code>	A <code>logical(1)</code> when labels should be placed inside the panel along the axes ( <code>TRUE</code> ) or placed outside the panel ( <code>FALSE</code> , default).
<code>ratio</code>	Either <code>NULL</code> , or a <code>numeric(1)</code> for a fixed aspect ratio, expressed as $y / x$ .
<code>expand</code>	If <code>TRUE</code> , the default, adds a small expansion factor to the limits to ensure that data and axes don't overlap. If <code>FALSE</code> , limits are taken exactly from the data or <code>xlim/ylim</code> .
<code>default</code>	Is this the default coordinate system? If <code>FALSE</code> (the default), then replacing this coordinate system with another one creates a message alerting the user that the coordinate system is being replaced. If <code>TRUE</code> , 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 = "axis_minor") +
  coord_axes_inside()
```

---

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

<code>side</code>	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 <code>element_rect()</code> .
<code>fill</code>	Fill colour.
<code>colour, color</code>	Line/border colour. Color is an alias for colour.
<code>linewidth</code>	Line/border size in mm.
<code>linetype</code>	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.
<code>inherit.blank</code>	Should this element inherit the existence of an <code>element_blank</code> 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")  
  )
```

---

faceted\_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**

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

**Arguments**

`x, y`                    A list wherein elements are either x/y position scales or NULLs. 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:** NULLs 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 `faceted_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 `faceted_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 +
  faceted_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 +
  faceted_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 = strip_vanilla()
)
```

**Arguments**

rows, cols	A set of variables or expressions quoted by <code>vars()</code> and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to <code>labeller</code> ). For compatibility with the classic interface, <code>rows</code> 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 <code>independent</code> argument. One of the following: "fixed" <b>or</b> 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" <b>or</b> 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 <code>independent</code> argument allows

	<p>for free scales in a dimension, the panel sizes cannot be proportional. Note that the <code>scales</code> argument must be free in the same dimension as the <code>space</code> argument to have an effect. One of the following:</p> <p>"fixed" <b>or</b> FALSE All panels have the same size (default).</p> <p>"free_x" Panel widths are proportional to the x-scales.</p> <p>"free_y" Panel heights are proportional to the y-scales.</p> <p>"free" <b>or</b> TRUE Both the widths and heights vary according to scales.</p>
axes	<p>A character(1) or logical(1) where axes should be drawn. One of the following:</p> <p>"margins" <b>or</b> FALSE Only draw axes at the outer margins (default).</p> <p>"x" Draw axes at the outer margins and all inner x-axes too.</p> <p>"y" Draw axes at the outer margins and all inner y-axes too.</p> <p>"all" <b>or</b> TRUE Draw the axes for every panel.</p>
remove_labels	<p>A character(1) or logical(1) determining whether axis text is displayed at inner panels. One of the following:</p> <p>"none" <b>or</b> FALSE Display axis text at all axes (default).</p> <p>"x" Display axis text at outer margins and all inner y-axes.</p> <p>"y" Display axis text at outer margins and all inner x-axes.</p> <p>"all" <b>or</b> TRUE Only display axis text at the outer margins.</p>
independent	<p>A character(1) or logical(1) determining whether scales can vary within a row or column of panels, like they can be in <code>ggplot2::facet_wrap</code>. The <code>scales</code> argument must be free for the same dimension before they can be set to independent. One of the following:</p> <p>"none" <b>or</b> FALSE All y-scales should be fixed in a row and all x-scales are fixed in a column (default).</p> <p>"x" x-scales are allowed to vary within a column.</p> <p>"y" y-scales are allowed to vary within a row.</p> <p>"all" <b>or</b> TRUE Both x- and y-scales are allowed to vary within a column or row respectively.</p>
shrink	<p>If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.</p>
labeller	<p>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 <code>vars(cyl, am)</code>. 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 <code>labeller()</code>. You can use different labeling functions for different kind of labels, for example use <code>label_parsed()</code> for formatting facet labels. <code>label_value()</code> is used by default, check it for more details and pointers to other options.</p>
as.table	<p>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.</p>

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 <code>logical(1)</code> : 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 <a href="#">strip_vanilla</a> .

### 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 faceting 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")
```

**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 = strip_vanilla()
)
```

**Arguments**

facets	A set of variables or expressions quoted by <code>vars()</code> and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to <code>labeller</code> ). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, <code>~a + b</code> , or a character vector, <code>c("a", "b")</code> .
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 <code>NULL</code> (default), the sizes are set per instructions of <code>coord</code> or <code>theme</code> 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 <code>logical(1)</code> . If <code>TRUE</code> , widths and heights specified in "null" units are proportional. If <code>FALSE</code> , "null" units in the x- and y-directions can vary independently. Alternatively, when <code>NULL</code> , the <code>respect</code> parameter takes instructions from the <code>coord</code> or <code>theme</code> .

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 <code>strip.position</code> it is possible to place the labels on either of the four sides by setting <code>strip.position = c("top", "bottom", "left", "right")</code>
scales	A character(1) or logical(1) whether scales are shared across facets or allowed to vary. One of the following: "fixed" <b>or</b> 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" <b>or</b> TRUE Both scales can vary
axes	A character(1) or logical(1) where axes should be drawn. One of the following: "margins" <b>or</b> 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" <b>or</b> 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" <b>or</b> 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" <b>or</b> 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 <code>vars(cyl, am)</code> . 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 <code>labeller()</code> . You can use different labeling functions for different kind of labels, for example use <code>label_parsed()</code> for formatting facet labels. <code>label_value()</code> 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	An object created by a call to a strip function, such as <code>strip_vanilla</code> .

**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 = strip_nested(),
bleed = NULL
)

```

### Arguments

rows, cols	<p>A set of variables or expressions quoted by <code>vars()</code> and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to <code>labeller</code>).</p> <p>For compatibility with the classic interface, <code>rows</code> 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).</p>
scales	<p>A <code>character(1)</code> or <code>logical(1)</code> whether scales are shared across facets or allowed to vary. Interacts with the <code>independent</code> argument. One of the following:</p> <ul style="list-style-type: none"> <li>"fixed" <b>or</b> FALSE Scales are shared across all facets (default).</li> <li>"free_x" x-scales are allowed to vary across rows.</li> <li>"free_y" y-scales are allowed to vary across columns.</li> <li>"free" <b>or</b> TRUE Scales can vary across rows and columns.</li> </ul>
space	<p>A <code>character(1)</code> or <code>logical(1)</code> determining whether the size of panels are proportional to the length of the scales. When the <code>independent</code> argument allows for free scales in a dimension, the panel sizes cannot be proportional. Note that the <code>scales</code> argument must be free in the same dimension as the <code>space</code> argument to have an effect. One of the following:</p> <ul style="list-style-type: none"> <li>"fixed" <b>or</b> FALSE All panels have the same size (default).</li> <li>"free_x" Panel widths are proportional to the x-scales.</li> <li>"free_y" Panel heights are proportional to the y-scales.</li> <li>"free" <b>or</b> TRUE Both the widths and heights vary according to scales.</li> </ul>
axes	<p>A <code>character(1)</code> or <code>logical(1)</code> where axes should be drawn. One of the following:</p> <ul style="list-style-type: none"> <li>"margins" <b>or</b> FALSE Only draw axes at the outer margins (default).</li> <li>"x" Draw axes at the outer margins and all inner x-axes too.</li> <li>"y" Draw axes at the outer margins and all inner y-axes too.</li> <li>"all" <b>or</b> TRUE Draw the axes for every panel.</li> </ul>
remove_labels	<p>A <code>character(1)</code> or <code>logical(1)</code> determining whether axis text is displayed at inner panels. One of the following:</p> <ul style="list-style-type: none"> <li>"none" <b>or</b> FALSE Display axis text at all axes (default).</li> </ul>

	"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" <b>or</b> 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 <code>ggplot2::facet_wrap</code> . The scales argument must be free for the same dimension before they can be set to independent. One of the following: <ul style="list-style-type: none"> <li>"none" <b>or</b> FALSE All y-scales should be fixed in a row and all x-scales are fixed in a column (default).</li> <li>"x" x-scales are allowed to vary within a column.</li> <li>"y" y-scales are allowed to vary within a row.</li> <li>"all" <b>or</b> TRUE Both x- and y-scales are allowed to vary within a column or row respectively.</li> </ul>
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 <code>vars(cyl, am)</code> . 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 <code>labeller()</code> . You can use different labeling functions for different kind of labels, for example use <code>label_parsed()</code> for formatting facet labels. <code>label_value()</code> 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 <code>element_blank()</code> or inheriting from <code>ggplot2::element_line()</code> . Lines are drawn between layers of strips indicating hierarchy. The element inherits from the <code>ggh4x.facet.nestline</code> element in the theme.
solo_line	A logical(1) indicating whether parent strips with a single child should be drawn with a <code>nest_line</code> (TRUE) or the line only applies to parents with multiple children (FALSE, default). Only relevant when <code>nest_line</code> is drawn.
reset	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 <a href="#">strip_nested()</a> .
bleed	<b>[Deprecated]</b> the bleed argument has moved to the <a href="#">strip_nested()</a> 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 faceting 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	<i>Ribbon of panels with nested strips.</i>
-------------------	---

---

## 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 = strip_nested(),
  bleed = NULL
)
```

**Arguments**

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

	"none" <b>or</b> 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" <b>or</b> 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 <code>vars(cyl, am)</code> . 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 <code>labeller()</code> . You can use different labeling functions for different kind of labels, for example use <code>label_parsed()</code> for formatting facet labels. <code>label_value()</code> 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 <code>strip.position</code> it is possible to place the labels on either of the four sides by setting <code>strip.position = c("top", "bottom", "left", "right")</code>
nest_line	a theme element, either <code>element_blank()</code> or inheriting from <code>ggplot2::element_line()</code> . Lines are drawn between layers of strips indicating hierarchy. The element inherits from the <code>ggh4x.facet.nestline</code> element in the theme.
solo_line	A <code>logical(1)</code> indicating whether parent strips with a single child should be drawn with a <code>nest_line</code> (TRUE) or the line only applies to parents with multiple children (FALSE, default). Only relevant when <code>nest_line</code> is drawn.
reset	a unit vector of length 1, indicating how much the nesting line should be shortened.
trim_blank	A <code>logical(1)</code> . When TRUE (default), does not draw rows and columns containing no panels. When FALSE, the <code>nrow</code> and <code>ncol</code> 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 <code>strip_nested()</code> .
bleed	<b>[Deprecated]</b> the <code>bleed</code> argument has moved to the <code>strip_nested()</code> 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 = strip_vanilla()
)

```

## Arguments

facets	<p>A set of variables or expressions quoted by <code>vars()</code> and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to <code>labeller</code>).</p> <p>For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, <math>\sim a + b</math>, or a character vector, <code>c("a", "b")</code>.</p>
nrow, ncol	Number of rows and columns.
scales	<p>A <code>character(1)</code> or <code>logical(1)</code> whether scales are shared across facets or allowed to vary. One of the following:</p> <p>"fixed" <b>or</b> FALSE Scales are shared across all facets (default).</p> <p>"free_x" x-scales are allowed to vary.</p> <p>"free_y" y-scales are allowed to vary.</p> <p>"free" <b>or</b> TRUE Both scales can vary</p>
axes	<p>A <code>character(1)</code> or <code>logical(1)</code> where axes should be drawn. One of the following:</p> <p>"margins" <b>or</b> FALSE Only draw axes at the outer margins (default).</p> <p>"x" Draw axes at the outer margins and all inner x-axes too.</p> <p>"y" Draw axes at the outer margins and all inner y-axes too.</p> <p>"all" <b>or</b> TRUE Draw the axes for every panel.</p>
remove_labels	<p>A <code>character(1)</code> or <code>logical(1)</code> determining whether axis text is displayed at inner panels. One of the following:</p> <p>"none" <b>or</b> FALSE Display axis text at all axes (default).</p> <p>"x" Display axis text at outer margins and all inner y-axes.</p> <p>"y" Display axis text at outer margins and all inner x-axes.</p> <p>"all" <b>or</b> TRUE Only display axis text at the outer margins.</p>
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	<p>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 <code>vars(cyl, am)</code>. 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 <code>labeller()</code>. You can use different labeling functions for different kind of labels, for example use <code>label_parsed()</code> for formatting facet labels. <code>label_value()</code> is used by default, check it for more details and pointers to other options.</p>

<code>as.table</code>	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.
<code>drop</code>	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.
<code>dir</code>	Direction: either "h" for horizontal, the default, or "v", for vertical.
<code>strip.position</code>	By default, the labels are displayed on the top of the plot. Using <code>strip.position</code> it is possible to place the labels on either of the four sides by setting <code>strip.position = c("top", "bottom", "left", "right")</code>
<code>trim.blank</code>	A <code>logical(1)</code> . When TRUE (default), does not draw rows and columns containing no panels. When FALSE, the <code>nrow</code> and <code>ncol</code> arguments are taken literally, even when there are more than needed to fit all panels.
<code>strip</code>	An object created by a call to a strip function, such as <a href="#">strip_vanilla</a> .

**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 faceted 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 faceting 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 `forcelsize` 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(displ, 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 <code>aes()</code> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
stat	<p>The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• A Stat ggproto subclass, for example <code>StatCount</code>.</li> <li>• A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as <code>"count"</code>.</li> </ul>

	<ul style="list-style-type: none"> <li>• For more information and other ways to specify the stat, see the <a href="#">layer stat</a> documentation.</li> </ul>
position	<p>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:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. 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.</li> <li>• When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept.</li> <li>• Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept.</li> <li>• The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as <a href="#">key glyphs</a>, to change the display of the layer in the legend.</li> </ul>
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. <code>borders()</code> .
radius	A <code>grid::unit</code> object of length 1 or <code>numeric(1)</code> to set rounded corners. If NULL (default), no rounded corners are applied. If <code>numeric(1)</code> , 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 <a href="#">aes()</a> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
stat	<p>The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• A Stat ggproto subclass, for example <code>StatCount</code>.</li> <li>• A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as "count".</li> <li>• For more information and other ways to specify the stat, see the <a href="#">layer stat</a> documentation.</li> </ul>
position	<p>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 <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <a href="#">layer()</a>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is</li> </ul>

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.

<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	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.
<code>inherit.aes</code>	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. <code>borders()</code> .

## 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 <a href="#">aes()</a> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> 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	<p>The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• A Stat ggproto subclass, for example <code>StatCount</code>.</li> <li>• A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as "count".</li> <li>• For more information and other ways to specify the stat, see the <a href="#">layer stat</a> documentation.</li> </ul>
position	<p>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 <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. 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.</li> <li>• When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept.</li> <li>• Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept.</li> <li>• The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as <a href="#">key glyphs</a>, to change the display of the layer in the legend.</li> </ul>

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 <code>grid::arrow()</code> .
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. <code>borders()</code> .

### 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 <a href="#">aes()</a> . 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 <a href="#">ggplot()</a> . A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> 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. <code>~ head(.x, 10)</code> ).
stat	The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following:

	<ul style="list-style-type: none"> <li>• A Stat ggproto subclass, for example StatCount.</li> <li>• A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as "count".</li> <li>• For more information and other ways to specify the stat, see the <a href="#">layer stat</a> documentation.</li> </ul>
position	<p>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 <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. 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.</li> <li>• When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept.</li> <li>• Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept.</li> <li>• The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as <a href="#">key glyphs</a>, to change the display of the layer in the legend.</li> </ul>
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.



---

geom_rectmargin	<i>Rectangular rugs in the margins</i>
-----------------	--

---

## 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 = "b1",  
  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 = "b1",  
  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	<p>The data to be displayed in this layer. There are three options:</p> <p>If NULL, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>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. <code>~ head(.x, 10)</code>).</p>
stat	<p>The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• A Stat ggproto subclass, for example <code>StatCount</code>.</li> <li>• A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as "count".</li> <li>• For more information and other ways to specify the stat, see the <a href="#">layer stat</a> documentation.</li> </ul>
position	<p>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 <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. 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.</li> <li>• When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept.</li> <li>• Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer.</li> </ul>

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.

<code>outside</code>	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 <code>coord_cartesian(clip = "off")</code> and avoid overlap with the default axes by changing the sides argument to <code>"tr"</code> .
<code>sides</code>	A string of length 1 that controls which sides of the plot the rug-rectangles appear on. A string containing any letters in <code>"trbl"</code> will set it to top, right, bottom and left respectively.
<code>length</code>	A <code>grid::unit()</code> 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.
<code>linejoin</code>	Line join style (round, mitre, bevel).
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	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.
<code>inherit.aes</code>	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. <code>borders()</code> .

## 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	<i>Aimed text</i>
-----------------	-------------------

---

### 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 <code>aes()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply <code>mapping</code> if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> 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	<p>The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• A Stat ggproto subclass, for example <code>StatCount</code>.</li> <li>• A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as "count".</li> <li>• For more information and other ways to specify the stat, see the <a href="#">layer stat</a> documentation.</li> </ul>
position	<p>A position adjustment to use on the data for this layer. Cannot be jointly specified with <code>nudge_x</code> or <code>nudge_y</code>. This can be used in various ways, including to prevent overplotting and improving the display. The <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <code>position_jitter()</code>.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. 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.</li> <li>• When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept.</li> <li>• Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept.</li> <li>• The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as <a href="#">key glyphs</a>, to change the display of the layer in the legend.</li> </ul>

parse	If TRUE, the labels will be parsed into expressions and displayed as described in <code>?plotmath</code> .
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 <code>position</code> .
flip_upsidedown	A <code>logical(1)</code> . 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. <code>check_overlap</code> happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling <code>geom_text()</code> . Note that this argument is not supported by <code>geom_label()</code> .
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. <code>borders()</code> .

## 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` 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](#)

---

`ggsubset`*Passing a subset of data to ggplot2 layers.*

---

### Description

This is a convenience function to allow layer objects, such as geoms, to take a subset of the data in the main `ggplot()` call, without storing a duplicate of the subset in the `ggplot` object.

### Usage

```
ggsubset(rowtest = NULL, omit = NULL)
```

### Arguments

<code>rowtest</code>	logical expression indicating which rows to keep.
<code>omit</code>	a character column name to exclude.

### Details

`ggsubset` is a wrapper around `subset.data.frame` where the `subset` argument is set to `rowtest` and the `select` argument to `-omit`. Since the `data` argument in the `layer()` function can take a function with one argument, we can pass the function returned from `ggsubset` as that argument to subset the data by rows.

### Value

A function that takes a `data.frame` as argument and returns a subset of that `data.frame` according to `rowtest`

### See Also

See [ggplot2::layer\(\)](#), specifically the `data` argument. See [subset.data.frame\(\)](#) for the internal function.

### Examples

```
ggplot(iris, aes(Sepal.Width, Sepal.Length)) +  
  geom_point(data = ggsubset(Species == "setosa"))
```

---

guide\_axis\_logticks    *Axis guide with ticks for logarithmic breaks*

---

## Description

**[Questioning]** This axis guide is probably best described as `ggplot2::annotation_logticks()` but implemented as a guide instead of a geom. The tick marks probably best suit log10 transformations. The function is questioned due to a possible migration of guide functions after ggplot2 releases a new guide system.

## Usage

```
guide_axis_logticks(
  title = waiver(),
  check.overlap = FALSE,
  angle = NULL,
  n.dodge = 1,
  order = 0,
  position = waiver(),
  prescaled = FALSE,
  trunc_lower = NULL,
  trunc_upper = NULL,
  colour = NULL,
  color = NULL,
  base = waiver()
)
```

## Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title.
check.overlap	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.
angle	Compared to setting the angle in <code>theme()</code> / <code>element_text()</code> , this also uses some heuristics to automatically pick the <code>hjust</code> and <code>vjust</code> that you probably want. Can be one of the following: <ul style="list-style-type: none"> <li>• NULL to take the angles and <code>hjust/vjust</code> directly from the theme.</li> <li>• <code>waiver()</code> to allow reasonable defaults in special cases.</li> <li>• A number representing the text angle in degrees.</li> </ul>
n.dodge	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.

order	A positive integer of length 1 that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a secret algorithm.
position	Where this guide should be drawn: one of top, bottom, left, or right.
prescaled	A logical of length one, indicating whether the data has been manually rescaled (TRUE) or the scale takes care of the transformation (FALSE).
trunc_lower, trunc_upper	The lower and upper range of the truncated axis: <ul style="list-style-type: none"> <li>• NULL to not perform any truncation.</li> <li>• A function that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as numeric.</li> <li>• A numeric value in data units for the lower and upper boundaries.</li> <li>• A unit object.</li> </ul>
colour, color	A character(1) with a valid colour for colouring the axis text, axis ticks and axis line. Overrides the colour assigned by the theme.
base	When this is provided, the guide takes this as the base for the log transformation instead of trying to guess the base. It is recommended to use this argument if the base is not 10.

**Value**

An *axis\_logticks* guide class object.

**Theme elements**

This axis guide uses the following the theme elements:

`ggh4x.axis.ticks.length.minor` An `rel()` object to set the size of minor ticks relative to the length of major ticks (`axis.ticks.length`). Defaults to `rel(2/3)`.

`ggh4x.axis.ticks.length.mini` An `rel()` object to set the size of smallest ticks, also relative to the length of major ticks (`axis.ticks.length`). Defaults to `rel(1/3)`.

**See Also**

Other axis-guides: `guide_axis_manual()`, `guide_axis_minor()`, `guide_axis_nested()`, `guide_axis_scalebar()`, `guide_axis_truncated()`

**Examples**

```
# The guide works well out of the box with log10 scales
p <- ggplot(pressure, aes(temperature, pressure)) +
  geom_line()
p + scale_y_log10(guide = "axis_logticks")

# If the data is already transformed, you can set 'prescaled' to TRUE
ggplot(pressure, aes(temperature, log10(pressure))) +
```

```

geom_line() +
guides(y = guide_axis_logticks(prescaled = TRUE))

# The lengths of the log-ticks are controlled by the theme relative to the
# major ticks.
p + scale_y_log10(guide = "axis_logticks") +
  theme(
    axis.ticks.length.y = unit(1, "cm"),
    ggh4x.axis.ticks.length.minor = rel(0.55),
    ggh4x.axis.ticks.length.mini = rel(0.1)
  )

```

---

guide\_axis\_manual      *Manual axis*

---

### Description

**[Questioning]** This axis allows a greater degree of control than the default axes guides. In particular, this axis allows setting break positions and labels independently from the scale and is not bound by the same constraints as secondary axes. Additionally, label attributes may be set in parallel to the labels themselves, circumventing the unsupported vectorised input to `element_text()`. The function is questioned due to a possible migration of guide functions after `ggplot2` releases a new guide system.

### Usage

```

guide_axis_manual(
  title = waiver(),
  breaks = waiver(),
  labels = waiver(),
  label_family = NULL,
  label_face = NULL,
  label_colour = NULL,
  label_size = NULL,
  label_hjust = NULL,
  label_vjust = NULL,
  label_lineheight = NULL,
  label_color = NULL,
  label_margin = NULL,
  check_overlap = FALSE,
  angle = NULL,
  n.dodge = 1,
  order = 0,
  colour = NULL,
  color = NULL,
  trunc_lower = NULL,
  trunc_upper = NULL,
  position = waiver()
)

```

**Arguments**

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title.
breaks	One of the following ways to parametrise the tick and label positions: <ul style="list-style-type: none"> <li>• NULL to draw no ticks and labels.</li> <li>• <code>waiver()</code> for the default breaks computed by the scale (default).</li> <li>• A numeric vector for continuous or discrete scales, or a character vector for discrete scales.</li> <li>• A function that takes the limits as input and returns breaks as output. Also accepts rlang <code>lambda</code> notation.</li> <li>• A <code>unit</code> vector for setting data-independent breaks.</li> </ul>
labels	One of the following ways to dictate the labels: <ul style="list-style-type: none"> <li>• NULL to draw no labels.</li> <li>• <code>waiver()</code> for the default labels computed by the scale on the breaks (default). Note that a scale with non-identity transformation is unlikely to graciously handle breaks defined in grid-units.</li> <li>• A character vector giving</li> <li>• A function that takes the breaks as input and returns labels as output. Also accepts rlang <code>lambda</code> notation.</li> </ul>
label_family, label_face, label_colour, label_size, label_hjust, label_vjust, label_lineheight, label_color, label_margin	Arguments passed down to the label constructor. See <code>element_text()</code> arguments, which these arguments mirror with the <code>label_</code> -prefix. With the exception of <code>label_margin</code> , the other <code>label_*</code> arguments are assumed be parallel to (the result of) the <code>labels</code> argument and will be recycled with <code>rep_len()</code> as necessary. By default, these parameters are taken from the theme.
check.overlap	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.
angle	Compared to setting the angle in <code>theme()</code> / <code>element_text()</code> , this also uses some heuristics to automatically pick the <code>hjust</code> and <code>vjust</code> that you probably want. Can be one of the following: <ul style="list-style-type: none"> <li>• NULL to take the angles and <code>hjust/vjust</code> directly from the theme.</li> <li>• <code>waiver()</code> to allow reasonable defaults in special cases.</li> <li>• A number representing the text angle in degrees.</li> </ul>
n.dodge	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.
order	A positive integer of length 1 that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a secret algorithm.
colour, color	A <code>character(1)</code> with a valid colour for colouring the axis text, axis ticks and axis line. Overrides the colour assigned by the theme.

trunc\_lower, trunc\_upper

The lower and upper range of the truncated axis:

- NULL to not perform any truncation.
- A function that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as numeric.
- A numeric value in data units for the lower and upper boundaries.
- A unit object.

position Where this guide should be drawn: one of top, bottom, left, or right.

### Value

An *axis\_manual* guide class object.

### See Also

Other axis-guides: [guide\\_axis\\_logticks\(\)](#), [guide\\_axis\\_minor\(\)](#), [guide\\_axis\\_nested\(\)](#), [guide\\_axis\\_scalebar\(\)](#), [guide\\_axis\\_truncated\(\)](#)

### Examples

```
ggplot(iris, aes(Species, Sepal.Width)) +
  geom_boxplot(aes(fill = Species)) +
  guides(x = guide_axis_manual(
    label_colour = scales::hue_pal()(3),
    label_face = c("bold", "italic", "plain"),
    labels = toupper
  ))

# Using the manual axis to annotate some specific point
ggplot(pressure, aes(temperature, pressure)) +
  geom_point() +
  geom_hline(yintercept = 300, linetype = 2, colour = "blue") +
  guides(y.sec = guide_axis_manual(breaks = 300, labels = "some\ntreshold",
    label_colour = "blue"))
```

---

guide\_axis\_minor

*Axis guide with ticks for minor breaks*

---

### Description

**[Questioning]** These are similar the the normal axis guides for position scales, but also place tick-marks at minor break positions. The function is questioned due to a possible migration of guide functions after ggplot2 releases a new guide system.

**Usage**

```
guide_axis_minor(
  title = waiver(),
  check.overlap = FALSE,
  angle = NULL,
  n.dodge = 1,
  order = 0,
  colour = NULL,
  color = NULL,
  trunc_lower = NULL,
  trunc_upper = NULL,
  position = waiver()
)
```

**Arguments**

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title.
check.overlap	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.
angle	Compared to setting the angle in <code>theme()</code> / <code>element_text()</code> , this also uses some heuristics to automatically pick the <code>hjust</code> and <code>vjust</code> that you probably want. Can be one of the following: <ul style="list-style-type: none"> <li>• NULL to take the angles and <code>hjust/vjust</code> directly from the theme.</li> <li>• <code>waiver()</code> to allow reasonable defaults in special cases.</li> <li>• A number representing the text angle in degrees.</li> </ul>
n.dodge	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.
order	A positive integer of length 1 that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a secret algorithm.
colour, color	A character(1) with a valid colour for colouring the axis text, axis ticks and axis line. Overrides the colour assigned by the theme.
trunc_lower, trunc_upper	The lower and upper range of the truncated axis: <ul style="list-style-type: none"> <li>• NULL to not perform any truncation.</li> <li>• A function that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as numeric.</li> <li>• A numeric value in data units for the lower and upper boundaries.</li> <li>• A unit object.</li> </ul>
position	Where this guide should be drawn: one of top, bottom, left, or right.

**Value**

An *axis\_minor* guide class object.

**Theme elements**

This axis guide uses the following the theme elements:

`ggh4x.axis.ticks.length.minor` An `rel()` object to set the size of minor ticks relative to the length of major ticks (`axis.ticks.length`). Defaults to `rel(2/3)`.

**See Also**

Other axis-guides: `guide_axis_logticks()`, `guide_axis_manual()`, `guide_axis_nested()`, `guide_axis_scalebar()`, `guide_axis_truncated()`

**Examples**

```
# Using the minor breaks axis
p <- ggplot(iris, aes(Sepal.Width, Sepal.Length)) +
  geom_point()
p + scale_y_continuous(guide = "axis_minor")

# Minor break positions are still controlled by the scale
p + scale_y_continuous(guide = "axis_minor",
  minor_breaks = seq(4, 8, by = 0.2))

# Minor tick length is controlled relative to major ticks
p + scale_y_continuous(guide = "axis_minor") +
  theme(ggh4x.axis.ticks.length.minor = rel(0.1))
```

---

guide_axis_nested	<i>Nested axis guide</i>
-------------------	--------------------------

---

**Description**

**[Questioning]** Discrete position scales containing interacting factors can be visualised more clearly with a nested axis guide. Nested axis guides separate labels based on a delimiter and groups identical later labels, indicating the grouping with a line spanning the earlier labels. The function is questioned due to a possible migration of guide functions after ggplot2 releases a new guide system.

**Usage**

```
guide_axis_nested(
  title = waiver(),
  check.overlap = FALSE,
  angle = NULL,
  n.dodge = 1,
```

```

order = 0,
position = waiver(),
delim = waiver(),
inv = FALSE,
trunc_lower = NULL,
trunc_upper = NULL,
colour = NULL,
color = NULL,
extend = 0.5
)

```

### Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title.
check.overlap	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.
angle	Compared to setting the angle in <code>theme()</code> / <code>element_text()</code> , this also uses some heuristics to automatically pick the <code>hjust</code> and <code>vjust</code> that you probably want. Can be one of the following: <ul style="list-style-type: none"> <li>• NULL to take the angles and <code>hjust/vjust</code> directly from the theme.</li> <li>• <code>waiver()</code> to allow reasonable defaults in special cases.</li> <li>• A number representing the text angle in degrees.</li> </ul>
n.dodge	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.
order	A positive integer of length 1 that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a secret algorithm.
position	Where this guide should be drawn: one of top, bottom, left, or right.
delim	A character of length 1 to tell <code>strsplit</code> how hierarchies should be broken up. Internally defaults to "." to match <code>interaction</code> 's default delimiter.
inv	A <code>logical(1)</code> which if TRUE, flips the grouping order. If FALSE (default), the grouping order is as-is.
trunc_lower, trunc_upper	The lower and upper range of the truncated axis: <ul style="list-style-type: none"> <li>• NULL to not perform any truncation.</li> <li>• A function that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as numeric.</li> <li>• A numeric value in data units for the lower and upper boundaries.</li> <li>• A unit object.</li> </ul>

colour, color	A character(1) with a valid colour for colouring the axis text, axis ticks and axis line. Overrides the colour assigned by the theme.
extend	A numeric of length 1 indicating how much to extend nesting lines relative to the smallest difference in break positions.

### Details

The guide itself makes no effort to group and order labels. To get nice groupings, consider re-ordering the levels of factor variables, or try setting the 'breaks' argument of a scale appropriately.

### Value

A *axis\_nested* guide class object.

### Theme elements

This axis guide uses the following the theme elements:

`ggh4x.axis.nestline.x/y` An `element_line()` object to alter the display of the line separating different layers of labels.

`ggh4x.axis.nesttext.x/y` An `element_text()` object to differentiate text higher up in the hierarchy, from the text closest to the axis line.

### See Also

`ggplot2::guide_axis()` for the classic axis documentation.

`weave_factors()` for an alternative to `interaction()`.

Other axis-guides: `guide_axis_logticks()`, `guide_axis_manual()`, `guide_axis_minor()`, `guide_axis_scalebar()`, `guide_axis_truncated()`

### Examples

```
# The defaults are suited for interaction variables
ggplot(mpg, aes(interaction(cyl, class), hwy)) +
  geom_boxplot() +
  scale_x_discrete(guide = "axis_nested")

# Control where labels are cut with the delim argument
ggplot(mpg, aes(interaction(cyl, class, sep = "~!~"), hwy)) +
  geom_boxplot() +
  scale_x_discrete(guide = guide_axis_nested(delim = "!"))

# The nesting lines inherit looks from axis ticks
ggplot(mpg, aes(interaction(cyl, class), hwy)) +
  geom_boxplot() +
  scale_x_discrete(guide = "axis_nested") +
  theme(axis.ticks = element_line(colour = "red"))

# The looks can be controlled independently by setting `ggh4x.axis.nestline`
ggplot(mpg, aes(interaction(cyl, class), hwy)) +
```

```
geom_boxplot() +  
scale_x_discrete(guide = "axis_nested") +  
theme(ggh4x.axis.nestline = element_line(linetype = 2))
```

---

guide\_axis\_scalebar    *Scale bar axis guide*

---

## Description

**[Questioning]** This axis guides draws a scale bar to indicate a distance rather than mark absolute values. The function is questioned due to a possible migration of guide functions after ggplot2 releases a new guide system.

## Usage

```
guide_axis_scalebar(  
  title = waiver(),  
  size = NULL,  
  label = NULL,  
  colour = NULL,  
  color = NULL,  
  just = 1,  
  position = waiver()  
)
```

## Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title.
size	A <code>numeric(1)</code> for a distance to indicate, in data units. If NULL (default), the median distance between breaks is taken.
label	A <code>character(1)</code> to use as scale bar label. If NULL (default), the size distance is fed to the scale's labeller.
colour, color	A <code>character(1)</code> with a valid colour for colouring the axis text, axis ticks and axis line. Overrides the colour assigned by the theme.
just	A <code>numeric(1)</code> between 0 and 1 to indicate where the scalebar should be drawn relative to the plot panel. The default (1), places is at the right or at the top.
position	Where this guide should be drawn: one of top, bottom, left, or right.

## Details

It is discouraged to use this guide in combination with a scale transformation.

## Value

A `axis_scalebar` guide class object.

### Theme elements

This axis guide has an alternative understanding of the following theme elements:

`axis.ticks`.\* An `element_line()` to draw the scale bar itself.

`axis.ticks.length`.\* A `unit()` indicating how far the scale bar should be placed from the plot panel. Can be a negative unit to place the scale bar inside the plot panel.

`axis.text`.\* The `hjust` and `vjust` parameters are used to justify the text along the scale bar, instead of along itself, in the x and y directions respectively.

### See Also

Other axis-guides: [guide\\_axis\\_logticks\(\)](#), [guide\\_axis\\_manual\(\)](#), [guide\\_axis\\_minor\(\)](#), [guide\\_axis\\_nested\(\)](#), [guide\\_axis\\_truncated\(\)](#)

### Examples

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

# Guide as secondary axis
p + guides(x.sec = "axis_scalebar")

# Customising size and label
p + guides(x.sec = guide_axis_scalebar(size = 0.5, label = "0.5 litre"))

# Placing the scale bar on top of the plotting panel
p + guides(x.sec = guide_axis_scalebar(just = 0.95)) +
  theme(axis.ticks.length.x.top = unit(-2, "lines"))

# Adding arrows through the axis.ticks theme element
p + guides(y.sec = guide_axis_scalebar(size = 10, label = "10\nmpg")) +
  theme(axis.ticks.y.right = element_line(arrow = arrow(ends = "both")))
```

---

guide\_axis\_truncated *Axis guide with truncated line*

---

### Description

**[Questioning]** This axis guide is similar to the normal axis guides for position scales, but can shorten the axis line that is being drawn. The `guide_axis_colour()` function is the same but with different defaults for the truncation that do not truncate the axis. Axis truncation and recolouring is supported throughout axes in `ggh4x`. The function is questioned due to a possible migration of guide functions after `ggplot2` releases a new guide system.

**Usage**

```
guide_axis_truncated(  
  title = waiver(),  
  check.overlap = FALSE,  
  angle = NULL,  
  n.dodge = 1,  
  order = 0,  
  colour = NULL,  
  color = NULL,  
  trunc_lower = min,  
  trunc_upper = max,  
  position = waiver()  
)
```

```
guide_axis_colour(  
  title = waiver(),  
  check.overlap = FALSE,  
  angle = NULL,  
  n.dodge = 1,  
  order = 0,  
  colour = NULL,  
  color = NULL,  
  trunc_lower = NULL,  
  trunc_upper = NULL,  
  position = waiver()  
)
```

```
guide_axis_color(  
  title = waiver(),  
  check.overlap = FALSE,  
  angle = NULL,  
  n.dodge = 1,  
  order = 0,  
  colour = NULL,  
  color = NULL,  
  trunc_lower = NULL,  
  trunc_upper = NULL,  
  position = waiver()  
)
```

**Arguments**

- |               |   |
|---------------|---|
| title         | A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title. |
| check.overlap | silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.  |

angle	Compared to setting the angle in <code>theme()</code> / <code>element_text()</code> , this also uses some heuristics to automatically pick the <code>hjust</code> and <code>vjust</code> that you probably want. Can be one of the following: <ul style="list-style-type: none"> <li>• <code>NULL</code> to take the angles and <code>hjust/vjust</code> directly from the theme.</li> <li>• <code>waiver()</code> to allow reasonable defaults in special cases.</li> <li>• A number representing the text angle in degrees.</li> </ul>
n.dodge	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.
order	A positive integer of length 1 that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a secret algorithm.
colour, color	A <code>character(1)</code> with a valid colour for colouring the axis text, axis ticks and axis line. Overrides the colour assigned by the theme.
trunc_lower, trunc_upper	The lower and upper range of the truncated axis: <ul style="list-style-type: none"> <li>• <code>NULL</code> to not perform any truncation.</li> <li>• A function that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as numeric.</li> <li>• A numeric value in data units for the lower and upper boundaries.</li> <li>• A unit object.</li> </ul>
position	Where this guide should be drawn: one of top, bottom, left, or right.

**Value**

An `axis_ggh4x` guide class object.

**See Also**

Other axis-guides: [guide\\_axis\\_logticks\(\)](#), [guide\\_axis\\_manual\(\)](#), [guide\\_axis\\_minor\(\)](#), [guide\\_axis\\_nested\(\)](#), [guide\\_axis\\_scalebar\(\)](#)

**Examples**

```
# Make a plot
p <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point() +
  theme(axis.line = element_line(colour = "black"))

# Setting the default truncated axis
p + guides(x = "axis_truncated")

# Truncating in data units
p + guides(x = guide_axis_truncated(
  trunc_lower = 2.5, trunc_upper = 4.5
```

```

))

# Truncate by setting units
p + guides(x = guide_axis_truncated(
  trunc_lower = unit(0.1, "npc"),
  trunc_upper = unit(0.9, "npc")
))

# Truncating with functions
p + guides(x = guide_axis_truncated(
  trunc_lower = function(x) {x - 0.2},
  trunc_upper = function(x) {x + 0.2}
))

# Recolouring the axes outside the theme
p + guides(x = guide_axis_colour(colour = "red"),
  y = guide_axis_colour(colour = "blue"))

```

---

guide\_dendro

*Dendrogram guide*


---

### Description

Visual representation of a discrete variable with hierarchical relationships between members, like those detailed in `scale_(x|y)_dendrogram()`.

### Usage

```

guide_dendro(
  title = waiver(),
  check.overlap = FALSE,
  n.dodge = 1,
  order = 0,
  position = waiver(),
  label = TRUE,
  trunc_lower = NULL,
  trunc_upper = NULL,
  colour = NULL,
  color = NULL,
  dendro = waiver()
)

```

### Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title.
check.overlap	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.



```
# The looks of the dendrogram are controlled through ticks
g + theme(axis.ticks = element_line(colour = "red"))

# The size of the dendrogram is controlled through tick size * 10
g + theme(axis.ticks.length = unit(5, "pt"))
```

---

guide\_stringlegend      *String legend*

---

### Description

**[Questioning]** This type of legend shows colour and fill mappings as coloured text. It does not draw keys as `guide_legend()` does. The function is questioned due to a possible migration of guide functions after ggplot2 releases a new guide system.

### Usage

```
guide_stringlegend(
  title = waiver(),
  title.position = NULL,
  title.theme = NULL,
  title.hjust = NULL,
  title.vjust = NULL,
  label.theme = NULL,
  label.hjust = NULL,
  label.vjust = NULL,
  family = NULL,
  face = NULL,
  size = NULL,
  spacing.x = NULL,
  spacing.y = NULL,
  spacing = NULL,
  default.units = "pt",
  direction = NULL,
  nrow = NULL,
  ncol = NULL,
  byrow = FALSE,
  reverse = FALSE,
  order = 0,
  ...
)
```

### Arguments

<code>title</code>	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title.
--------------------	---

<code>title.position</code>	A character string indicating the position of a title. One of "top", "bottom", "left" or "right".
<code>title.theme</code>	A theme object for rendering the title text.
<code>title.hjust</code> , <code>title.vjust</code>	A number specifying horizontal and vertical justification of the title text.
<code>label.theme</code>	A theme object for rendering the legend text.
<code>label.hjust</code> , <code>label.vjust</code>	A numer specifying horizontal and vertical justification of the legend text.
<code>family</code>	A character(1) setting a font family for labels.
<code>face</code>	A character(1) setting a font face for labels. One of the following: "plain", "italic" or "bold", "bold.italic".
<code>size</code>	A numeric(1) setting the label text size in pts.
<code>spacing.x</code> , <code>spacing.y</code> , <code>spacing</code>	A numeric(1) or unit for the spacing between label rows and columns. Internally defaults to half the size of the title.
<code>default.units</code>	A character(1) indicating the default units to use if the <code>spacing.*</code> arguments are only given as numeric vectors.
<code>direction</code>	A character string indicating the direction of the guide. One of "horizontal" or "vertical."
<code>nrow</code> , <code>ncol</code>	The desired number of rows and column of legends respectively.
<code>byrow</code>	logical. If FALSE (default) the legend-matrix is filled by columns, otherwise the legend-matrix is filled by rows.
<code>reverse</code>	logical. If TRUE the order of legends is reversed.
<code>order</code>	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.
<code>...</code>	ignored.

**Value**

A guide, stringlegend S3 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	<i>Secondary axis helper</i>
----------------	------------------------------

---

## 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 <code>lm(primary ~ secondary)</code> 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 <code>logical(1)</code> : whether to remove missing values (TRUE) or propagate missing values (FALSE). Applies to the <code>method = "range"</code> and <code>method = "max"</code> methods.
...	Arguments passed on to <code>ggplot2::sec_axis</code>
transform	A formula or function of a strictly monotonic transformation
name	The name of the secondary axis
breaks	One of: <ul style="list-style-type: none"> <li>• NULL for no breaks</li> <li>• <code>waiver()</code> for the default breaks computed by the transformation object</li> <li>• A numeric vector of positions</li> <li>• A function that takes the limits as input and returns breaks as output</li> </ul>

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

<code>extend</code>	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.
<code>stepsize</code>	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:

$$\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

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:

$$\begin{vmatrix} 2 & 0 \\ 0 & 2 \end{vmatrix}$$

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:

$$\begin{vmatrix} 2 & 0 \\ 0 & 0.5 \end{vmatrix}$$

or

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

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

$$\begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix}$$

or

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

Reflecting around the y-axis:

$$\begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix}$$

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:

$$\begin{array}{cccc} | & 0 & 0 & | \\ | & 0 & 1 & | \end{array}$$

or

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

Projecting onto the x-axis:

$$\begin{array}{cccc} | & 1 & 0 & | \\ | & 0 & 0 & | \end{array}$$

or

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

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

$$\begin{array}{cccc} | & 1 & 0 & | \\ | & 0.1 & 1 & | \end{array}$$

or

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

Shearing horizontally by 200\

$$\begin{array}{cccc} | & 1 & 2 & | \\ | & 0 & 1 & | \end{array}$$

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:

$$\begin{array}{c|cc|c} & 0 & -1 & \\ \hline & 1 & 0 & \\ \hline \end{array}$$

or

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

For a rotation around any angle  $\theta$  :

$$\begin{array}{c|cc|c} & \cos\theta & -\sin\theta & \\ \hline & \sin\theta & \cos\theta & \\ \hline \end{array}$$

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\_dendrogram      *Dendrogram position scales*

---

### Description

When discrete data has some inherent hierarchy to the relationship between discrete categories, you can display a dendrogram instead of a tick axis.

### Usage

```
scale_x_dendrogram(
  ...,
  hclust = waiver(),
  expand = waiver(),
  guide = waiver(),
  position = "bottom"
)

scale_y_dendrogram(
  ...,
  hclust = waiver(),
  expand = waiver(),
  guide = waiver(),
  position = "left"
)
```

### Arguments

... Arguments passed on to `ggplot2::discrete_scale`

aesthetics The names of the aesthetics that this scale works with.

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

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take (e.g., `scales::pal_hue()`).

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.

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 `?plot-math` for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang `lambda` function notation.

	<p><code>limits</code> One of:</p> <ul style="list-style-type: none"> <li>• NULL to use the default scale values</li> <li>• A character vector that defines possible values of the scale and their order</li> <li>• A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang <code>lambda</code> function notation.</li> </ul> <p><code>na.translate</code> 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 <code>na.translate = FALSE</code>.</p> <p><code>na.value</code> If <code>na.translate = TRUE</code>, 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.</p> <p><code>drop</code> 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 <code>show.legend = TRUE</code>.</p> <p><code>call</code> The call used to construct the scale for reporting messages.</p> <p><code>super</code> The super class to use for the constructed scale</p>
<code>hclust</code>	An object of the type produced by the <code>stats::hclust()</code> function.
<code>expand</code>	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 <code>expansion()</code> to generate the values for the <code>expand</code> 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.
<code>guide</code>	A function used to create a guide or its name. See <code>guides()</code> for more information.
<code>position</code>	For position scales, The position of the axis. <code>left</code> or <code>right</code> for y axes, <code>top</code> or <code>bottom</code> for x axes.

## Details

The dendrogram type of scale does two things, first it reorders the values along the relevant direction such that they follow the order captured in the `hclust` argument. Secondly, it draws the dendrogram at the axis. The dendrogram visuals inherit from the ticks theme elements, so defining a `linetype` for the tick marks sets the `linetype` for the dendrogram.

## Value

A *ScaleDendrogram* ggproto object.

## Examples

```
# Hierarchically cluster USArrests
yclus <- hclust(dist(USArrests), "ave")
xclus <- hclust(dist(t(USArrests)), "ave")

# Melting USArrests
```

```
df <- data.frame(
  State = rownames(USArrests)[row(USArrests)],
  variable = colnames(USArrests)[col(USArrests)],
  value = unname(do.call(c, USArrests))
)

# Supply the clustering to the scales
ggplot(df, aes(variable, State, fill = value)) +
  geom_raster() +
  scale_y_dendrogram(hclust = yclus) +
  scale_x_dendrogram(hclust = xclus)
```

---

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 `faceted_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 faceted 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 [faceted\\_pos\\_scales\(\)](#) function.

### Examples

```
# A standard plot with continuous scales
p <- ggplot(mtcars, aes(displ, 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 aesthetics.

In contrast to other `scale_*_continuous`-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
iriscol <- cor(t(iris[, 1:4]))

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

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")
  )
```

**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 `?plot-math` 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))

# Together with grouped axis
p + scale_x_manual(values = sep_discrete(), guide = guide_axis_nested())
```

---

sep_discrete	<i>Separator for discrete grouped labels</i>
--------------	--

---

**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	<i>Difference ribbon</i>
-----------------	--------------------------

---

**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 <a href="#">aes()</a> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	Use to override the default connection between <a href="#">geom_ribbon()</a> and <a href="#">stat_difference()</a> .
position	<p>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 <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <a href="#">position_jitter()</a>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <a href="#">position_jitter()</a>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <a href="#">layer()</a>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The <code>geom</code>'s documentation has an <b>Aesthetics</b> section that lists the</li> </ul>

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.

<code>levels</code>	A character(2) indicating factor levels for the fill aesthetic for the cases where (1) <code>max &gt; min</code> and where (2) <code>max &lt; min</code> .
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>orientation</code>	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 <code>orientation</code> to either "x" or "y". See the <i>Orientation</i> section for more detail.
<code>show.legend</code>	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.
<code>inherit.aes</code>	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. <code>borders()</code> .

## 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 <code>aes()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
geom	The geometric object to use to display the data for this layer. When using a <code>stat_*()</code> function to construct a layer, the <code>geom</code> argument can be used to override the default coupling between stats and geoms. The <code>geom</code> argument accepts the following: <ul style="list-style-type: none"> <li>• A <code>Geom</code> ggproto subclass, for example <code>GeomPoint</code>.</li> <li>• A string naming the geom. To give the geom as a string, strip the function name of the <code>geom_</code> prefix. For example, to use <code>geom_point()</code>, give the</li> </ul>

	geom as "point".
	<ul style="list-style-type: none"> <li>For more information and other ways to specify the geom, see the <a href="#">layer geom</a> documentation.</li> </ul>
position	<p>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:</p> <ul style="list-style-type: none"> <li>The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position.</li> <li>A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter".</li> <li>For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. 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.</li> <li>When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept.</li> <li>Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept.</li> <li>The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as <a href="#">key glyphs</a>, to change the display of the layer in the legend.</li> </ul>
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. <code>borders()</code> .

**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 <a href="#">aes()</a> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	Use to override the default connection between <a href="#">geom_rect()</a> and <a href="#">stat_rle()</a> .
position	<p>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 <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <a href="#">position_jitter()</a>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <a href="#">position_jitter()</a>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <a href="#">layer()</a>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. 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.</li> <li>• When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept.</li> <li>• Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer.</li> </ul>

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.

<code>align</code>	A character of length one that effect the computed start and end variables. One of the following: <code>"none"</code> Take exact start and end x values. <code>"center"</code> Return start and end x values in between an end and the subsequent start. <code>"start"</code> Align start values with previous end values. <code>"end"</code> Align end values with next start values.
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>orientation</code>	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 <code>orientation</code> to either <code>"x"</code> or <code>"y"</code> . See the <i>Orientation</i> section for more detail.
<code>show.legend</code>	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.
<code>inherit.aes</code>	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. <code>borders()</code> .

### Details

The data is first ordered on the x aesthetic before run lengths are calculated for the `label` aesthetic. In contrast to `base::rle()`, NAs 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 <a href="#">aes()</a> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	Use to override the default geom ("line").
position	<p>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 <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <a href="#">position_jitter()</a>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <a href="#">position_jitter()</a>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <a href="#">layer()</a>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the</li> </ul>

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: <ul style="list-style-type: none"> <li>• A numeric of length one indicating a measure of kernel width, in data units.</li> <li>• A <code>rel</code> object of length one constructed for setting a bandwidth relative to the group data range. Can be constructed with the <code>rel()</code> function.</li> <li>• A character of length one, naming one of the functions documented in <a href="#">bw.nrd()</a>.</li> </ul>
kernel	One of the following: <ul style="list-style-type: none"> <li>• A function that takes a vector of distances as first argument, a numeric bandwidth as second argument and returns relative weights.</li> <li>• A character of length one that can take one of the following values:           <ul style="list-style-type: none"> <li>"gaussian" <b>or</b> "norm" A kernel that follows a normal distribution with 0 mean and bandwidth as standard deviation.</li> <li>"mean" <b>or</b> "unif" A kernel that follows a uniform distribution with <math>bandwidth * -0.5</math> and <math>bandwidth * 0.5</math> as minimum and maximum. This is similar to a simple, unweighted moving average.</li> <li>"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.</li> </ul> </li> </ul>
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. <a href="#">borders()</a> .

**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	<i>Fitted theoretical density</i>
------------------	-----------------------------------

---

**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 <a href="#">aes()</a> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	Use to override the default geom for <code>stat_theodensity</code> .
position	<p>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 <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter".</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
...	<p>Other arguments passed on to <a href="#">layer()</a>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• 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, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. 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.</li> </ul>

- 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.

<code>distri</code>	A character of length 1 naming a distribution without prefix. See details.
<code>n</code>	An integer of length 1 with the number of equally spaced points at which the density function is evaluated. Ignored if distribution is discrete.
<code>fix.arg</code>	An optional named list giving values of fixed parameters of the named distribution. Parameters with fixed value are not estimated by maximum likelihood procedures.
<code>start.arg</code>	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.
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	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.
<code>inherit.aes</code>	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. <code>borders()</code> .

## 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	<i>Nested strips</i>
--------------	----------------------

---

### 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

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

### 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 `gg4x`.

### See Also

Other strips: [strip\\_split\(\)](#), [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 <a href="#">strip_nested</a> for more info. Note that currently, <code>strip_split()</code> cannot recognise collisions between strips, so changing to <code>bleed = TRUE</code> can have unexpected results.
text_x, text_y	A list() with <a href="#">element_text()</a> elements. See the details section in <a href="#">strip_themed()</a> .
background_x, background_y	A list() with <a href="#">element_rect()</a> elements. See the details section in <a href="#">strip_themed()</a> .
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 <code>rep_len()</code> .

**Details**

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

**Value**

A `StripSplit` ggproto object that can be given as an argument to facets in `gg4x`.

**See Also**

Other strips: [strip\\_nested\(\)](#), [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\_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 <code>element_text()</code> elements. See details.
background_x, background_y	A list() with <code>element_rect()</code> 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 <code>rep_len()</code> .

## 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_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 they 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\\_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

- \* **axis-guides**
  - guide\_axis\_logticks, 46
  - guide\_axis\_manual, 48
  - guide\_axis\_minor, 50
  - guide\_axis\_nested, 52
  - guide\_axis\_scalebar, 55
  - guide\_axis\_truncated, 56
- \* **datasets**
  - ggh4x\_extensions, 44
- \* **facetting functions**
  - facet\_grid2, 11
  - facet\_manual, 14
  - facet\_nested, 16
  - facet\_nested\_wrap, 19
  - facet\_wrap2, 22
- \* **strips**
  - strip\_nested, 95
  - strip\_split, 96
  - strip\_themed, 98
  - strip\_vanilla, 100
- aes(), 26, 29, 31, 34, 37, 41, 80, 83, 86, 89, 92
- alpha, 30, 43, 82
- at\_panel, 3
- borders(), 27, 30, 33, 36, 39, 43, 81, 84, 87, 90, 93
- bw.nrd(), 90
- center\_limits, 4
- colour, 30, 43, 82
- coord\_axes\_inside, 5
- coord\_cartesian(), 5
- coord\_fixed(), 5
- CoordAxesInside (ggh4x\_extensions), 44
- discrete scales, 76
- distribute\_args, 6
- distributions(), 93
- elem\_list\_rect (distribute\_args), 6
- elem\_list\_rect(), 99
- elem\_list\_text (distribute\_args), 6
- elem\_list\_text(), 99
- element\_line(), 54, 101
- element\_part\_rect, 8
- element\_rect(), 6, 7, 95, 97, 99
- element\_text(), 6, 7, 46, 49, 51, 53, 54, 58, 95, 97, 99, 101
- expansion(), 71, 77
- facet\_grid2, 11, 15, 19, 22, 24
- facet\_grid2(), 19
- facet\_manual, 13, 14, 19, 22, 24
- facet\_nested, 13, 15, 16, 22, 24
- facet\_nested(), 95, 101
- facet\_nested\_wrap, 13, 15, 19, 19, 24
- facet\_nested\_wrap(), 95, 101
- facet\_wrap2, 13, 15, 19, 22, 22
- facet\_wrap2(), 21
- FacetGrid2 (ggh4x\_extensions), 44
- FacetManual (ggh4x\_extensions), 44
- FacetNested (ggh4x\_extensions), 44
- FacetNestedWrap (ggh4x\_extensions), 44
- facetted\_pos\_scales, 9
- facetted\_pos\_scales(), 72, 73
- FacetWrap2 (ggh4x\_extensions), 44
- fill, 30, 82
- fitdistrplus::fitdist(), 94
- force\_panel\_sizes, 24
- fortify(), 26, 29, 31, 34, 38, 41, 80, 83, 86, 89, 92
- geom\_box, 26
- geom\_outline\_point, 28
- geom\_pointpath, 31
- geom\_polygonraster, 34
- geom\_raster(), 36
- geom\_rect(), 39, 40, 65
- geom\_rectmargin, 37
- geom\_text\_aimed, 41

- geom\_tilemargin (geom\_rectmargin), 37
- GeomBox (ggh4x\_extensions), 44
- GeomPointPath (ggh4x\_extensions), 44
- GeomPointpath (ggh4x\_extensions), 44
- GeomPolygonRaster (ggh4x\_extensions), 44
- GeomRectMargin (ggh4x\_extensions), 44
- GeomTextAimed (ggh4x\_extensions), 44
- GeomTileMargin (ggh4x\_extensions), 44
- ggdendro::dendro\_data(), 60
- ggh4x.axis.nestline.x/y, 54
- ggh4x.axis.nesttext.x/y, 54
- ggh4x.axis.ticks.length.mini, 47
- ggh4x.axis.ticks.length.minor, 47, 52
- ggh4x.facet.nestline, 18, 21
- ggh4x\_extensions, 44
- ggh4x\_theme\_elements
  - (theme\_extensions), 101
- ggplot(), 26, 29, 31, 34, 38, 41, 80, 83, 86, 89, 92
- ggplot2::annotation\_logticks(), 46
- ggplot2::continuous\_scale(), 74
- ggplot2::coord\_polar(), 36
- ggplot2::discrete\_scale, 70, 77
- ggplot2::element\_line(), 18, 21
- ggplot2::facet\_grid, 11
- ggplot2::facet\_grid(), 9, 19, 25
- ggplot2::facet\_null(), 25
- ggplot2::facet\_wrap, 12, 18
- ggplot2::facet\_wrap(), 9, 22, 25
- ggplot2::geom\_density(), 94
- ggplot2::geom\_histogram(), 94
- ggplot2::geom\_rug(), 40
- ggplot2::geom\_tile(), 40, 65
- ggplot2::guide\_axis(), 54
- ggplot2::guides(), 75
- ggplot2::layer(), 45
- ggplot2::scale\_x\_continuous(), 10
- ggplot2::sec\_axis, 63
- ggproto, 44
- ggsubset, 45
- grid::arrow(), 33
- grid::unit, 27
- grid::unit(), 19, 22, 25, 39
- group, 30, 44, 82
- guide\_axis(), 64
- guide\_axis\_color
  - (guide\_axis\_truncated), 56
- guide\_axis\_colour
  - (guide\_axis\_truncated), 56
- guide\_axis\_logticks, 46, 50, 52, 54, 56, 58
- guide\_axis\_logticks(), 101, 102
- guide\_axis\_manual, 47, 48, 52, 54, 56, 58
- guide\_axis\_minor, 47, 50, 50, 54, 56, 58
- guide\_axis\_minor(), 101
- guide\_axis\_nested, 47, 50, 52, 52, 56, 58
- guide\_axis\_nested(), 101
- guide\_axis\_scalebar, 47, 50, 52, 54, 55, 58
- guide\_axis\_truncated, 47, 50, 52, 54, 56, 56
- guide\_dendro, 59
- guide\_stringlegend, 61
- guides(), 71, 78
- help\_secondary, 63
- interaction(), 102
- key glyphs, 27, 30, 32, 35, 39, 42, 81, 84, 87, 90, 93
- label\_parsed(), 12, 15, 18, 21, 23
- label\_value(), 12, 15, 18, 21, 23
- labeller(), 12, 15, 18, 21, 23
- labs(), 46, 49, 51, 53, 55, 57, 59, 61
- lambda, 49, 70, 71, 77
- layer geom, 84
- layer position, 27, 29, 32, 35, 38, 42, 80, 84, 86, 89, 92
- layer stat, 27, 29, 32, 35, 38, 42
- layer(), 3, 27, 29, 30, 32, 35, 38, 39, 42, 80, 81, 84, 86, 87, 89, 90, 92, 93
- layout(), 14
- linetype, 82
- linewidth, 82
- position\_disjoint\_ranges, 65
- position\_lineartrans, 66
- position\_lineartrans(), 36
- PositionDisjointRanges
  - (ggh4x\_extensions), 44
- PositionLinearTrans (ggh4x\_extensions), 44
- rel(), 47, 52, 101, 102
- rep\_len(), 49
- scale\_colour\_gradientn(), 74
- scale\_colour\_multi (scale\_fill\_multi), 74

scale\_dendrogram, 70  
 scale\_facet, 72  
 scale\_fill\_multi, 74  
 scale\_listed, 75  
 scale\_x\_dendrogram (scale\_dendrogram),  
     70  
 scale\_x\_facet (scale\_facet), 72  
 scale\_x\_manual, 76  
 scale\_y\_dendrogram (scale\_dendrogram),  
     70  
 scale\_y\_facet (scale\_facet), 72  
 scale\_y\_manual (scale\_x\_manual), 76  
 ScaleDendrogram (ggh4x\_extensions), 44  
 scales::pal\_hue(), 70  
 sep\_discrete, 79  
 sep\_discrete(), 78  
 shape, 30  
 size, 30, 44  
 stat\_centroid (stat\_funxy), 82  
 stat\_difference, 79  
 stat\_funxy, 82  
 stat\_midpoint (stat\_funxy), 82  
 stat\_rle, 85  
 stat\_rollingkernel, 88  
 stat\_theodensity, 91  
 StatDifference (ggh4x\_extensions), 44  
 StatFunxy (ggh4x\_extensions), 44  
 StatRle (ggh4x\_extensions), 44  
 StatRollingkernel (ggh4x\_extensions), 44  
 stats::Distributions(), 94  
 stats::hclust(), 71  
 StatTheoDensity (ggh4x\_extensions), 44  
 Strip (ggh4x\_extensions), 44  
 strip\_nested, 95, 97, 99, 100  
 strip\_nested(), 19, 21, 22  
 strip\_split, 96, 96, 99, 100  
 strip\_themed, 96, 97, 98, 100  
 strip\_themed(), 95, 97  
 strip\_vanilla, 13, 15, 24, 96, 97, 99, 100  
 StripNested (ggh4x\_extensions), 44  
 StripSplit (ggh4x\_extensions), 44  
 StripThemed (ggh4x\_extensions), 44  
 subset.data.frame(), 45  
  
 theme(), 46, 49, 51, 53, 58  
 theme\_extensions, 101  
  
 unit, 49  
  
 vars(), 11, 14, 17, 20, 23  
  
 waiver(), 46, 49, 51, 53, 55, 57, 59, 61  
 weave\_factors, 102  
 weave\_factors(), 54  
  
 x, 30, 43, 81  
 xend, 44  
 xmax, 81  
 xmin, 81  
  
 y, 30, 43, 81  
 yend, 44  
 ymax, 81  
 ymin, 81