Package 'legendry'

May 13, 2026

Title Extended Legends and Axes for 'ggplot2'

Version 0.3.0

Description A 'ggplot2' extension that focusses on expanding the plotter's arsenal of guides. Guides in 'ggplot2' include axes and legends. 'legendry' offers new axes and annotation options, as well as new legends and colour displays.

License MIT + file LICENSE

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

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

Depends ggplot2 (>= 4.0.0), R (>= 4.1.0)

Imports cli, grid (>= 4.1.0), gtable, lifecycle, rlang (>= 1.2.0), S7, scales (>= 1.1.1), vctrs (>= 0.6.0)

Suggests knitr, ragg, rmarkdown, svglite, testthat (>= 3.0.0), vdiffr, withr

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2026-05-13 05:10:25 UTC

Contents

bracket_options *Bracket options*

Description

These functions construct various sorts of brackets. They construct a matrix that can be supplied as the bracket argument in primitive_bracket().

Usage

```
bracket_line()

bracket_square()

bracket_chevron()

bracket_round(angle = 180, n = 100L)

bracket_sigmoid(curvature = 10, n = 100L)

bracket_atan(curvature = 5, n = 100L)

bracket_curvy(angle = 225, n = 100L)
```

Arguments

angle	A numeric(1): the angle in degrees for which a circle piece is drawn. For bracket_curvy(), an angle between 180 and 270.
n	An integer(1) number of points to use for the bracket.
curvature	A numeric(1) that controls the curliness of the bracket. More precisely, it is used to construct the sequence seq(-curvature, curvature, length.out = n) over which the logistic or arctangent functions is evaluated.

Details

When designing custom bracket shapes, the expectation is both columns are are a number between 0 and 1. The first column follows the direction of the guide whereas the second column is orthogonal to that direction.

Value

A <matrix[n, 2]> with coordinates for points on the brackets.

Functions

- `bracket_line()`: A simple line as bracket. Has `n = 2` points.
- `bracket_square()`: A square bracket. Has `n = 4` points.
- `bracket_chevron()`: A chevron (V-shape) that makes a bracket. Has `n = 3` points.
- `bracket_round()`: One circular arc that makes a bracket.
- `bracket_sigmoid()`: Two sigmoid curves stacked on top of one another to form a bracket.
- `bracket_atan()`: Two arctangent curves stacked on top of one another to form a bracket.
- `bracket_curvy()`: Four circular arcs that make a bracket.

Examples

```
plot(bracket_sigmoid(), type = 'l')
```

`cap_options` *Cap options*

Description

These functions construct various sorts of caps. They construct a matrix that can be supplied as the `shape` argument in `gizmo_barcap()`.

Usage

```
cap_triangle()

cap_round(n = 100L)

cap_arch(n = 100L)

cap_ogee(n = 100L)

cap_none()
```

Arguments

`n`	An `<integer[n]>` number of points to use for the cap.

Details

When designing custom cap shapes, the expectation is that the first point starts at the `(0, 0)` coordinate and the last point ends at the `(0, 1)` coordinate. The first column follows the orthogonal direction of the bar whereas the second column follows the direction of the bar.

Value

A `<matrix[n, 2]>` with coordinates for points on the brackets.

Functions

- `cap_triangle()`: An equilateral triangle with `n = 3` points.
- `cap_round()`: A semicircle.
- `cap_arch()`: Two circular arcs forming an equilateral Gothic arch.
- `cap_ogee()`: Four circular arcs forming an 'ogee' arch.
- `cap_none()`: No cap.

Examples

```
plot(cap_arch(), type = 'l')
```

`compose_crux` *Compose guides in a cross*

Description

[Experimental]

This guide composition has a central guide optionally surrounded by other guides on all four sides.

Usage

```
compose_crux(
  key = NULL,
  centre = "none",
  left = "none",
  right = "none",
  top = "none",
  bottom = "none",
  args = list(),
  complete = FALSE,
  theme = NULL,
  theme_defaults = list(),
  reverse = FALSE,
  order = 0L,
  title = waiver(),
  position = waiver(),
  available_aes = NULL
)
```

Arguments

`key`	A standard key specification. The key is shared among all guides that have `NULL` keys themselves. See more information in the linked topic.
`centre`, `left`, `right`, `top`, `bottom`	Guides to use in composition per position. Each guide can be specified as one of the following:

- A <Guide> class object.
- A <function> that returns a <Guide> class object.
- A <character> naming such a function, without the guide_ or primitive_ prefix.

args — A <list> of arguments to pass to guides that are given either as a function or as a string.

complete — A <logical[1]> whether to treat the composition as a complete guide. If TRUE, a title and margin are added to the result. If FALSE (default), no titles and margins are added.

theme — A <theme> object to style the guide individually of differently from the plot's theme settings. The theme arguments in the guide overrides, and is combined with, the plot's theme.

theme_defaults — A <list> of theme elements to override undeclared theme arguments.

reverse — A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.

order — A positive <integer[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 hashing indicative settings of a guide.

title — One of the following to indicate the title of the guide:

- A <character[1]> or <expression[1]> to set a custom title.
- NULL to not display any title.
- waiver() (default) to take the name of the scale object or the name specified in labs() as the title.

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

available_aes — A <character> vector listing the aesthetics for which this guide can be build.

Details

Styling options:

Below are the theme options that determine the styling of this guide.

Theme setting	Type	Description
legend.title	element_text()	The title of the legend.
legend.title.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.text.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.margin	margin()	Padding around the legend.
legend.background	element_rect()	Background of the legend.

There are no further styling options.

The context-agnostic alternative to using theme() is to use theme_guide():

```
  compose_crux(theme = theme_guide(
    title = element_text(),
    title.position = "top",
    text.position = "right",
    margin = margin(5),
    background = element_rect()
  ))
```

Value

A <ComposeCrux> guide object.

See Also

Other composition: compose_ontop(), compose_sandwich(), compose_stack(), guide-composition

Examples

```
# Roughly recreating a colour bar with extra text on top and bottom
crux <- compose_crux(
  centre = gizmo_barcap(), left = "axis_base",
  right = "axis_base",
  top = primitive_title("A lot"),
  bottom = primitive_title("A little")
)

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = cty)) +
  guides(colour = crux)
```

---

compose_ontop *Compose guides on top of one another*

Description

This guide can place other guides on top of one another.

Usage

```
compose_ontop(
  ...,
  args = list(),
  key = NULL,
  title = waiver(),
  angle = waiver(),
  theme = NULL,
  order = 0L,
  position = waiver(),
  available_aes = NULL
)
```

Arguments

...	Guides to stack in composition. Each guide can be specified as one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character[1]> naming such a function, without the guide_ or primitive_ prefix.
args	A <list> of arguments to pass to guides that are given either as a function or as a string.
key	A standard key specification. The key is shared among all guides that have NULL keys themselves. See more information in the linked topic.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
order	A positive <integer[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 hashing indicative settings of a guide.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
available_aes	A <character> giving aesthetics that must match the the guides.

Details

Styling options:

There are no styling options in theme() for this composition.

Value

A <ComposeOntop> composite guide object.

See Also

Other composition: compose_crux(), compose_sandwich(), compose_stack(), guide-composition

Examples

```
# Using the ontop composition to get two types of ticks with different
# lengths
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  guides(x = compose_ontop(
    guide_axis_base(
      key_manual(c(2, 4, 6)),
      theme = theme(
        axis.ticks = element_line(colour = "limegreen"),
        axis.ticks.length = unit(11, "pt")
      )
    ),
    guide_axis_base(
      key_manual(c(3, 5, 7)),
      theme = theme(
        axis.ticks = element_line(colour = "tomato"),
        axis.ticks.length = unit(5.5, "pt")
      )
    )
  ))
```

---

compose_sandwich *Compose guides as a sandwich*

Description

[Experimental]

This guide composition has a middle guide flanked by two parallel guides.

Usage

```
compose_sandwich(
  key = key_auto(),
  middle = gizmo_barcap(),
  text = "none",
  opposite = "none",
  args = list(),
  suppress_labels = "opposite",
  complete = TRUE,
  theme = NULL,
  theme_defaults = list(),
  reverse = FALSE,
  order = 0L,
  title = waiver(),
  position = waiver(),
  available_aes = NULL
)
```

Arguments

key	A standard key specification. The key is shared among all guides that have NULL keys themselves. See more information in the linked topic.
middle	Guide to use as the middle guide. Each guide can be specified as one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character> naming such a function, without the guide_ or primitive_ prefix.
text, opposite	Guides to use at the legend.text.position location and on the opposite side of the middle guide respectively. Guide specification is the same as in the middle argument.
args	A <list> of arguments to pass to guides that are given either as a function or as a string.
suppress_labels	A <character> vector giving any of "text" and "opposite" for the parallel guides. The guide(s) listed here will not draw labels if they support a label suppression mechanism.
complete	A <logical[1]> whether to treat the composition as a complete guide. If TRUE, a title and margin are added to the result. If FALSE (default), no titles and margins are added.
theme	A <theme> object to style the guide individually of differently from the plot's theme settings. The theme arguments in the guide overrides, and is combined with, the plot's theme.
theme_defaults	A <list> of theme elements to override undeclared theme arguments.
reverse	A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.
order	A positive <integer[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 hashing indicative settings of a guide.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
position	Where this guide should be drawn: one of "top", "bottom", "left", or "right".
available_aes	A <character> vector listing the aesthetics for which this guide can be build.

Details

The sandwich composition is effectively the same as a crux composition lacking two opposing arms.

Styling options:

Below are the theme options that determine the styling of this guide.

Theme setting	Type	Description
legend.title	element_text()	The title of the legend.
legend.title.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.text.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.margin	margin()	Padding around the legend.
legend.background	element_rect()	Background of the legend.

There are no further styling options.

The context-agnostic alternative to using theme() is to use theme_guide():

```
compose_sandwich(theme = theme_guide(
  title = element_text(),
  title.position = "top",
  text.position = "right",
  margin = margin(5),
  background = element_rect()
))
```

Value

A <ComposeSandwich> guide object.

See Also

Other composition: compose_crux(), compose_ontop(), compose_stack(), guide-composition

Examples

```
# A standard plot with a sandwich guide
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = cty)) +
  guides(colour = compose_sandwich(
    middle = "colourbar",
    text = "axis_base",
    opposite = primitive_bracket(key = key_range_manual(
      start = c(10, 20), end = c(25, 30), name = c("A", "B")
    ))
  ))
```

compose_stack *Compose guides as stack*

Description

This guide can stack other guides.

Usage

```
compose_stack(
  ...,
  args = list(),
  key = NULL,
  title = waiver(),
  side.titles = waiver(),
  angle = waiver(),
  theme = NULL,
  order = 0L,
  drop = NULL,
  position = waiver(),
  available_aes = NULL
)
```

Arguments

`...`	Guides to stack in composition. Each guide can be specified as one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character[1]> naming such a function, without the guide_ or primitive_ prefix.
`args`	A <list> of arguments to pass to guides that are given either as a function or as a string.
`key`	A standard key specification. The key is shared among all guides that have NULL keys themselves. See more information in the linked topic.
`title`	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
`side.titles`	A <character> giving labels for titles displayed on the side of the stack. Set to NULL to display no side titles. If waiver(), an attempt is made to extract the titles from the guides and use these as side titles.

angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
order	A positive <integer[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 hashing indicative settings of a guide.
drop	An <integer> giving the indices of guides that should be dropped when a facet requests no labels to be drawn at axes in between panels. The default, NULL, will drop every guide except the first.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
available_aes	A <character> giving aesthetics that must match the the guides.

Details

Styling options:

Below are the theme options that determine the styling of this guide.

Theme setting	Type	Description
legendry.axis.subtitle	element_text()	Display of the side.titles argument's labels.
legendry.axis.subtitle.position	<character[1]>	One of "top", "right", "bottom" or "left".
legendry.guide.spacing	unit()	Spacing between guides.

There are no further styling options.

The context-agnostic alternative to using theme() is to use theme_guide():

```
compose_stack(theme = theme_guide(
  subtitle = element_text(),
  subtitle.position = "left",
  guide.spacing = unit(5, "mm")
))
```

Value

A <ComposeStack> guide object.

See Also

Other composition: compose_crux(), compose_ontop(), compose_sandwich(), guide-composition

Examples

```
ggplot() +
  geom_function(fun = dnorm, xlim = c(-3, 3)) +
  guides(x = compose_stack(
    "axis", "axis",
    side.titles = c("first", "second")
  )) +
  # Add margin to make room for side titles
  theme(plot.margin = margin(5.5, 5.5, 5.5, 11))
```

`gizmo_barcap` *Guide gizmo: capped colour bar*

Description

This guide displays a colour bar with optional caps at either ends of the bar.

Usage

```
gizmo_barcap(
  key = "sequence",
  shape = "triangle",
  size = NULL,
  show = NA,
  alpha = NA,
  oob = "keep",
  theme = NULL,
  position = waiver(),
  direction = NULL
)
```

Arguments

key	A sequence key specification. Defaults to `key_sequence(n = 15)`. Changing the argument to `key_sequence()` is fine, but changing the key type is not advised.
shape	A cap specification by providing one of the following: • A cap `<function>`, such as `cap_triangle()`. • A `<character[1]>` naming a cap function without the 'cap_'-prefix, e.g. `"round"`. • A two column `<matrix[n, 2]>` giving coordinates for a cap, like those created by cap functions such as `cap_arch()`.
size	A `<unit>` setting the size of the cap. When `NULL` (default), cap size will be proportional to the `shape` coordinates and the `legend.key.size` theme setting.

show — A <logical> to control how caps are displayed at the ends of the bar. When TRUE, caps are always displayed. When FALSE, caps are never displayed. When NA (default), caps are displayed when the data range exceed the limits. When given as <logical[2]>, show[1] controls the display at the lower end and show[2] at the upper end.

alpha — A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.

oob — An out-of-bounds handling function that affects the cap colour. Can be one of the following:

- A <function> like oob_squish.
- A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

direction — A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

Details

Styling options:

Below are the theme options that determine the styling of this guide. Note that the width or height (depending on the direction argument) *includes* the cap.

Theme setting	Type	Description
legend.frame	element_rect()	Frame drawn around the bar and caps. The fill setting is ignored.
legend.key.width	unit()	Width of the bar
legend.key.height	unit()	Height of the bar

Please note that depending on the direction argument, the legend.key.width/legend.key.height setting are expanded 5-fold if originating from the global theme. To set these directly, you can use the local theme argument in the guide. These settings have shorthands in theme_guide():

```
gizmo_barcap(theme = theme_guide(
  frame = element_rect(),
  key.width = unit(5, "mm")
  key.height = unit(5, "cm")
))
```

Value

A <GizmoBarcap> object.

See Also

Other gizmos: gizmo_density(), gizmo_grob(), gizmo_histogram(), gizmo_stepcap()

Examples

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

# Just a bar
p + scale_colour_viridis_c(guide = gizmo_barcap())

# Caps show up when there is data outside the limits
p + scale_colour_viridis_c(
  limits = c(10, 30),
  guide  = gizmo_barcap()
)

# The scale's out-of-bounds handler determines cap colour
p + scale_colour_viridis_c(
  limits = c(10, 30), oob = scales::oob_squish,
  guide = gizmo_barcap()
)

# Customising display of the guide
p +
  scale_colour_viridis_c(
    oob = scales::oob_squish,
    guide = gizmo_barcap(
      shape = "arch", show = c(FALSE, TRUE),
      size = unit(2, "cm"),
      theme = theme(legend.key.height = unit(4, "cm"))
    )
  ) +
  theme(
    legend.frame = element_rect(colour = "black"),
    legend.key.width = unit(0.5, "cm")
  )
```

---

gizmo_density *Guide gizmo: kernel density estimate*

Description

This guide displays a kernel density estimate (KDE) of the aesthetic. If the aesthetic is `colour` or `fill`, the shape will reflect this.

Usage

```
gizmo_density(
  key = waiver(),
  density = NULL,
  density.args = list(),
  density.fun = stats::density,
  just = 0.5,
  oob = "keep",
  alpha = NA,
  theme = NULL,
  position = waiver(),
  direction = NULL
)
```

Arguments

key	A sequence key or binned key specification. Internally defaults to a sequence key when the scale is continuous and a binned key when the scale is binned.
density	One of the following: • NULL for using kernel density estimation on the data values (default). • a <numeric> vector to feed to the density.fun function. • A named <list> with x and y numeric elements of equal length, such as one returned by using the density() function. Please note that <list> input is expected in scale-transformed space, not original data space.
density.args	A <list> with additional arguments to the density.fun argument. Only applies when density is not provided as a <list>. already.
density.fun	A <function> to use for kernel density estimation when the density argument is not provided as a list already.
just	A <numeric[1]> between 0 and 1. Use 0 for bottom- or left-aligned densities, use 1 for top- or right-aligned densities and 0.5 for violin shapes.
oob	An out-of-bounds handling function that affects the cap colour. Can be one of the following: • A <function> like oob_squish. • A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".
alpha	A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
direction	A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

Details

Non-finite values such as NA and NaN are ignored while infinite values such as -Inf and Inf are squished to the limits.

Styling options:

Below are the theme options that determine the styling of this guide.

Theme setting	Type	Description
legend.frame	element_rect()	Outline drawn around the density itself. The fill setting is ignored.
legend.key	element_rect()	Background underneath the density area.
legend.key.width	unit()	Width of the density area.
legend.key.height	unit()	Height of the density area.

Please note that depending on the direction argument, the legend.key.width/legend.key.height setting are expanded 5-fold if originating from the global theme. To set these directly, you can use the local theme argument in the guide. These settings have shorthands in theme_guide():

```
gizmo_density(theme = theme_guide(
  frame = element_rect(),
  key = element_rect(),
  key.width = unit(5, "mm")
  key.height = unit(5, "cm")
))
```

Value

A <GizmoDensity> object.

See Also

Other gizmos: gizmo_barcap(), gizmo_grob(), gizmo_histogram(), gizmo_stepcap()

Examples

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

# Density from plot data
p + guides(colour = gizmo_density())

# Using bins instead of gradient
p + guides(colour = gizmo_density("bins"))

# Providing custom values to compute density of
p + guides(colour = gizmo_density(density = runif(1000, min = 5, max = 35)))

# Providing a precomputed density
```

```
p + guides(colour = gizmo_density(density = density(mpg$cty, adjust = 0.5)))

# Alternatively, parameters may be passed through density.args
p + guides(colour = gizmo_density(density.args = list(adjust = 0.5)))
```

---

gizmo_grob *Guide gizmo: custom grob*

Description

This guide displays a user-provided grob.

Usage

```
gizmo_grob(
  grob,
  width = grobWidth(grob),
  height = grobHeight(grob),
  hjust = 0.5,
  vjust = 0.5,
  position = waiver()
)
```

Arguments

grob	A <grob> to display.
width, height	A [<unit[1]>][grid::unit] setting the allocated width and height of the the grob respectively.
hjust, vjust	A <numeric[1]> between 0 and 1 setting the horizontal and vertical justification of the grob when used as a guide for the x and y aesthetics.
position	Where this guide should be drawn: one of "top", "bottom", "left", or "right".

Details

Styling options:

There are no theme() styling options for gizmo_grob().

Value

A <GizmoGrob> object.

See Also

Other gizmos: gizmo_barcap(), gizmo_density(), gizmo_histogram(), gizmo_stepcap()

Examples

```
circle <- grid::circleGrob()

# A standard plot with grob gizmos
ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point() +
  guides(
    x.sec = gizmo_grob(
      circle, hjust = 0.75,
      width = unit(2, "cm"), height = unit(2, "cm")
    ),
    colour = gizmo_grob(
      circle, width = unit(1, "cm"), height = unit(1, "cm")
    )
  )
```

`gizmo_histogram` *Guide gizmo: histogram*

Description

This guide displays a histogram of the aesthetic. If the aesthetic is `colour` or `fill`, the shape will reflect this.

Usage

```
gizmo_histogram(
  key = waiver(),
  hist = NULL,
  hist.args = list(),
  hist.fun = graphics::hist,
  just = 1,
  oob = oob_keep,
  metric = "counts",
  alpha = NA,
  theme = NULL,
  position = waiver(),
  direction = NULL
)
```

Arguments

`key`	A sequence key or binned key specification. Internally defaults to a sequence key when the scale is continuous and a binned key when the scale is binned.
`hist`	One of the following: • `NULL` for computing histograms on the data values (default). • an atomic `<vector>` to feed to the `hist.fun` function.

- A named <list> with breaks and counts numeric items, where the breaks item is exactly one element longer than the counts item. A typical way to construct such list is using the hist() function. Please note that <list> input is expected in scale-transformed space, not original data space.

hist.args — A <list> with additional arguments to the hist.fun argument. Only applies when hist is not provided as a <list> already. Please note that these arguments are only used for binning and counting: graphical arguments are ignored.

hist.fun — A <function> to use for computing histograms when the hist argument is not provided as a list already.

just — A <numeric[1]> between 0 and 1. Use 0 for bottom- or left-aligned histograms, use 1 for top- or right-aligned histograms and 0.5 for centred histograms.

oob — An out-of-bounds handling function that affects the cap colour. Can be one of the following:

- A <function> like oob_squish.
- A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".

metric — A <character[1]> either "counts" or "density" stating which field of the <histogram> class to display. The "density" metric might be more appropriate to display when the histogram breaks have non-constant intervals.

alpha — A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

direction — A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

Details

Non-finite values such as NA and NaN are ignored while infinite values such as -Inf and Inf are squished to the limits.

Styling options:

Below are the theme options that determine the styling of this guide.

Theme setting	Type	Description
legend.frame	element_rect()	Outline drawn around the histogram itself. The fill setting is ignored.
legend.key	element_rect()	Background underneath the histogram area.
legend.key.width	unit()	Width of the histogram area.
legend.key.height	unit()	Height of the histogram area.

Please note that depending on the direction argument, the legend.key.width/legend.key.height setting are expanded 5-fold if originating from the global theme. To set these directly, you can use the local theme argument in the guide. These settings have shorthands in theme_guide():

```
  gizmo_histogram(theme = theme_guide(
    frame = element_rect(),
    key = element_rect(),
    key.width = unit(5, "mm")
    key.height = unit(5, "cm")
  ))
```

Value

A <GizmoHistogram> object.

See Also

Other gizmos: gizmo_barcap(), gizmo_density(), gizmo_grob(), gizmo_stepcap()

Examples

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

# Histogram from plot data
p + guides(colour = gizmo_histogram())

# Using bins instead of gradient
p + guides(colour = gizmo_histogram("bins"))

# Providing custom values to compute histogram
p + guides(colour = gizmo_histogram(hist = runif(1000, min = 5, max = 35)))

# Providing precomputed histogram
p + guides(colour = gizmo_histogram(hist = hist(mpg$cty, breaks = 10)))

# Alternatively, parameters may be passed through hist.args
p + guides(colour = gizmo_histogram(hist.arg = list(breaks = 10)))
```

---

gizmo_stepcap *Guide gizmo: capped colour steps*

Description

This guide displays a binned variant of the colour bar with optional caps at either ends of the bar.

Usage

```
gizmo_stepcap(
  key = "bins",
  shape = "triangle",
```

```
  size = NULL,
  show = NA,
  alpha = NA,
  oob = "keep",
  theme = NULL,
  position = waiver(),
  direction = NULL
)
```

Arguments

key	A bins key specificiation. Defaults to `key_bins(even.steps = FALSE, show.limits = NULL)`. Changing the arguments to `key_bins()` is fine, but changing the key type is not advised.
shape	A cap specification by providing one of the following: • A cap `<function>`, such as `cap_triangle()`. • A `<character[1]>` naming a cap function without the 'cap_'-prefix, e.g. `"round"`. • A two column `<matrix[n, 2]>` giving coordinates for a cap, like those created by cap functions such as `cap_arch()`.
size	A `<unit>` setting the size of the cap. When `NULL` (default), cap size will be proportional to the `shape` coordinates and the `legend.key.size` theme setting.
show	A `<logical>` to control how caps are displayed at the ends of the bar. When `TRUE`, caps are always displayed. When `FALSE`, caps are never displayed. When `NA` (default), caps are displayed when the data range exceed the limits. When given as `<logical[2]>`, `show[1]` controls the display at the lower end and `show[2]` at the upper end.
alpha	A `<numeric[1]>` between 0 and 1 setting the colour transparency of the bar. Use `NA` to preserve the alpha encoded in the colour itself.
oob	An out-of-bounds handling function that affects the cap colour. Can be one of the following: • A `<function>` like `oob_squish`. • A `<character[1]>` naming such a function without the 'oob'-prefix, such as `"keep"`.
theme	A `<theme>` object to style the guide individually or differently from the plot's theme settings. The `theme` argument in the guide overrides and is combined with the plot's theme.
position	A `<character[1]>` giving the location of the guide. Can be one of `"top"`, `"bottom"`, `"left"` or `"right"`.
direction	A `<character[1]>` indicating the direction of the guide. Can be on of `"horizontal"` or `"vertical"`.

Details

Styling options:

Below are the theme options that determine the styling of this guide. Note that the width or height (depending on the `direction` argument) *includes* the cap.

Theme setting	Type	Description
`legend.frame`	`element_rect()`	Frame drawn around the bar and caps. The `fill` setting is ignored.
`legend.key.width`	`unit()`	Width of the bar
`legend.key.height`	`unit()`	Height of the bar

Please note that depending on the `direction` argument, the `legend.key.width`/`legend.key.height` setting are expanded 5-fold if originating from the global theme. To set these directly, you can use the local `theme` argument in the guide. These settings have shorthands in `theme_guide()`:

```
gizmo_stepcap(theme = theme_guide(
  frame = element_rect(),
  key.width = unit(5, "mm")
  key.height = unit(5, "cm")
))
```

Value

A `GizmoStepcap` object.

See Also

Other gizmos: `gizmo_barcap()`, `gizmo_density()`, `gizmo_grob()`, `gizmo_histogram()`

Examples

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

# Just some recangles
p + scale_colour_viridis_c(guide = gizmo_stepcap())

# Caps show up when there is data outside the limits
p + scale_colour_viridis_c(
  limits = c(10, 30),
  guide = gizmo_stepcap()
)

# The scale's out-of-bounds handler determines cap colour
p + scale_colour_viridis_c(
  limits = c(10, 30), oob = scales::oob_squish,
  guide = gizmo_stepcap()
)

# Customising the display of the guide
p +
  scale_colour_viridis_c(
```

```
    oob = scales::oob_squish,
    guide = gizmo_stepcap(
      shape = "round", show = c(FALSE, TRUE),
      size = unit(1, "cm"),
      theme = theme(legend.key.height = unit(4, "cm"))
    )
  ) +
  theme(
    legend.frame = element_rect(colour = "black"),
    legend.key.width = unit(0.5, "cm")
  )
```

`guide-composition` *Guide composition*

Description

Guide composition is a meta-guide orchestrating an ensemble of other guides. On their own, a 'composing' guide is not very useful as a visual reflection of a scale.

Usage

```
new_compose(
  guides,
  args = list(),
  ...,
  available_aes = c("any", "x", "y", "r", "theta"),
  call = caller_env(),
  super = Compose
)
```

Arguments

`guides`	A `<list>` of guides wherein each element is one of the following: • A `<Guide>` class object. • A `<function>` that returns a `<Guide>` class object. • A `<character[1]>` naming such a function, without the `guide_` or `primitive_` prefix.
`args`	A `<list>` of arguments to pass to guides that are given either as a function or as a string.
`...`	Additional parameters to pass on to `new_guide()`.
`available_aes`	A `<character>` giving aesthetics that must match the the guides.
`call`	A call to display in messages.
`super`	A `<Compose>` class object giving a meta-guide for composition.

Details

Styling options:

Below are the theme options that determine the styling of this guide.

Theme setting	Type	Description
`legendry.guide.spacing`	`unit()`	Spacing between guides.

There are no further styling options.

Value

A <Compose> (sub-)class guide that composes other guides.

See Also

Other composition: `compose_crux()`, `compose_ontop()`, `compose_sandwich()`, `compose_stack()`

Examples

```
# The `new_compose()` function is not intended to be used directly
my_composition <- new_compose(list("axis", "axis"), super = ComposeStack)

# Is the same as
my_composition <- compose_stack("axis", "axis")
```

`guide-gizmos` *Guide gizmos*

Description

Guide gizmos are a speciality guide components that are very specific to one or a few aesthetics to display.

Typically they can be composed with other guides or guide primitives to form a complete guide.

guide-primitives *Guide primitives*

Description

Guide primitives are the building blocks of more complex guides. On their own, they are not very useful as a visual reflection of a scale.

Their purpose is to be combined with one another to form a more complex, complete guides that *do* reflect a scale in some way.

Details

The guide primitives are simple, but flexible in that they are not tailored for one particular aesthetic. That way they can be reused and combined at will.

guide_axis_annotation *Annotation axis guide*

Description

[Experimental]

This axis guide acts as annotation: it draws labels at specified places. It also wraps an inner guide, making the behaviour look like one is 'adding' the annotation on top of the regular guide.

Usage

```
guide_axis_annotation(
  aesthetic,
  label = as.character(aesthetic),
  ...,
  key = NULL,
  arrow = NULL,
  inner = waiver(),
  title = waiver(),
  theme = NULL,
  order = 0L,
  position = waiver(),
  call = NULL
)

annotate_top(..., position = "top")

annotate_right(..., position = "right")

annotate_bottom(..., position = "bottom")

annotate_left(..., position = "left")
```

Arguments

aesthetic	A vector of values for the guide to represent.
label	A <character[n]> or list of expressions to use as labels.
...	Additional graphical properties parallel to aesthetic. Can be text_colour, size, face, hjust, vjust, angle or lineheight for the labels. Can be line_colour, linewidth and linetype for ticks. Setting colour will also set text_colour and line_colour. For annotate_top(), annotate_right(), annotate_bottom() and annotate_left(), arguments are passed on to guide_axis_annotation().
key	A standard key overriding the aesthetic, label and ... arguments.
arrow	A grid::arrow() specification. Can be NULL (default) to draw no arrow.
inner	A guide that supports the aesthetic to draw the annotation across. When waiver() (default), populates a guide_axis_base() except in the "top", "right" and "theta.sec" positions.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
theme	A <theme> object to style the *annotation* part of this guide differently from the plot's theme settings. The theme argument in this guide overrides and is combined with the plot's theme.
order	A positive <integer[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 hashing indicative settings of a guide.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
call	A call to display in messages.

Details

Under the hood, this guide is a hybrid composition guide. The theme() options that govern the styling are partially determined by its consituents. They are linked below so you can find their 'Styling options' sections.

Constituent	**Description**
compose_ontop()	Composes the annotation on top of the inner guide
guide_axis_base()	The default inner guide
primitive_ticks()	Display of the annotation ticks
primitive_labels()	Display of the annotation labels

Styling options *per annotation* can be set via ... as described above.

The context-agnostic alternative to using theme() is to use theme_guide():

```
# Note that `theme` is used *only* for the annotation
guide_axis_annotation(theme = theme_guide(
  text = element_text(),
  ticks = element_line(),
  ticks.length = unit(5, "mm")
))
```

Value

A <Guide> object

See Also

Other standalone guides: guide_axis_base(), guide_axis_dendro(), guide_axis_nested(), guide_axis_plot(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

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

# Typical use
p +
  annotate_top(5, face = "bold") +
  annotate_bottom(c(2.5, 4.5), c("Bottom annotation", "Second label"))

# If you want to combine them with secondary axes, you must
# set the `inner` argument manually.
p +
  scale_y_continuous(
    sec.axis = dup_axis(breaks = c(15, 25, 35))
  ) +
  annotate_right(30, inner = "axis")

# Use in theta axis
p + coord_radial() +
  guides(theta = guide_axis_annotation(4.5, "Theta annotation"))

# Specialised use as part of other guides
# Note that `guide_colbar` imposes white inward ticks
# We can overrule these impositions with a replacement theme
p + aes(colour = cty) +
  guides(colour = guide_colbar(
    second_guide = guide_axis_annotation(22, "This", theme = theme_gray())
  ))
```

---

guide_axis_base *Custom axis guide*

Description

This axis guide is a visual representation of position scales and can represent the x, y, theta and r aesthetics. It differs from guide_axis() in that it can accept custom keys and is can act as an axis for coord_radial() like guide_axis_theta().

Usage

```
guide_axis_base(
  key = NULL,
  title = waiver(),
  subtitle = NULL,
  theme = NULL,
  n.dodge = 1L,
  check.overlap = FALSE,
  angle = waiver(),
  cap = "none",
  bidi = FALSE,
  order = 0L,
  position = waiver()
)
```

Arguments

key	A standard key specification. Defaults to key_auto(). See more information in the linked topic and the 'Details' section.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
subtitle	Passed on to primitive_title(title). Follow the linked topic for more details.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
n.dodge	An positive <integer[1]> setting the number of layers text labels can occupy to avoid overlapping labels.
check.overlap	A <logical[1]> indicating whether to check for and omit overlapping text. If TRUE, first, last and middle labels are recursively prioritised in that order. If FALSE, all labels are drawn.
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
cap	A method to cap the axes. One of the following: • A <character[1]> with one of the following: – "none" to perform no capping. – "both" to cap the line at both ends at the most extreme breaks.

  - "upper" to cap the line at the upper extreme break.
  - "lower" to cap the line at the lower extreme break.
- A <logical>[1], where TRUE is equivalent to "both" and FALSE is equivalent to "none" in the options above.
- A sorted <numeric>[2n] with an even number of members. The lines will be drawn between every odd-even pair.
- A <function> that takes the scale's breaks as the first argument, the scale's limits as the second argument and returns a <numeric>[2n] as described above.

bidi — A <logical[1]>: whether ticks should be drawn bidirectionally (TRUE) or in a single direction (FALSE, default).

order — A positive <integer[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 hashing indicative settings of a guide.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Under the hood, this guide is a stack composition of a line, ticks and labels primitives.

To set minor ticks, use key = "minor", or use the type argument in key_manual() or key_map().

To use this as a logarithmic axis, set key = "log".

Styling options:

Because this guide is pure composite guide, the theme options that govern the styling are determined by its constituents. They are linked below so you can find their 'Styling options' sections.

Primitive	Description
compose_stack	Stacks the lines, tick marks and labels.
primitive_line()	Makes up the axis line.
primitive_ticks()	Makes up the tick marks.
primitive_labels()	Makes up the labels.

Styling options *per break* can be set in the standard key. These override theme settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
guide_axis_base(theme = theme_guide(
  # Common options
  line = element_line(),
  text = element_text(),
  ticks = element_line(),
  ticks.length = unit(5, "mm"),

  # Niche options below
  minor.ticks = element_line(),
```

```
    minor.ticks.length = unit(5, "mm"),
    mini.ticks = element_line(),
    mini.ticks.length = unit(5, "mm"),
  ))
```

Value

A <Guide> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_dendro(), guide_axis_nested(), guide_axis_plot(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

```
# A standard plot with custom keys
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  scale_x_continuous(
    guide = guide_axis_base(key = key_minor())
  ) +
  scale_y_continuous(
    guide = guide_axis_base(key = key_manual(c(20, 25, 30, 40)))
  )
p

# Is translated to theta axis without fuss
p + coord_radial()

# To use as logarithmic axis:
ggplot(msleep, aes(bodywt, brainwt)) +
  geom_point(na.rm = TRUE) +
  scale_x_continuous(
    transform = "log10",
    guide = guide_axis_base("log")
  )
```

---

guide_axis_dendro *Dendrogram guide*

Description

This axis is a speciality axis for discrete data that has been hierarchically clustered. Please be aware that the guide cannot affect the scale limits, which should be set appropriately. This guide will give misleading results when this step is skipped!

Usage

```
guide_axis_dendro(
  key = "dendro",
  title = waiver(),
  theme = NULL,
  labels = TRUE,
  space = rel(10),
  vanish = TRUE,
  n.dodge = 1L,
  angle = waiver(),
  check.overlap = FALSE,
  ticks = "none",
  axis_line = "none",
  order = 0L,
  position = waiver()
)
```

Arguments

key	A segment key specification. See more information in the linked topic. Alternatively, an object of class <hclust> that automatically invokes key_dendro().
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
labels, ticks, axis_line	Guides to use as labels, ticks or axis lines. Can be specified as one of the following: • A <logical[1]> which when FALSE will set the guide to guide_none() and if TRUE, will set guide to appropriate primitive. • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character[1]> naming such a function, without the guide_ or primitive_ prefix.
space	Either a <unit> or <rel> object of length 1 determining the space allocated in the orthogonal direction. When the space argument is of class <rel> (default) the base size is taken from the tick length theme setting.
vanish	Only relevant when the guide is used in the secondary theta position: a <logical[1]> on whether the continue to draw the segments until they meed in the center (TRUE) or strictly observe the space setting (FALSE).
n.dodge	An positive <integer[1]> setting the number of layers text labels can occupy to avoid overlapping labels.

angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
check.overlap	A <logical[1]> indicating whether to check for and omit overlapping text. If TRUE, first, last and middle labels are recursively prioritised in that order. If FALSE, all labels are drawn.
order	A positive <integer[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 hashing indicative settings of a guide.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Styling options:

Because this guide is pure composite guide, the theme options that govern the styling are determined by its constituents. They are linked below so you can find their 'Styling options' sections.

Primitive	Description
compose_stack	Stacks the lines, tick marks and labels and dendrogram.
primitive_segments()	The dendrogram.
primitive_line()	Makes up the axis line.
primitive_ticks()	Makes up the tick marks.
primitive_labels()	Makes up the labels.

The context-agnostic alternative to using theme() is to use theme_guide():

```
guide_axis_dendro(theme = theme_guide(
  line = element_line(),
  text = element_text(),
  ticks = element_line(),
  ticks.length = unit(5, "mm"),
))
```

Value

A <Guide> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_nested(), guide_axis_plot(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

```
# Hierarchically cluster data
clust <- hclust(dist(scale(mtcars)), "ave")

# Using the guide along with appropriate limits
p <- ggplot(mtcars, aes(disp, rownames(mtcars))) +
  geom_col() +
  scale_y_discrete(limits = clust$labels[clust$order])

# Standard usage
p + guides(y = guide_axis_dendro(clust))

# Adding ticks and axis line
p +
  guides(y = guide_axis_dendro(clust, ticks = "ticks", axis_line = "line")) +
  theme(axis.line = element_line())

# Controlling space allocated to dendrogram
p + guides(y = guide_axis_dendro(clust, space = unit(4, "cm"))) +
  theme(axis.ticks.y.left = element_line("red"))

# If want just the dendrogram, use `labels = FALSE`
p + guides(y = guide_axis_dendro(clust, labels = FALSE), y.sec = "axis")
```

---

guide_axis_nested *Nested axis guide*

Description

This axis guide gives extra range annotations to position scales. It can be used to infer nesting structure from labels or annotate ranges.

Usage

```
guide_axis_nested(
  key = "range_auto",
  regular_key = "auto",
  type = "bracket",
  title = waiver(),
  subtitle = NULL,
  theme = NULL,
  angle = waiver(),
  cap = "none",
  bidi = FALSE,
  oob = "squish",
  drop_zero = TRUE,
  pad_discrete = NULL,
  levels_text = NULL,
```

```
  ...,
  order = 0L,
  position = waiver()
)
```

Arguments

key	One of the following: • A range key specification. If not key = "range_auto", additional labels will be inserted to represent point values. • A <character[1]> passed to the key_range_auto(sep) argument. An exception is made when the string is a valid key specification.
regular_key	A standard key specification for the appearance of regular tick marks.
type	Appearance of ranges. One of the following: • "box" to put text in boxes. • "bracket" (default) to text over brackets. • "fence" to put text near fences.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
subtitle	Passed on to primitive_title(title). Follow the linked topic for more details.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
cap	A method to cap the axes. One of the following: • A <character[1]> with one of the following: – "none" to perform no capping. – "both" to cap the line at both ends at the most extreme breaks. – "upper" to cap the line at the upper extreme break. – "lower" to cap the line at the lower extreme break. • A <logical>[1], where TRUE is equivalent to "both" and FALSE is equivalent to "none" in the options above. • A sorted <numeric>[2n] with an even number of members. The lines will be drawn between every odd-even pair.

- A <function> that takes the scale's breaks as the first argument, the scale's limits as the second argument and returns a <numeric>[2n] as described above.

bidi	A <logical[1]>: whether ticks should be drawn bidirectionally (TRUE) or in a single direction (FALSE, default).
oob	A method for dealing with out-of-bounds (oob) ranges. Can be one of "squish", "censor" or "none".
drop_zero	A <logical[1]> whether to drop near-zero width ranges (TRUE, default) or preserve them (FALSE).
pad_discrete	A <numeric[1]> giving the amount ranges should be extended when given as a discrete variable. This is applied after the drop_zero setting.
levels_text	A list of <element_text> objects to customise how text appears at every level.
...	Arguments passed on to primitive_bracket(), primitive_box() or primitive_fence() (depending on the type argument).
order	A positive <integer[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 hashing indicative settings of a guide.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

To offer other keys the opportunity to display ranges alongside regular-looking labels, the regular_key argument can be used to setup a separate key for display in between the ticks and ranges.

By default, the key = "range_auto" will incorporate the 0th level labels inferred from the scale's labels. These labels will look like regular labels.

Styling options:

Because this guide is pure composite guide, the theme options that govern the styling are determined by its constituents. They are linked below so you can find their 'Styling options' sections.

Primitive	Context	Description
compose_stack	Always	Stacks the other primitives.
primitive_line()	Always	Makes up the axis line.
primitive_ticks()	Always	Makes up the tick marks.
primitive_bracket()	type = "bracket"	Range display as brackets.
primitive_box()	type = "box"	Range display as boxes.
primitive_fence()	type = "fence"	Range display as fences.
primitive_title()	subtitle = <...>	Used for displaying subtitles.
primitive_labels()	key != "range_auto"	Used for displaying range-less, 0th level labels

Styling options *per range* can be set in the range key. These override theme settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
guide_axis_nested(theme = theme_guide(
  # Common options
  line = element_line(),
  text = element_text(),
  ticks = element_line(),
  ticks.length = unit(5, "mm"),

  # For brackets
  bracket = element_line(),
  bracket.size = unit(5, "mm"),

  # For boxes
  box = element_rect(),

  # For fences
  fence = element_line(),
  fence.post = element_line(),
  fence.rail = element_line(),

  # For subtitle (not main title)
  title = element_text()
))
```

Value

A <Guide> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_dendro(), guide_axis_plot(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

```
# A plot with nested categories on the x-axis
p <- ggplot(mpg, aes(interaction(drv, cyl), hwy)) +
  geom_boxplot()

p + guides(x = "axis_nested")

# Apply styling to brackets
p + guides(x = "axis_nested") +
  theme_guide(bracket = element_line("red", linewidth = 1))

# Don't drop nesting indicators that have 0-width
p + guides(x = guide_axis_nested(drop_zero = FALSE))

# Change additional padding for discrete categories
```

```
p + guides(x = guide_axis_nested(pad_discrete = 0))

# Change bracket type
p + guides(x = guide_axis_nested(bracket = "curvy"))

# Use boxes instead of brackets + styling of boxes
p + guides(x = guide_axis_nested(type = "box")) +
  theme_guide(box = element_rect("limegreen", "forestgreen"))

# Using fences instead of brackets + styling of fences
p + guides(x = guide_axis_nested(type = "fence", rail = "inner")) +
  theme_guide(
    fence.post = element_line("tomato"),
    fence.rail = element_line("dodgerblue")
  )

# Use as annotation of a typical axis
# `regular_key` controls display of typical axis
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  guides(x = guide_axis_nested(
    key = key_range_manual(
      start = 2:3,
      end = 5:6,
      name = c("First", "Second")
    ),
    regular_key = key_manual(c(2, 2.5, 3, 5, 7))
  ))
```

`guide_axis_plot` *Side-plot axis*

Description

[Experimental]

Displays an axis-sharing plot to the side of the panel.

Usage

```
guide_axis_plot(
  plot,
  title = NULL,
  size = unit(2, "cm"),
  reposition = TRUE,
  theme = theme_sub_legend(position = "none"),
  position = waiver()
)
```

Arguments

plot	A <ggplot> object, subject to limitations listed in the 'Details' section. Alternatively, a <function> that takes the scale as argument and returns a <ggplot> object.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL (default) to not display any title. • waiver() to take the name of the scale object or the name specified in labs() as the title. Please note that plot will still display a title unless instructed otherwise. To avoid duplicated titles, the default is to have no title for the guide.
size	An absolute <unit> to set the size of the plot panel in the orthogonal direction.
reposition	A <logical[1]>. If TRUE (default) the position argument of this guide will be propagated to the position scale bestowed upon the plot argument. If FALSE, that position scale will retain its original position field. Setting reposition = TRUE will generally tend to point axes outwards.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The order in which themes are applies is as follows: (1) the main plot's theme (2) the plot argument's theme and (3) this theme argument. The default theme argument suppresses legends.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

This guide is subject to the following limitations:

- The x- or y-scale of the main plot override the corresponding scale in the plot argument. This ensures that the scales line up. The plot argument should not have the relevant scale.
- The plot argument cannot have custom facets. It must use the default facet_null().
- This guide cannot be used in non-linear coordinate systems of the main plot and does not support non-linear coordinate systems in the plot argument.
- The theme(panel.widths, panel.heights) setting in the plot argument will be ignored in favour of the size argument.
- There is no mechanism to accommodate extra space needed by plot components outside the panel. This applies in the horizontal direction for x-axes and the vertical direction for y-axes. You may need to manually tweak the theme(plot.margin) setting of the main plot to accommodate these components.

Styling options:

This guide has no style options in theme() on itself. However, the plot herein is subject to styling as described under the theme argument.

Value

A <Guide> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_dendro(), guide_axis_nested(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

```
# A standard plot
main_plot <- ggplot(mpg, aes(displ, hwy, colour = drv)) +
  geom_point()

# Simple plot sharing the x-variable
x_plot <- ggplot(mpg, aes(displ, fill = drv)) +
  geom_density(alpha = 0.7)

# Simple plot sharing the y-variable
y_plot <- ggplot(mpg, aes(drv, hwy, colour = drv)) +
  geom_boxplot()

# Typical use
main_plot + guides(
  x = guide_axis_plot(x_plot),
  y = guide_axis_plot(y_plot)
)

main_plot + guides(
  # Include `fill` legend by overriding theme
  x = guide_axis_plot(x_plot, theme = NULL),
  # Change the size of the side-plot
  y = guide_axis_plot(y_plot, size = unit(4, "cm"))
)

# Components outside panels may need to be manually acommodated
main_plot +
  guides(y = guide_axis_plot(y_plot + labs(title = "Boxplot"))) +
  theme(plot.margin = margin(25, 5.5, 5.5, 5.5))

# Recursive use of this guide
main_plot + guides(x = guide_axis_plot(
  main_plot + guides(x = guide_axis_plot(x_plot))
))
```

---

guide_axis_symbols *Symbol and upset axis guide*

Description

[Experimental]

These axis guides can be used for set annotations of discrete categories. The upset guide displays set intersections in matrix and is can be used to replace Venn/Euler diagrams. The symbol guide also displays a matrix of symbols, but requires manually specifying them.

Usage

```
guide_axis_symbols(
  key = NULL,
  connect = NULL,
  title = waiver(),
  theme = NULL,
  override.aes = list(),
  position = waiver(),
  direction = NULL,
  call = NULL
)

guide_axis_upset(
  key = "upset",
  connect = "perpendicular",
  title = waiver(),
  theme = NULL,
  override.aes = list(),
  position = waiver(),
  direction = NULL,
  call = NULL
)
```

Arguments

`key` One of the following:

- An upset key specification. For `guide_axis_upset`, specifying a `<character[n]>` is also passed to the `key_upset(order)` argument. An exception is made when the string is a valid key specification.
- A symbol key specification.

`connect` One of the following:

- A `<data.frame>` containing the following columns:
  - `value_start`, `value_end` Scale break values or `<numeric[n]>` values connecting along the axis.
  - `level_start`, `level_end` must be `<integer[n]>` values connecting perpendicular to the axis.
  - (Optional) columns for graphical parameters: `colour`, `linewidth` and `linetype`.
- A `<character[1]>` keyword for upset guides or logical symbols:
  - `"perpendicular"`: connect `TRUE` symbols (set membership for upset) perpendicular to the axis.

  - "parallel": connect TRUE symbols parallel to the axis. This is *not* appropriate for upset guides.

title — One of the following to indicate the title of the guide:

- A <character[1]> or <expression[1]> to set a custom title.
- NULL to not display any title.
- waiver() (default) to take the name of the scale object or the name specified in labs() as the title.

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

override.aes — A named <list> specifying graphical properties of points to apply to symbols. Every element must either be length 1 or match the number of symbols determined by the key. 3 symbols are used in upset guides or for logical symbolism. Otherwise the number of unique values to the key_symbols(symbol) argument determines the number of symbols.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

direction — A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

call — A call to display in messages.

Details

The upset axis does not predetermine the order the categories. If any sorting based on set size needs to occur, the scale is the correct tool to handle this task.

Styling options:

Several styling options are provided in the theme, while individualised styling is discussed below the table.

Theme setting	Type	Description
legendry.axis.subtitle	element_text()	Titles on the side labelling levels.
legendry.axis.subtitle.position	<character[1]>	One of "top", "right", "bottom"
legendry.zebra.light	element_rect()	Row shading.
legendry.zebra.dark	element_rect()	Alternate row shading.
legendry.table.spacing	rel()/unit()	Padding between levels.
legendry.point	element_point()	Styling of the symbols
legendry.connector	element_line()	Drawing the connect lines.

Moreover, styling options *per group* of symbols can be set via the override.aes argument. These override theme settings.

Styling options *per symbol* can be set in key_symbols() via the ... argument. These override theme settings and 'per group' settings.

Styling options *per line* in line connectors can be set by including graphical parameters as columns in the connect = <data.frame> argument. These override theme settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
  guide_axis_symbols(theme = theme_guide(
    subtitle = element_line(),
    subtitle.position = "left",
    zebra.light = element_rect(),
    zebra.dark = element_rect(),
    table.spacing = unit(5, "mm"),
    point = element_point(),
    connector = element_line()
  ))
```

Value

A <Guide> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_dendro(), guide_axis_nested(), guide_axis_plot(), guide_circles(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

```
# Example plot
p <- ggplot(mpg, aes(paste(drv, year))) +
  geom_bar()

# A standard upset axis might not have right order of levels
p + guides(x = "axis_upset")

# The levels can be manually adjusted to taste
p + guides(x = guide_axis_upset(c("1999", "2008", "4", "f", "r")))

# The connections can be turned off to just show symbols
p + guides(x = guide_axis_upset(connect = NULL))

# The style can be changed per group of symbols.
# We need to give 3 colours to also cover NA-breaks
p + guides(x = guide_axis_upset(
  override.aes = list(colour = c("purple", "orange", NA))
))

# For symbol guides you have to manually specify where you want symbols and
# connection lines appear.
p + guides(x = guide_axis_symbols(
  key_symbols(
    aesthetic = c("4 1999", "4 2008", "r 1999"),
    level = c("Lvl 1", "Lvl 2", "Lvl 3")
  ),
  connect = data.frame(
    value_start = "4 2008", value_end = "r 1999",
    level_start = 2, level_end = 3
```

```
  )
))
```

`guide_circles` *Circle size guide*

Description

This guide displays the sizes of points as a series of circles. It is typically paired with `geom_point()` with `draw_key_point()` glyphs.

Usage

```
guide_circles(
  key = NULL,
  title = waiver(),
  theme = NULL,
  hjust = 0.5,
  vjust = 0,
  text_position = NULL,
  clip_text = FALSE,
  override.aes = list(shape = 1L),
  position = waiver(),
  direction = NULL
)
```

Arguments

`key`	A standard key specification. Defaults to `key_auto()`. See more information in the linked topic.
`title`	One of the following to indicate the title of the guide: • A `<character[1]>` or `<expression[1]>` to set a custom title. • `NULL` to not display any title. • `waiver()` (default) to take the name of the scale object or the name specified in `labs()` as the title.
`theme`	A `<theme>` object to style the guide individually or differently from the plot's theme settings. The `theme` argument in the guide overrides and is combined with the plot's theme.
`hjust, vjust`	A `<numeric[1]>` between 0 and 1 giving the horizontal and vertical justification, respectively, of the central shapes. It is recommended `hjust = 0.5` when text is placed on the left or right and `vjust = 0.5` is recommended when text is placed on top or in the bottom.
`text_position`	A string, one of `"ontop"`, `"top"`, `"right"`, `"bottom"`, or `"left"` do describe the placement of labels. The default (`NULL`), will take the `legend.text.position` theme setting.

clip_text	A <logical[1]> whether to give text in the "ontop" position a small rectangle of background colour.
override.aes	A named <list> specifying aesthetic parameters of the key glyphs. See details and examples in guide_legend().
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
direction	A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

Details

Please note that the default size scales scale to area, not radius, so equidistant breaks will appear at irregularly spaced positions due to labelling the diameter of a circle.

This graph was designed with standard round shapes in mind, i.e. shapes 1, 16, 19 and 21. For that reason, shape = 1 is the default override.aes argument. Other shapes will probably be drawn but the quality of their alignment and label placement may be unsatisfactory.

Styling options:

Below are the theme options that determine the styling of this guide.

Theme setting	Type	Description
legend.background	element_rect()	Background of the legend.
legend.margin	margin()	Padding around the legend.
legend.key	element_rect()	Background of the key area underneath circles.
legend.text	element_text()	Labels displayed next to tick marks or on top of circles.
legendry.legend.key.margin	margin()	Padding between circles and edge of key area.
legend.ticks	element_line()	Tick marks connecting circle to label.
legend.title	element_text()	Title of the legend.
legend.title.position	<character[1]>	One of "top", "right", "bottom" or "left".

Styling options *per break* can be set in the standard key. These override theme settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
guide_circles(theme = theme_guide(
  text = element_text(),
  title = element_text(),
  title.position = "top",
  margin = margin(5),
  key = element_rect(),
  key.margin = margin(5),
  background = element_rect(),
  ticks = element_line(),
))
```

Value

A <GuideCircles> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_dendro(), guide_axis_nested(), guide_axis_plot(), guide_axis_symbols(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

```
# A standard plot
p <- ggplot(mtcars, aes(disp, mpg)) +
  geom_point(aes(size = hp), alpha = 0.3)

# By default, the sizes aren't large enough to make this guide clear
p + scale_size_area(guide = "circles")

# Update with a more approrpriate scale
p <- p +
  scale_size_area(
    max_size = 30,
    limits = c(0, NA),
    breaks = c(0, 25, 100, 250)
  )
p + guides(size = "circles")

# Horizontal orientation
p + guides(size = guide_circles(
  vjust = 0.5, hjust = 0, text_position = "bottom"
))

# Alternative text placement
p + guides(size = guide_circles(
  text_position = "ontop",
  clip_text = TRUE
))

# More styling options
p + guides(size = guide_circles(override.aes = aes(colour = "red")))+
  theme(
    # Key background
    legend.key = element_rect(colour = "black", fill = 'white'),
    # Padding around central shapes
    legendry.legend.key.margin = margin(1, 1, 1, 1, "cm"),
    legend.ticks = element_line(colour = "blue"),
    legend.text.position = "left"
  )
```

---

guide_colbar *Custom colour bar guide*

Description

Similar to guide_colourbar(), this guide displays continuous colour or fill aesthetics. It has additional options to display caps at the end of the bar, depending on out-of-bounds values.

Usage

```
guide_colbar(
  title = waiver(),
  key = "auto",
  first_guide = "axis_base",
  second_guide = first_guide,
  shape = "triangle",
  size = NULL,
  show = NA,
  nbin = 15L,
  alpha = NA,
  reverse = FALSE,
  suppress_labels = "second",
  oob = scales::oob_keep,
  theme = NULL,
  vanilla = TRUE,
  position = waiver(),
  available_aes = c("colour", "fill")
)
```

Arguments

title — One of the following to indicate the title of the guide:

- A <character[1]> or <expression[1]> to set a custom title.
- NULL to not display any title.
- waiver() (default) to take the name of the scale object or the name specified in labs() as the title.

key — A sequence key specification. Defaults to key_sequence(n = 15). Changing the argument to key_sequence() is fine, but changing the key type is not advised.

first_guide, second_guide — Guides to flank the colour bar. Each guide can be specified using one of the following:

- A <Guide> class object.
- A <function> that returns a <Guide> class object.
- A <character> naming such a function, without the guide_ or primitive_ prefix.

The first_guide will be placed at the location specified by the legend.text.position theme setting. The second_guide will be placed opposite that position. When second_guide has a label suppression mechanism, no labels will be drawn for that guide.

shape — A cap specification by providing one of the following:

- A cap <function>, such as cap_triangle().
- A <character[1]> naming a cap function without the 'cap_'-prefix, e.g. "round".
- A two column <matrix[n, 2]> giving coordinates for a cap, like those created by cap functions such as cap_arch().

size — A <unit> setting the size of the cap. When NULL (default), cap size will be proportional to the shape coordinates and the legend.key.size theme setting.

show — A <logical> to control how caps are displayed at the ends of the bar. When TRUE, caps are always displayed. When FALSE, caps are never displayed. When NA (default), caps are displayed when the data range exceed the limits. When given as <logical[2]>, show[1] controls the display at the lower end and show[2] at the upper end.

nbin — A positive <integer[1]> determining how many colours to use for the colour gradient.

alpha — A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.

reverse — A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.

suppress_labels — A <character> vector giving any of "first" and "second" for the parallel guides. The guide(s) listed here will not draw labels if they support a label suppression mechanism.

oob — An out-of-bounds handling function that affects the cap colour. Can be one of the following:

- A <function> like oob_squish.
- A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

vanilla — A <logical[1]> whether to have the default style match the vanilla guide_colourbar() (TRUE) or take the theme verbatim (FALSE).

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

available_aes — A <character> vector listing the aesthetics for which this guide can be build.

Details

As colours are always rendered as gradients, it is important to use a graphics device that can render these. This can be checked by using check_device("gradients").

Styling options:

Because this guide is pure composite guide, the theme options that govern the styling are determined by its constituents. They are linked below so you can find their 'Styling options' sections. Note that `guide_axis_base()` is just a default that can be swapped out.

Constituent	Description
`compose_sandwich`	Combines the bar with two side-guides.
`gizmo_barcap()`	Makes up the colour bar.
`guide_axis_base()`	Makes up the tick marks and labels.

Styling options *per break* can be set in the standard key. These override theme settings.

The context-agnostic alternative to using `theme()` is to use `theme_guide()`:

```
guide_colbar(theme = theme_guide(
  # Composition settings
  title = element_text(),
  title.position = "top",
  text.position = "right",
  margin = margin(5),
  background = element_rect(),

  # Bar settings
  frame = element_rect(),
  key.width = unit(5, "mm")
  key.height = unit(5, "cm")

  # Common options for `guide_axis_base()`
  line = element_line(),
  text = element_text(),
  ticks = element_line(),
  ticks.length = unit(5, "mm"),
))
```

Value

A <Guide> object

See Also

Other standalone guides: `guide_axis_annotation()`, `guide_axis_base()`, `guide_axis_dendro()`, `guide_axis_nested()`, `guide_axis_plot()`, `guide_axis_symbols()`, `guide_circles()`, `guide_colring()`, `guide_colsteps()`, `guide_legend_base()`, `guide_legend_cross()`, `guide_legend_group()`, `guide_legend_manual()`

Examples

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

```
# The colourbar shows caps when values are out-of-bounds (oob)
p + scale_colour_viridis_c(
  limits = c(10, NA),
  guide = "colbar"
)

# It also shows how oob values are handled
p + scale_colour_viridis_c(
  limits = c(10, NA), oob = scales::oob_squish,
  guide = "colbar"
)

# Adjusting the type of cap
p + scale_colour_viridis_c(
  limits = c(10, 30), oob = scales::oob_squish,
  guide = guide_colbar(shape = "round")
)

# One-sided ticks
p + scale_colour_viridis_c(
  guide = guide_colbar(second_guide = "none")
)

# Colour bar with minor breaks
p + scale_colour_viridis_c(
  minor_breaks = scales::breaks_width(1),
  guide = guide_colbar(key = "minor")
)

# Using log ticks on a colourbar
ggplot(msleep, aes(sleep_total, sleep_rem)) +
  geom_point(aes(colour = bodywt), na.rm = TRUE) +
  scale_colour_viridis_c(
    transform = "log10",
    guide = guide_colbar(key = "log")
  )
```

`guide_colring` *Colour rings and arcs*

Description

Similar to `guide_colourbar()`, this guide displays continuous `colour` or `fill` aesthetics. Instead of a bar, the gradient in shown in a ring or arc, which can be convenient for cyclical palettes such as some provided in the **scico** package.

Usage

```
guide_colring(
  title = waiver(),
```

```
  key = "auto",
  start = 0,
  end = NULL,
  outer_guide = "axis_base",
  inner_guide = "axis_base",
  nbin = 300L,
  reverse = FALSE,
  show_labels = "outer",
  theme = NULL,
  vanilla = TRUE,
  position = waiver(),
  available_aes = c("colour", "fill"),
  ...
)
```

Arguments

`title`	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
`key`	A standard key specification. Defaults to key_auto().
`start, end`	A <numeric[1]> in radians specifying the offset of the starting and end points from 12 o'clock. The NULL default for end, internally defaults to start + 2 * pi.
`outer_guide, inner_guide`	Guides to display on the outside and inside of the colour ring. Each guide can be specified using one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character> naming such function, without the guide_ or primitive_ prefix.
`nbin`	A positive <integer[1]> determining how many colours to display.
`reverse`	A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.
`show_labels`	A <character[1]> indicating for which guide labels should be shown. Can be one of "outer" (default), "inner", "both" or "none". Note that labels can only be omitted if the related guide has a label suppression mechanism.
`theme`	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
`vanilla`	A <logical[1]> whether to have the default style match the vanilla guide_colourbar() (TRUE) or take the theme verbatim (FALSE).

position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
available_aes	A <character> vector listing the aesthetics for which this guide can be build.
...	Arguments forwarded to the outer_guide and inner_guide if provided as functions or strings.

Details

Styling options:

This guide is a hybrid composition guide, where the theme settings apply both this guide itself and the constituents it manages. The constituents are linked below so you can find their 'Styling options' sections. Note that guide_axis_base() is just a default that can be swapped out.

Constituent	Description
guide_axis_base()	Makes up the tick marks and labels at inner and outer rings.

In addition to the constituent's theme setting, it also has the following settings:

Theme setting	Type	Description
legend.background	element_rect()	Background of the legend.
legend.margin	margin()	Padding around the legend.
legend.key.width	unit()	Radial thickness of the donut ring.
legend.key.size	unit()	Size of the legend. Multiplied by 5 to roughly match dimensions of colour ba
legend.frame	element_rect()	Outline of the donut. The fill setting is ignored.
legend.title	element_text()	Title of the legend.
legend.title.position	<character[1]>	One of "top", "right", "bottom" or "left".

Styling options *per break* can be set in the standard key. These override theme settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
guide_colring(theme = theme_guide(
  # Ring settings
  title = element_text(),
  title.position = "top",
  margin = margin(5),
  background = element_rect(),
  frame = element_rect(),
  key.size = unit(1, "cm"),
  key.width = unit(5, "mm"),

  # Common options for `guide_axis_base()`
  line = element_line(),
  text = element_text(),
  ticks = element_line(),
  ticks.length = unit(5, "mm"),
))
```

Value

A <Guide> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_dendro(), guide_axis_nested(), guide_axis_plot(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

```
# Rings works best with a cyclical palette
my_pal <- c("black", "tomato", "white", "dodgerblue", "black")

p <- ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point() +
  scale_colour_gradientn(colours = my_pal)

# Standard colour ring
p + guides(colour = "colring")

# As an arc
p + guides(colour = guide_colring(
  start = 1.25 * pi, end = 2.75 * pi
))

# Removing the inner tick marks
p + guides(colour = guide_colring(inner_guide = "none"))

# Include labels on the inner axis
p + guides(colour = guide_colring(show_labels = "both"))

# Passing an argument to inner/outer guides
p + guides(colour = guide_colring(angle = 0))
```

---

guide_colsteps *Custom colour steps guide*

Description

Similar to guide_coloursteps(), this guide displays continuous colour or fill aesthetics. It has additional options to display caps at the end of the bar, depending on out-of-bounds values.

Usage

```
guide_colsteps(
  title = waiver(),
  key = "bins",
```

```
  first_guide = "axis_base",
  second_guide = "axis_base",
  shape = "triangle",
  size = NULL,
  show = NA,
  alpha = NA,
  reverse = FALSE,
  suppress_labels = "second",
  oob = scales::oob_keep,
  theme = NULL,
  position = waiver(),
  vanilla = TRUE,
  available_aes = c("colour", "fill")
)
```

Arguments

`title` One of the following to indicate the title of the guide:

- A <character[1]> or <expression[1]> to set a custom title.
- NULL to not display any title.
- waiver() (default) to take the name of the scale object or the name specified in labs() as the title.

`key` A bins key specificiation. Defaults to key_bins(even.steps = FALSE, show.limits = NULL). Changing the arguments to key_bins() is fine, but changing the key type is not advised.

`first_guide, second_guide`
Guides to flank the colour steps. Each guide can be specified using one of the following:

- A <Guide> class object.
- A <function> that returns a <Guide> class object.
- A <character> naming such a function, without the guide_ or primitive_ prefix.

The first_guide will be placed at the location specified by the legend.text.position theme setting. The second_guide will be placed opposite that position. When second_guide has a label suppression mechanism, no labels will be drawn for that guide.

`shape` A cap specification by providing one of the following:

- A cap <function>, such as cap_triangle().
- A <character[1]> naming a cap function without the 'cap_'-prefix, e.g. "round".
- A two column <matrix[n, 2]> giving coordinates for a cap, like those created by cap functions such as cap_arch().

`size` A <unit> setting the size of the cap. When NULL (default), cap size will be proportional to the shape coordinates and the legend.key.size theme setting.

show — A <logical> to control how caps are displayed at the ends of the bar. When TRUE, caps are always displayed. When FALSE, caps are never displayed. When NA (default), caps are displayed when the data range exceed the limits. When given as <logical[2]>, show[1] controls the display at the lower end and show[2] at the upper end.

alpha — A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.

reverse — A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.

suppress_labels — A <character> vector giving any of "text" and "opposite" for the parallel guides. The guide(s) listed here will not draw labels if they support a label suppression mechanism.

oob — An out-of-bounds handling function that affects the cap colour. Can be one of the following:

- A <function> like oob_squish.
- A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

vanilla — A <logical[1]> whether to have the default style match the vanilla guide_colourbar() (TRUE) or take the theme verbatim (FALSE).

available_aes — A <character> vector listing the aesthetics for which this guide can be build.

Details

As steps are rendered as clipped rectangles, it is important to use a graphics device that can render clipped paths. This can be checked by using check_device("clippingPaths").

Styling options:

Because this guide is pure composite guide, the theme options that govern the styling are determined by its constituents. They are linked below so you can find their 'Styling options' sections. Note that guide_axis_base() is just a default that can be swapped out.

Constituent	Description
compose_sandwich	Combines the bar with two side-guides.
gizmo_stepcap()	Makes up the colour bar.
guide_axis_base()	Makes up the tick marks and labels.

Styling options *per break* can be set in the standard key. These override theme settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
guide_colsteps(theme = theme_guide(
  # Composition settings
  title = element_text(),
  title.position = "top",
  text.position = "right",
  margin = margin(5),
  background = element_rect(),

  # Steps settings
  frame = element_rect(),
  key.width = unit(5, "mm")
  key.height = unit(5, "cm")

  # Common options for `guide_axis_base()`
  line = element_line(),
  text = element_text(),
  ticks = element_line(),
  ticks.length = unit(5, "mm"),
))
```

Value

A <Guide> object

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_dendro(), guide_axis_nested(), guide_axis_plot(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colring(), guide_legend_base(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

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

# The colour steps show caps when values are out-of-bounds
p + scale_colour_viridis_b(
  limits = c(10, NA),
  guide = "colsteps"
)

# It also shows how oob values are handled
p + scale_colour_viridis_b(
  limits = c(10, 30), oob = scales::oob_censor,
  guide = "colsteps"
)

# Adjusting the type of cap
p + scale_colour_viridis_b(
  limits = c(10, 30),
  guide = guide_colsteps(shape = "round")
```

```
)

# The default is to use the breaks as-is
p + scale_colour_viridis_b(
  limits = c(10, 30), breaks = c(10, 20, 25),
  guide = "colsteps"
)

# But the display can be set to use evenly spaced steps
p + scale_colour_viridis_b(
  limits = c(10, 30), breaks = c(10, 20, 25),
  guide = guide_colsteps(key = key_bins(even.steps = TRUE))
)

# Using tick marks by swapping side guides
p + scale_colour_viridis_b(
  guide = guide_colsteps(
    first_guide  = "axis_base",
    second_guide = "axis_base"
  )
)
```

`guide_legend_base` *Custom legend guide*

Description

This legend closely mirrors `ggplot2::guide_legend()`, but has two adjustments. First, `guide_legend_base()` supports a `design` argument for a more flexible layout. Secondly, the `legend.spacing.y` theme element is observed verbatim instead of overruled.

Usage

```
guide_legend_base(
  key = NULL,
  title = waiver(),
  theme = NULL,
  design = NULL,
  nrow = NULL,
  ncol = NULL,
  reverse = FALSE,
  override.aes = list(),
  position = NULL,
  direction = NULL,
  order = 0L
)
```

Arguments

key	A standard key specification. Defaults to key_auto(). See more information in the linked topic.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
design	Specification of the legend layout. One of the following: • NULL (default) to use the layout algorithm of guide_legend(). • A <character[1]> string representing a cell layout wherein # defines an empty cell. See examples. • A <matrix[n, m]> representing a cell layout wherein NA defines an empty cell. See examples. Non-string atomic vectors will be treated with as.matrix().
nrow, ncol	A positive <integer[1]> setting the desired dimensions of the legend layout. When NULL (default), the dimensions will be derived from the design argument or fit to match the number of keys.
reverse	A <logical[1]> whether the order of keys should be inverted.
override.aes	A named <list> specifying aesthetic parameters of the key glyphs. See details and examples in guide_legend().
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
direction	A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".
order	A positive <integer[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 hashing indicative settings of a guide.

Details

Styling options:

Below are the theme options that determine the styling of this guide. Note that these are the same for ggplot2::guide_legend().

Theme setting	Type	Description
legend.background	element_rect()	Background of the legend.
legend.margin	margin()	Padding around the legend.
legend.text	element_text()	Labels displayed next to keys.
legend.text.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.title	element_text()	Title of the legend.

legend.title.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.key	element_rect()	Background of the key areas.
legend.key.height	unit()	Height of keys.
legend.key.width	unit()	Width of keys.
legend.key.justification	<numeric[2]>	Justification for placing legend keys in excess space.
legend.key.spacing.x	unit()	Horizontal spacing between keys.
legend.key.spacing.y	unit()	Vertical spacing between keys. Taken literally.
legend.byrow	<logical[1]>	Row-order key filling (TRUE) or column-order (FALSE)

The context-agnostic alternative to using theme() is to use theme_guide():

```
guide_legend_base(theme = theme_guide(
  text = element_text(),
  text.position = "right",
  title = element_text(),
  title.position = "top",
  key = element_rect(),
  key.height = unit(5, "mm"),
  key.width = unit(5, "mm"),
  key.justification = c(0.5, 0.5),
  key.spacing.x = unit(5, "mm"),
  key.spacing.y = unit(5, "mm"),
  byrow = TRUE,
  margin = margin(5),
  background = element_rect(),
))
```

Value

A <GuideLegend> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_dendro(), guide_axis_nested(), guide_axis_plot(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Other legend guides: guide_legend_cross(), guide_legend_group(), guide_legend_manual()

Examples

```
# A dummy plot
p <- ggplot(data.frame(x = 1:3, type = c("tic", "tac", "toe"))) +
  aes(x, x, shape = type) +
  geom_point(na.rm = TRUE) +
  scale_shape_manual(values = c(1, 4, NA))

# A design string, each character giving a cell value.
# Newlines separate rows, white space is ignored.
design <- "
```

```
  123
  213
  321
"

# Alternatively, the same can be specified using a matrix directly
# design <- matrix(c(1, 2, 3, 2, 1, 3, 3, 2, 1), 3, 3, byrow = TRUE)

p + guides(shape = guide_legend_base(design = design))

# Empty cells can be created using `#`
design <- "
  #2#
  1#3
"

# Alternatively:
# design <- matrix(c(NA, 1, 2, NA, NA, 3), nrow = 2)

p + guides(shape = guide_legend_base(design = design))
```

`guide_legend_cross` *Cross legend guide*

Description

This is a legend type similar to `guide_legend()` that displays crosses, or: interactions, between two variables.

Usage

```
guide_legend_cross(
  key = NULL,
  title = waiver(),
  row_title = waiver(),
  col_title = waiver(),
  swap = FALSE,
  col_text = element_text(angle = 90, vjust = 0.5),
  subtitle_position = position_text(angle = c(0, -90, 0, 90), hjust = 0.5),
  override.aes = list(),
  reverse = FALSE,
  theme = NULL,
  position = NULL,
  direction = NULL,
  order = 0L
)
```

Arguments

key	One of the following key specifications: • A group split specification when using the legend to display a compound variable like `paste(var1, var2)`. • A standard key specification, like `key_auto()`, when crossing two separate variables across two scales.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • `waiver()` (default) to take the name of the scale object or the name specified in `labs()` as the title.
row_title, col_title	One of the following to indicate subtitles spanning the rows and columns of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • `waiver()` to propagate subtitles from merging guides (default).
swap	A <logical[1]> which when TRUE exchanges the column and row variables in the displayed legend.
col_text	An <element_text> object giving adjustments to text for the column labels. Can be NULL to display column labels in equal fashion to the row labels.
subtitle_position	A named list of 4 text elements, having the names "top", "right", "bottom" and "left". These govern the display of subtitles when placed in any of these positions relative to the keys. See `position_text()` for a convenient helper.
override.aes	A named <list> specifying aesthetic parameters of the key glyphs. See details and examples in `guide_legend()`.
reverse	A <logical[2]> whether the order of the keys should be inverted, where the first value controls the row order and second value the column order. Input as <logical[1]> will be recycled.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
direction	A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".
order	A positive <integer[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 hashing indicative settings of a guide.

Details

Styling options:

Below are the theme options that determine the styling of this guide. Note that these are almost the same for `ggplot2::guide_legend()`.

Theme setting	Type	Description
`legend.background`	`element_rect()`	Background of the legend.
`legend.margin`	`margin()`	Padding around the legend.
`legend.text`	`element_text()`	Labels displayed next to keys.
`legend.text.position`	`<character[1]>`	One of `"top"`, `"right"`, `"bottom"` or `"left"`.
`legend.title`	`element_text()`	Title of the legend.
`legend.title.position`	`<character[1]>`	One of `"top"`, `"right"`, `"bottom"` or `"left"`.
`legend.key`	`element_rect()`	Background of the key areas.
`legend.key.height`	`unit()`	Height of keys.
`legend.key.width`	`unit()`	Width of keys.
`legend.key.justification`	`<numeric[2]>`	Justification for placing legend keys in excess space.
`legend.key.spacing.x`	`unit()`	Horizontal spacing between keys.
`legend.key.spacing.y`	`unit()`	Vertical spacing between keys. Taken literally.

The context-agnostic alternative to using `theme()` is to use `theme_guide()`:

```
guide_legend_cross(theme = theme_guide(
  text = element_text(),
  text.position = "right",
  title = element_text(),
  title.position = "top",
  key = element_rect(),
  key.height = unit(5, "mm"),
  key.width = unit(5, "mm"),
  key.justification = c(0.5, 0.5),
  key.spacing.x = unit(5, "mm"),
  key.spacing.y = unit(5, "mm"),
  margin = margin(5),
  background = element_rect(),
))
```

Value

A `<GuideLegend>` object.

See Also

Other standalone guides: `guide_axis_annotation()`, `guide_axis_base()`, `guide_axis_dendro()`, `guide_axis_nested()`, `guide_axis_plot()`, `guide_axis_symbols()`, `guide_circles()`, `guide_colbar()`, `guide_colring()`, `guide_colsteps()`, `guide_legend_base()`, `guide_legend_group()`, `guide_legend_manual()`

Other legend guides: `guide_legend_base()`, `guide_legend_group()`, `guide_legend_manual()`

Examples

```
# Standard use for single aesthetic. The default is to split labels to
# disentangle aesthetics that are already crossed (by e.g. `paste()`)
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = paste(year, drv))) +
  guides(colour = "legend_cross")

# If legends should be merged between identical aesthetics, both need the
# same legend type.
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = paste(year, drv), shape = paste(year, drv))) +
  guides(colour = "legend_cross", shape = "legend_cross")

# Crossing two aesthetics requires a shared title and `key = "auto"`. The
# easy way to achieve this is to predefine a shared guide.
my_guide <- guide_legend_cross(key = "auto", title = "My title")

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = drv, shape = factor(year))) +
  guides(colour = my_guide, shape  = my_guide)

# You can cross more than 2 aesthetics but not more than 2 unique aesthetics.
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = drv, shape = factor(year), size = factor(drv))) +
  scale_size_ordinal() +
  guides(colour = my_guide, shape = my_guide, size = my_guide)

# You can merge an aesthetic that is already crossed with an aesthetic that
# contributes to only one side of the cross.
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = paste(year, drv), shape  = drv)) +
  guides(
    colour = guide_legend_cross(title = "My Title"),
    shape  = guide_legend_cross(title = "My Title", key = "auto")
  )
```

`guide_legend_group` *Grouped legend*

Description

This legend resembles `ggplot2::guide_legend()`, but has the ability to keep groups in blocks with their own titles.

Usage

```
guide_legend_group(
  key = "group_split",
  title = waiver(),
```

```
  override.aes = list(),
  nrow = NULL,
  ncol = NULL,
  theme = NULL,
  position = NULL,
  direction = NULL,
  order = 0L
)
```

Arguments

key	A group key specification. Defaults to key_group_split() to split labels to find groups.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
override.aes	A named <list> specifying aesthetic parameters of the key glyphs. See details and examples in guide_legend().
nrow, ncol	A positive <integer[1]> setting the desired dimensions of the legend layout. Either nrow or ncol can be set, but not both,
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
direction	A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".
order	A positive <integer[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 hashing indicative settings of a guide.

Details

Styling options:

Below are the theme options that determine the styling of this guide.

Theme setting	Type	Description
legendry.legend.subtitle	element_text()	Title of groups in the legend.
legendry.legend.subtitle.position	<character[1]>	One of "top", "right", "bottom" or "left".
legendry.group.spacing	unit()	Spacing in between groups of keys.
legend.background	element_rect()	Background of the legend.
legend.margin	margin()	Padding around the legend.
legend.text	element_text()	Labels displayed next to keys.

legend.text.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.title	element_text()	Title of the legend.
legend.title.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.key	element_rect()	Background of the key areas.
legend.key.height	unit()	Height of keys.
legend.key.width	unit()	Width of keys.
legend.key.justification	<numeric[2]>	Justification for placing legend keys in excess space.
legend.key.spacing.x	unit()	Horizontal spacing between keys.
legend.key.spacing.y	unit()	Vertical spacing between keys. Taken literally.
legend.byrow	<logical[1]>	Row-order key filling (TRUE) or column-order (FALSE)

The context-agnostic alternative to using theme() is to use theme_guide():

```
guide_legend_group(theme = theme_guide(
  subtitle = element_text(),
  subtitle.position = "top",
  group.spacing = unit(5, "mm"),
  text = element_text(),
  text.position = "right",
  title = element_text(),
  title.position = "top",
  key = element_rect(),
  key.height = unit(5, "mm"),
  key.width = unit(5, "mm"),
  key.justification = c(0.5, 0.5),
  key.spacing.x = unit(5, "mm"),
  key.spacing.y = unit(5, "mm"),
  byrow = TRUE,
  margin = margin(5),
  background = element_rect(),
))
```

Value

A <GuideLegend> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_dendro(), guide_axis_nested(), guide_axis_plot(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_manual()

Other legend guides: guide_legend_base(), guide_legend_cross(), guide_legend_manual()

Examples

```
# Standard plot for selection of `msleep`
df <- msleep[c(9, 28, 11, 5, 34, 54, 64, 24, 53), ]
```

```
p <- ggplot(df) +
  aes(bodywt, awake, colour = paste(order, name)) +
  geom_point()

# By default, groups are inferred from the name
p + guides(colour = "legend_group")

# You can also use a look-up table for groups
# The lookup table can be more expansive than just the data:
# We're using the full 'msleep' data here instead of the subset
lut <- key_group_lut(msleep$name, msleep$order)

p + aes(colour = name) +
  guides(colour = guide_legend_group(key = lut))

# `nrow` and `ncol` apply within groups
p + guides(colour = guide_legend_group(nrow = 1))

# Groups are arranged according to `direction`
p + guides(colour = guide_legend_group(ncol = 1, direction = "horizontal")) +
  theme(legend.title.position = "top")

# Customising the group titles
p + guides(colour = "legend_group") +
  theme(
    legendry.legend.subtitle.position = "left",
    legendry.legend.subtitle = element_text(
      hjust = 1, vjust = 1, size = rel(0.9),
      margin = margin(t = 5.5, r = 5.5)
    )
  )

# Changing the spacing between groups
p + guides(colour = "legend_group") +
  theme(legendry.group.spacing = unit(0, "cm"))
```

`guide_legend_manual` *Manual legend*

Description

[Experimental]

This is a guide that displays user-defined keys independent of scales. It should only be used as a last resort when struggling to format a conventional legend.

Usage

```
guide_legend_manual(
  labels,
```

```
  ...,
  layers = list(geom_point()),
  title = NULL,
  theme = NULL,
  design = NULL,
  nrow = NULL,
  ncol = NULL,
  reverse = FALSE,
  position = NULL,
  direction = NULL,
  order = 0L
)
```

Arguments

labels	Labels to display next to the keys. Can be a <character> or <expression> vector to set labels, or NULL to draw no labels.
...	Arguments interpreted as aesthetics. For example: colour = "red". The aesthetics must have the same size as the labels argument, or have size 1. These aesthetics may be overruled by fixed aesthetics set in the layers argument.
layers	A <list> of layers (<LayerInstance> objects) created by the geom_*() or stat_*() family of functions. These layers are used for their key_glyph drawing functions, as well as to populate default aesthetics. Any fixed aesthetics provided to these layers overrule aesthetics passed to the ... argument.
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
design	Specification of the legend layout. One of the following: • NULL (default) to use the layout algorithm of guide_legend(). • A <character[1]> string representing a cell layout wherein # defines an empty cell. See examples. • A <matrix[n, m]> representing a cell layout wherein NA defines an empty cell. See examples. Non-string atomic vectors will be treated with as.matrix().
nrow, ncol	A positive <integer[1]> setting the desired dimensions of the legend layout. When NULL (default), the dimensions will be derived from the design argument or fit to match the number of keys.
reverse	A <logical[1]> whether the order of keys should be inverted.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

direction A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

order A positive <integer[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 hashing indicative settings of a guide.

Details

Because this guide is not tied to a scale, it can be given an arbitrary name in guides(); as long as it doesn't clash with other aesthetics.

Styling options:

Below are the theme options that determine the styling of this guide.

Theme setting	Type	Description
legend.background	element_rect()	Background of the legend.
legend.margin	margin()	Padding around the legend.
legend.text	element_text()	Labels displayed next to keys.
legend.text.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.title	element_text()	Title of the legend.
legend.title.position	<character[1]>	One of "top", "right", "bottom" or "left".
legend.key	element_rect()	Background of the key areas.
legend.key.height	unit()	Height of keys.
legend.key.width	unit()	Width of keys.
legend.key.justification	<numeric[2]>	Justification for placing legend keys in excess space.
legend.key.spacing.x	unit()	Horizontal spacing between keys.
legend.key.spacing.y	unit()	Vertical spacing between keys. Taken literally.
legend.byrow	<logical[1]>	Row-order key filling (TRUE) or column-order (FALSE)

The context-agnostic alternative to using theme() is to use theme_guide():

```
guide_legend_manual(legend_args = list(theme = theme_guide(
  text = element_text(),
  text.position = "right",
  title = element_text(),
  title.position = "top",
  key = element_rect(),
  key.height = unit(5, "mm"),
  key.width = unit(5, "mm"),
  key.justification = c(0.5, 0.5),
  key.spacing.x = unit(5, "mm"),
  key.spacing.y = unit(5, "mm"),
  margin = margin(5),
  background = element_rect(),
)))
```

Value

A <GuideCustom> object.

See Also

Other standalone guides: guide_axis_annotation(), guide_axis_base(), guide_axis_dendro(), guide_axis_nested(), guide_axis_plot(), guide_axis_symbols(), guide_circles(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group()

Other legend guides: guide_legend_base(), guide_legend_cross(), guide_legend_group()

Examples

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

# Typical usage: set `label` and some aesthetics
p + guides(
  some_name = guide_legend_manual(
    label  = c("foo", "bar"),
    colour = c(NA, "black"),
    fill   = c("grey40", NA),
    layers = geom_col()
  )
)

# Alternative: use `layers` to set aesthetics
p + guides(
  some_name = guide_legend_manual(
    label = c("foo", "bar"),
    layers = geom_col(
      # Must match length of `label`
      colour = c(NA, "black"),
      fill = c("grey40", NA)
    )
  )
)

# You can use >1 layer
p + guides(
  some_name = guide_legend_manual(
    label = c("foo", "bar"),
    colour = c("tomato", "dodgerblue"),
    fill = NA,
    layers = list(geom_col(), geom_point())
  )
)
```

key_group *Group keys*

Description

These functions are helper functions for working with grouped data as keys in guides. They all share the goal of creating a guide key, but have different methods.

- `key_group_split()` is a function factory whose functions make an attempt to infer groups from the scale's labels.
- `key_group_lut()` is a function factory whose functions use a look up table to sort out group membership.

Usage

```
key_group_split(sep = "[^[:alnum:]]+", reverse = FALSE)

key_group_lut(members, group, ungrouped = "Other")
```

Arguments

`sep`	A `<character[1]>` giving a regular expression to use for splitting labels provided by the scale using the `strsplit()` function. By defaults, labels are splitted on any non-alphanumeric character.
`reverse`	A `<logical[1]>` which if `FALSE` (default) treats the first part of the split string as groups and later parts as members. If `TRUE`, treats the last part as groups.
`members`	A vector including the scale's `breaks` values.
`group`	A vector parallel to `members` giving the group of each member.
`ungrouped`	A `<character[1]>` giving a group label to assign to the scale's `breaks` that match no values in the `members` argument.

Value

A function to use as the `key` argument in a guide.

See Also

Other keys: `key_range`, `key_segments`, `key_specialty`, `key_standard`

Examples

```
# Example scale
values <- c("group A:value 1", "group A:value 2", "group B:value 1")
template <- scale_colour_hue(limits = values)

# Treat the 'group X' part as groups
key <- key_group_split(sep = ":")
```

```
key(template)

# Treat the 'value X' part as groups
key <- key_group_split(sep = ":", reverse = TRUE)
key(template)

# Example scale
template <- scale_colour_hue(limits = msleep$name[c(1, 7, 9, 23, 24)])

# A lookup table can have more entries than needed
key <- key_group_lut(msleep$name, msleep$order)
key(template)

# Or less entries than needed
key <- key_group_lut(
  msleep$name[23:24], msleep$order[23:24],
  ungrouped = "Other animals"
)
key(template)
```

---

key_range *Range keys*

Description

These functions are helper functions for working with ranged data as keys in guides. They all share the goal creating of a guide key, but have different methods:

- `key_range_auto()` is a function factory whose functions make an attempt to infer ranges from the scale's labels.
- `key_range_manual()` uses user-provided vectors to set ranges.
- `key_range_map()` makes mappings from a `<data.frame>` to set ranges.
- `key_range_rle()` uses run-length encoding to determine the start and end of runs (blocks of repeated data values).

Usage

```
key_range_auto(sep = "[^[:alnum:]]+", reverse = FALSE, ...)

key_range_manual(start, end, name = NULL, level = NULL, ..., call = NULL)

key_range_map(data, ..., call = NULL)

key_range_rle(x, ..., call = NULL)
```

Arguments

sep	A <character[1]> giving a regular expression to use for splitting labels provided by the scale using strsplit(). Defaults to splitting on any non-alphanumeric character.
reverse	A <logical[1]> which if FALSE (default) treats the first labels as the inner labels and the last labels as the outer labels. If TRUE, thee first labels are treated as the outer labels and the last labels are treated as the inner labels.
...	The ... parameter has two purposes. 1. In key_range_map() it is <data-masking>. A set of mappings similar to those provided to aes(), which will be evaluated in the data argument. These *must* contain start and end mappings. 2. In other keys, ... can be used to transfer graphical properties to the individual ranges of a guide. For example, using colour = "blue" will draw parts of the guides associated with ranges in blue. There is a shallow hierarchy in that text_colour, line_colour, rect_colour and point_colour are the specific properties for elements, but all inherit from the main colour setting. Likewise, size, linewidth, linetype and fill have specific variants for elements.
start, end	A vector that can be interpreted by the scale, giving the start and end positions of each range respectively.
name	A <character> or list of expressions
level	An <integer> giving the depth of each range to avoid overlaps between different ranges. When level is smaller than 1, no brackets are drawn.
call	A call to display in messages.
data	A <data.frame> or similar object coerced by fortify() to a <data.frame>, in which the mapping argument is evaluated.
x	A <vector[n]> for which to determine run-starts and run-ends.

Details

The level variable is optional and when missing, the guides use an algorithm similar to IRanges::disjointBins() to avoid overlaps.

The key_range_auto() does *not* work with expression labels.

Value

For key_range_auto() a function. For key_range_manual() and key_range_map() a <data.frame> with the <key_range> class.

See Also

Other keys: key_group, key_segments, key_specialty, key_standard

Examples

```
# Example scale
template <- scale_x_discrete(limits = c("A 1", "B 1", "C&1", "D&2", "E&2"))

# By default, splits on all non-alphanumeric characters
auto <- key_range_auto()
auto(template)

# Only split on a specific character
auto <- key_range_auto(sep = "&")
auto(template)

# Treating the letters as outer labels and numbers as inner labels
auto <- key_range_auto(reverse = TRUE)
auto(template)

# Providing custom values
key_range_manual(
  start = c(1, 5,  10),
  end   = c(4, 15, 11),
  level = c(0, 2, 1),
  name  = c("A", "B", "C")
)

# Values from a <data.frame>
key_range_map(presidential, start = start, end = end, name = name)

# Values from run length encoding
key_range_rle(c("AB", "AB", "C", "DEF", "DEF", "DEF"))
```

---

key_segments *Segment keys*

Description

These functions are helper functions for working with segment data as keys in guides. They all share the goal of creating a guide key, but have different methods:

- `key_segment_manual()` directly uses user-provided vectors to set segments.
- `key_segment_map()` makes mappings from a `<data.frame>` to set segments.
- `key_dendro()` is a specialty case for coercing dendrogram data to segments. Be aware that setting the key alone cannot affect the scale limits, and will give misleading results when used incorrectly!

Usage

```
key_segment_manual(value, oppo, value_end = value, oppo_end = oppo, ...)
```

```
key_segment_map(data, ..., .call = caller_env())

key_dendro(dendro = NULL, type = "rectangle", ..., .call = NULL)
```

Arguments

value, value_end	A vector that is interpreted to be along the scale that the guide codifies.
oppo, oppo_end	A vector that is interpreted to be orthogonal to the value and value_end variables.
...	The ... parameter has two purposes. 1. In key_segments_map() it is <data-masking>. A set of mappings similar to those provided to aes(), which will be evaluated in the data argument. These *must* contain value and oppo mappings. 2. In other keys, ... can be used to transfer graphical properties to the individual breaks of a guide. For example, using colour = "blue" will draw parts of the guides associated with breaks in blue. There is a shallow hierarchy in that line_colour is the specific property for segment elements, but others inherit from the main colour setting. Likewise, linewidth and linetype have specific variants for line elements.
data	A <data.frame> or similar object coerced by fortify() to a <data.frame>, in which the mapping argument is evaluated.
.call	A call to display in messages.
dendro	A data structure that can be coerced to a dendrogram through the as.dendrogram() function. When NULL (default) an attempt is made to search for such data in the scale.
type	A string, either "rectangle" or "triangle", indicating the shape of edges between nodes of the dendrogram.

Value

For key_segments_manual() and key_segments_map(), a <data.frame> with the <key_range> class.

See Also

Other keys: key_group, key_range, key_specialty, key_standard

Examples

```
# Giving vectors directly
key_segment_manual(
  value = 0:1, value_end = 2:3,
  oppo  = 1:0, oppo_end  = 3:2
)

# Taking columns of a data frame
data <- data.frame(x = 0:1, y = 1:0, xend = 2:3, yend = 3:2)
```

```
key_segment_map(data, value = x, oppo = y, value_end = xend, oppo_end = yend)

# Using dendrogram data
clust <- hclust(dist(USArrests), "ave")
key_dendro(clust)(scale_x_discrete())
```

`key_specialty` *Speciality keys*

Description

These functions are helper functions for working with keys in guides. The functions described here are not widely applicable and may only apply to a small subset of guides. As such, it is fine to adjust the arguments of a speciality key, but swapping types is ill-advised.

- `key_sequence()` is a function factory whose functions create a regularly spaced sequence between the limits of a scale. It is used in colour bar guides.
- `key_bins()` is a function factory whose function create a binned key given the breaks in the scale. It is used in colour steps guides.
- `key_upset()` is a function factory whose function creates an upset key from splitting the breaks in the scale. It is used in the upset guide.
- `key_symbols()` is a function factory whose function creates a key from the literal provided values. It is used in the symbols guide.

Usage

```
key_sequence(n = 15L)

key_bins(even.steps = FALSE, show.limits = NULL)

key_upset(sep = "[^[:alnum:]]+", order = NULL, empty_label = "Other")

key_symbols(aesthetic, level, symbol = NULL, ...)
```

Arguments

`n`	A positive `<integer[1]>` giving the number of colours to use for a gradient.
`even.steps`	A `<logical[1]>` indicating whether the size of bins should be displayed as equal (`TRUE`) or proportional to their length in data space (`FALSE`).
`show.limits`	A `<logical[1]>` stating whether the limits of the scale should be shown with labels and ticks (`TRUE`) or remain hidden (`FALSE`). Note that breaks coinciding with limits are shown regardless of this setting. The default, `NULL`, consults the scale's `show.limits` setting or defaults to `FALSE`.
`sep`	A `<character[1]>` giving a regular expression to use for splitting labels provided by the scale using the `strsplit()` function. By defaults, labels are splitted on any non-alphanumeric character.

order	Order to set the upset layers in. One of the following: • A <character[n]> giving pieces of split labels. • An <integer[n]> giving the numerical order in which pieces of split labels should appear.
empty_label	A <character[1]> giving a level label to assign to the breaks that match no values to the pieces of split labels. Can be NULL to omit labels for empty levels.
aesthetic	A vector of values for the guide to represent equivalent to the breaks argument in scales. These will be mapped by the scale to positions. Alternatively, a <numeric[n]> vector to set positions directly. Positions are used to place symbols.
level	A <factor[n]> or <character[n]> parallel to the aesthetic argument setting the label level of the symbol.
symbol	(Optional) An <integer[n]> indexing the guide's override.aes parameter.
...	Additional graphical properties to set for each symbol. Valid properties are colour, shape, size, fill and stroke. These graphical properties have priority over properties derived via symbol or the theme.

Value

A function.

See Also

Other keys: key_group, key_range, key_segments, key_standard

Examples

```
# An example scale
template <- scale_fill_viridis_c(limits = c(0, 10), breaks = c(2, 4, 6, 8))

# Retrieving colourbar and colourstep keys
key_sequence()(template)
key_bins()(template)

# Upset key with example scale
template <- scale_x_discrete(limits = c("A", "A,B", ""))
key_upset()(template)
# Putting 'B' in 1st level
key_upset(order = c("B", "A"))(template)
# Omit level for the empty break
key_upset(empty_label = NULL)(template)

# Symbol key with example scale
template <- scale_x_discrete(limits = LETTERS[1:5])
key_symbols(aesthetic = LETTERS[1:3], level = 3:1)(template)
# Aesthetic can also be numeric
key_symbols(1:3, 3:1)(template)
# Setting level order via factors
ordered <- factor(c("X", "Y", "Z"), c("Y", "X", "Z"))
```

```
key_symbols(1:3, level = ordered)(template)
# Setting groups for symbols, for `guide_axis_symbols(override.aes)`
key_symbols(1:3, 1:3, symbol = c(1, 1, 2))(template)
# Passing individual graphical parameters
key_symbols(1:3, 3:1, colour = c("red", "green", "blue"))(template)
```

`key_standard` *Standard keys*

Description

These functions are helper functions for working with tick marks as keys in guides. They all share the goal of creating a guide key, but have different outcomes:

- `key_auto()` is a function factory whose functions extract a typical key from major breaks in a scale.
- `key_manual()` uses user-provided vectors to make a key.
- `key_map()` makes mappings from a `<data.frame>` to make a key.
- `key_minor()` is a function factory whose functions also extract minor break positions for minor tick marks.
- `key_log()` is a function factory whose functions place ticks at intervals in log10 space.
- `key_none()` makes an empty key with no entries.

Usage

```
key_auto(..., call = NULL)

key_manual(
  aesthetic,
  value = aesthetic,
  label = as.character(value),
  type = NULL,
  ...,
  call = NULL
)

key_map(data, ..., call = NULL)

key_minor(..., call = NULL)

key_log(
  prescale_base = NULL,
  negative_small = 0.1,
  expanded = TRUE,
  labeller = NULL,
  ...,
```

```
  call = NULL
)

key_none()
```

Arguments

`...`	The `...` parameter has two purposes. 1. In `key_map()` it is `<data-masking>`. A set of mappings similar to those provided to `aes()`, which will be evaluated in the `data` argument. These must contain `aesthetic` mapping. 2. In other keys, `...` can be used to transfer graphical properties to the individual breaks of a guide. For example, using `colour = "blue"` will draw parts of the guides associated with breaks in blue. There is a shallow hierarchy in that `text_colour`, `line_colour`, `rect_colour` and `point_colour` are the specific properties for elements, but all inherit from the main `colour` setting. Likewise, `size`, `linewidth`, `linetype` and `fill` have specific variants for elements.
`call`	A call to display in messages.
`aesthetic, value`	A vector of values for the guide to represent equivalent to the `breaks` argument in scales. The `aesthetic` will be mapped, whereas `value` will not. For most intents and purposes, `aesthetic` and `value` should be identical.
`label`	A `<character>` or list of expressions to use as labels.
`type`	A `<character>` vector representing the one of the break types: `"major"`, `"minor"` or `"mini"`. If `NULL` (default), all breaks are treated as major breaks.
`data`	A `<data.frame>` or similar object coerced by `fortify()` to a `<data.frame>`, in which the `mapping` argument is evaluated.
`prescale_base`	A `<numeric[1]>` giving the base of logarithm to transform data manually. The default, `NULL`, will use the scale transformation to calculate positions. It is only advisable to set the `prescale_base` argument when the data have already been log-transformed. When using a log-transform in the scale or in `coord_transform()`, the default `NULL` is recommended.
`negative_small`	A `<numeric[1]>` setting the smallest absolute value that is marked with a tick in case the scale limits include 0 or negative numbers.
`expanded`	A `<logical[1]>` determining whether the ticks should cover the entire range after scale expansion (`TRUE`, default), or be restricted to the scale limits (`FALSE`).
`labeller`	A `<function>` that receives major breaks and returns formatted labels. For `key_log()`, `NULL` will default to `scales::label_log()` for strictly positive numbers and a custom labeller when negative numbers are included.

Value

For `key_auto()`, `key_minor()` and `key_log()` a function. For `key_manual()` and `key_map()` a `<data.frame>` with the `<key_standard>` class.

See Also

Other keys: key_group, key_range, key_segments, key_specialty

Examples

```
# An example scale
template <- scale_x_continuous(limits = c(0, 10))

# The auto, minor and log keys operate on scales
key_auto()(template)
key_minor()(template)

# So does the log key
template <- scale_x_continuous(transform = "log10", limits = c(0.1, 10))
key_log()(template)

# Providing custom values
key_manual(
  aesthetic = 1:5,
  label = c("one", "two", "three", "four", "five")
)

# Values from a `<data.frame>`
key_map(ToothGrowth, aesthetic = unique(supp))

# Empty key
key_none()
```

---

position_text *Helper to position text*

Description

This is a helper function for use in guide_legend_cross(). It creates a list of text elements, each corresponding the top, right, bottom or left positions. Input is given for that order as well.

Usage

```
position_text(angle = NA, hjust = NA, vjust = NA, ..., element = element_text)
```

Arguments

angle	A <numeric[1-4]> value setting the angle of the text.
hjust, vjust	A <numeric[1-4]> between 0 and 1 to set the horizontal justification (hjust) or vertical justification (vjust) for the text.
...	Other arguments passed to the element function.
element	An <function> creating an object that inherits from the <element_text> class.

Details

Input other than the `element` argument will be recycled to length 4. `NA` and 0-length input will be dropped.

Value

A named list of `element` objects of length 4. The list has the names `"top"`, `"right"`, `"bottom"` and `"left"`.

Examples

```
# Red text turning along with position
position_text(angle = c(0, -90, 180, 90), colour = "red")
```

---

primitive_box *Guide primitives: boxes*

Description

This function constructs a boxes guide primitive.

Usage

```
primitive_box(
  key = "range_auto",
  angle = waiver(),
  oob = "squish",
  drop_zero = TRUE,
  pad_discrete = 0.4,
  min_size = NULL,
  levels_box = NULL,
  levels_text = NULL,
  theme = NULL,
  position = waiver()
)
```

Arguments

`key`	A range key specification. See more information in the linked topic.
`angle`	A specification for the text angle. Compared to setting the `angle` argument in `element_text()`, this argument uses some heuristics to automatically pick the `hjust` and `vjust` that you probably want. Can be one of the following: • `NULL` to take angles and justification settings directly from the theme. • `waiver()` to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.

oob	A method for dealing with out-of-bounds (oob) ranges. Can be one of "squish", "censor" or "none".
drop_zero	A <logical[1]> whether to drop near-zero width ranges (TRUE, default) or preserve them (FALSE).
pad_discrete	A <numeric[1]> giving the amount ranges should be extended when given as a discrete variable. This is applied after the drop_zero setting.
min_size	A [<grid::unit[1]>][grid::unit] setting the minimal size of a box.
levels_box	A list of <element_rect> objects to customise how boxes appear at every level.
levels_text	A list of <element_text> objects to customise how text appears at every level.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Styling options:

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

The possible {position} suffixes mentioned below are x, x.top, x.bottom, y, y.left, y.right. The theta and r position suffixes in **ggplot2** are *not* obeyed in **legendry**.

Theme setting	Context	Type	Description
legendry.box	Both	element_rect()	The boxes themselves
axis.text.{position}*	Axis	element_text()	The text in the boxes.
legend.text	Legend	element_text()	The text in the boxes.

Styling options *per level* can be set in the levels_box and levels_text arguments. These override theme settings.

Styling options *per range* can be set in the range key. The rect and text prefixed properties are prioritised for the boxes and text respectively. These override theme settings and 'per level' settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
primitive_box(theme = theme_guide(
  box = element_rect(),
  text = element_text()
))
```

Value

A <PrimitiveBox> primitive guide that can be used inside other guides.

See Also

Other primitives: primitive_bracket(), primitive_fence(), primitive_labels(), primitive_line(), primitive_segments(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

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

key <- key_range_manual(c(2, 4), c(5, 6), c("A", "B"))

# Adding as secondary guides
p + guides(
  x.sec = primitive_box(),
  y.sec = primitive_box(key = key)
)
```

---

primitive_bracket *Guide primitive: brackets*

Description

This function constructs a brackets guide primitive.

Usage

```
primitive_bracket(
  key = "range_auto",
  bracket = "line",
  angle = waiver(),
  oob = "squish",
  drop_zero = TRUE,
  pad_discrete = 0.4,
  levels_brackets = NULL,
  levels_text = NULL,
  theme = NULL,
  position = waiver()
)
```

Arguments

key	A range key specification. See more information in the linked topic.
bracket	A bracket by providing one of the following: • A bracket <function>, such as bracket_square. • A <character[1]> naming a bracket function without the 'bracket_'-prefix, e.g. "square".

- A two-column <matrix[n, 2]> giving line coordinates for a bracket, like those created by bracket functions, such as bracket_round().

angle — A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following:

- NULL to take angles and justification settings directly from the theme.
- waiver() to allow reasonable defaults in special cases.
- A <numeric[1]> between -360 and 360 for the text angle in degrees.

oob — A method for dealing with out-of-bounds (oob) ranges. Can be one of "squish", "censor" or "none".

drop_zero — A <logical[1]> whether to drop near-zero width ranges (TRUE, default) or preserve them (FALSE).

pad_discrete — A <numeric[1]> giving the amount ranges should be extended when given as a discrete variable. This is applied after the drop_zero setting.

levels_brackets — A list of <element_line> objects to customise how brackets appear at every level.

levels_text — A list of <element_text> objects to customise how text appears at every level.

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Styling options:

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

The possible {position} suffixes mentioned below are x, x.top, x.bottom, y, y.left, y.right. The theta and r position suffixes in **ggplot2** are *not* obeyed in **legendry**.

Theme setting	Context	Type	Description
legendry.bracket	Both	element_line()	The bracket lines themselves.
legendry.bracket.size	Both	unit()	The space (in the orthogonal direction) afforded to a bracket.
axis.text.{position}	Axis	element_text()	The text over brackets.
legend.text	Legend	element_text()	The text over brackets.

Styling options *per level* can be set in the levels_brackets and levels_text arguments. These override theme settings.

Styling options *per range* can be set in the range key. The line and text prefixed properties are prioritised for the brackets and text respectively. These override theme settings and 'per level' settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
  primitive_bracket(theme = theme_guide(
    bracket = element_line(),
    bracket.size = unit(5, "mm")
  ))
```

Value

A <PrimitiveBracket> primitive guide that can be used inside other guides.

See Also

Other primitives: primitive_box(), primitive_fence(), primitive_labels(), primitive_line(), primitive_segments(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

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

key <- key_range_manual(c(2, 4), c(5, 6), c("A", "B"))

# Adding as secondary guides
p + guides(
  x.sec = primitive_bracket(),
  y.sec = primitive_bracket(key = key)
)
```

---

primitive_fence *Guide primitive: fence*

Description

This function constructs a fence guide primitive. The customisation options are easier to understand if we view fence 'post' as the vertical pieces of a real world fence, and the 'rail' as the horizontal pieces.

Usage

```
primitive_fence(
  key = "range_auto",
  rail = "none",
  angle = waiver(),
  oob = "squish",
  drop_zero = TRUE,
  pad_discrete = 0.5,
  levels_text = NULL,
  levels_post = NULL,
```

```
  levels_rail = NULL,
  theme = NULL,
  position = waiver()
)
```

Arguments

key	A range key specification. See more information in the linked topic.
rail	A <character[1]> giving an option for how to display fence railing. Can be either "none" (default) to display no railings, "inner" to draw one rail closer to the plot panel, "outer" to display one rail farther from the plot panel, or "both" to sandwich the labels between rails.
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
oob	A method for dealing with out-of-bounds (oob) ranges. Can be one of "squish", "censor" or "none".
drop_zero	A <logical[1]> whether to drop near-zero width ranges (TRUE, default) or preserve them (FALSE).
pad_discrete	A <numeric[1]> giving the amount ranges should be extended when given as a discrete variable. This is applied after the drop_zero setting.
levels_text	A list of <element_text> objects to customise how text appears at every level.
levels_post, levels_rail	A list of <element_line> objects to customise how fence posts and rails are displayed at every level.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Styling options:

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

The possible {position} suffixes mentioned below are x, x.top, x.bottom, y, y.left, y.right. The theta and r position suffixes in **ggplot2** are *not* obeyed in **legendry**.

Theme setting	Context	Type	Description
legendry.fence	Both	element_line()	Line segments for both 'post' and 'rail' segments
legendry.fence.post	Both	element_line()	Line segments orthogonal to the scale

legendry.fence.rail	Both	element_line()	Line segments parallel to the scale
axis.text.{position}	Axis	element_text()	The text labels at the fence.
legend.text	Legend	element_text()	The text labels at the fence.

Styling options *per level* can be set in the levels_post, levels_rail and levels_text arguments. These override theme settings.

Styling options *per range* can be set in the range key. The line and text prefixed properties are prioritised for the fence and text respectively. The 'post' and 'rail' distinction does not apply at the 'per range' settings. These override theme settings and the 'per level' settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
primitive_fence(theme = theme_guide(
  fence = element_line(),
  fence.post = element_line(),
  fence.rail = element_line(),
  text = element_text()
))
```

Value

A <PrimitiveFence> primitive guie that can be used inside other guides.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_labels(), primitive_line(), primitive_segments(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

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

key <- key_range_manual(c(2, 4), c(5, 6), c("A", "B"))

# Adding as secondary guides
p + guides(
  x.sec = primitive_fence(rail = "inner"),
  y.sec = primitive_fence(key = key, rail = "outer")
)
```

---

primitive_labels *Guide primitive: labels*

Description

This function constructs a labels guide primitive.

Usage

```
primitive_labels(
  key = NULL,
  angle = waiver(),
  n.dodge = 1L,
  check.overlap = FALSE,
  theme = NULL,
  position = waiver()
)
```

Arguments

key	A standard key specification. See more information in the linked topic.
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
n.dodge	An positive <integer[1]> setting the number of layers text labels can occupy to avoid overlapping labels.
check.overlap	A <logical[1]> indicating whether to check for and omit overlapping text. If TRUE, first, last and middle labels are recursively prioritised in that order. If FALSE, all labels are drawn.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Styling options:

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

The possible {position} suffixes mentioned below are x, x.top, x.bottom, y, y.left, y.right. The theta and r position suffixes in **ggplot2** are *not* obeyed in **legendry**.

Theme setting	Context	Type	Description
axis.text.{position}	Axis	element_text()	The text labels.
legend.text	Legend	element_text()	The text labels.

Styling options *per break* can be set in the standard key. The text prefixed properties are prioritised. These override theme settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
  primitive_labels(theme = theme_guide(
    text = element_line()
  ))
```

Value

A <PrimitiveLabels> primitive guide that can be used inside other guides.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_fence(), primitive_line(), primitive_segments(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

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

# Adding as secondary guides
p + guides(
  x.sec = primitive_labels(),
  y.sec = primitive_labels(n.dodge = 2)
)
```

---

primitive_line *Guide primitive: line*

Description

This function constructs a line guide primitive.

Usage

```
primitive_line(key = NULL, cap = "none", theme = NULL, position = waiver())
```

Arguments

key	A standard key specification. See more information in the linked topic.
cap	A method to cap the axes. One of the following: • A <character[1]> with one of the following: – "none" to perform no capping. – "both" to cap the line at both ends at the most extreme breaks. – "upper" to cap the line at the upper extreme break. – "lower" to cap the line at the lower extreme break. • A <logical>[1], where TRUE is equivalent to "both" and FALSE is equivalent to "none" in the options above.

- A sorted <numeric>[2n] with an even number of members. The lines will be drawn between every odd-even pair.
- A <function> that takes the scale's breaks as the first argument, the scale's limits as the second argument and returns a <numeric>[2n] as described above.

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Styling options:

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

Theme setting	Context	Type	Description
axis.line.{x/y}.{position}	Axis	element_line()	The axis line.
legend.axis.line	Legend	element_line()	The axis line.

There are no other styling options. The context-agnostic alternative to using theme() is to use theme_guide():

```
primitive_line(theme = theme_guide(
  line = element_line()
))
```

Value

A PrimitiveLine primitive guide that can be used inside other guides.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_fence(), primitive_labels(), primitive_segments(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

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

# Adding as secondary guides
p + guides(
  x.sec = primitive_line(),
  y.sec = primitive_line(cap = "both")
)
```

`primitive_segments` *Guide primitives: segments*

Description

This function constructs a guide primitive.

Usage

```
primitive_segments(
  key = NULL,
  space = rel(10),
  vanish = FALSE,
  theme = NULL,
  position = waiver()
)
```

Arguments

`key`	A segment key specification. See more information in the linked topic. Alternatively, an object of class `<hclust>` that automatically invokes `key_dendro()`.
`space`	Either a `<unit>` or `<rel>` object of length 1 determining the space allocated in the orthogonal direction. When the `space` argument is of class `<rel>` (default) the base size is taken from the tick length theme setting.
`vanish`	Only relevant when the guide is used in the secondary theta position: a `<logical[1]>` on whether the continue to draw the segments until they meed in the center (`TRUE`) or strictly observe the `space` setting (`FALSE`).
`theme`	A `<theme>` object to style the guide individually or differently from the plot's theme settings. The `theme` argument in the guide overrides and is combined with the plot's theme.
`position`	A `<character[1]>` giving the location of the guide. Can be one of `"top"`, `"bottom"`, `"left"` or `"right"`.

Details

Styling options:

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

The possible `{position}` suffixes mentioned below are `x`, `x.top`, `x.bottom`, `y`, `y.left`, `y.right`. The `theta` and `r` position suffixes in **ggplot2** are *not* obeyed in **legendry**.

Theme setting	Context	Type	Description
`axis.ticks.{position}`	Axis	`element_line()`	The line segments.
`axis.ticks.length.{position}`	Axis	`unit()`	Basis for the `space` argument
`legend.ticks`	Legend	`element_line()`	The line segments

legend.ticks.length	Legend	unit()	Basis for the space argument

Styling options *per segment* can be set in the segment key. The line prefixed properties are prioritised for segments. These override theme settings.

The context-agnostic alternative to using theme() is to use theme_guide():

```
primitive_segments(theme = theme_guide(
  ticks = element_line(),
  ticks.length = unit(5, "mm")
))
```

Value

A <PrimitiveSegments> primitive guide that can be used inside other guides.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_fence(), primitive_labels(), primitive_line(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

```
# Building a key
key <- key_segment_manual(
  value     = c(1.6, 1.6, 3.4, 5.2),
  value_end = c(7.0, 7.0, 3.4, 5.2),
  oppo      = c(1.0, 2.0, 0.0, 0.0),
  oppo_end  = c(1.0, 2.0, 3.0, 3.0)
)

# Using the primitive in a plot
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  scale_x_continuous(
    guide = primitive_segments(key = key)
  )
```

---

primitive_spacer *Guide primitive: spacer*

Description

This function constructs a spacer guide primitive. The spacer is intended for use in guide composition.

Usage

```
primitive_spacer(
  space = NULL,
  title = waiver(),
  theme = NULL,
  position = waiver()
)
```

Arguments

space	A [<unit[1]>][grid::unit()]
title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Styling options:

Below are the theme options that determine the styling of this guide. In context to many primitive guides, whether it is used in an axis or legend has no bearing on the style.

Theme setting	Context	Type	Description
legendry.guide.spacing	Any	unit()	Fallback amount of spacing when the space argument is NULL

There are no other styling options. The context-agnostic alternative to using theme() is to use theme_guide():

```
primitive_spacer(theme = theme_guide(
  spacing = unit(5, "mm"),
))
```

Value

A <PrimitiveSpacer> primitive guide that can be used inside other guides.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_fence(), primitive_labels(), primitive_line(), primitive_segments(), primitive_ticks(), primitive_title()

Examples

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  guides(
    x = guide_axis_stack("axis", primitive_spacer(unit(1, "cm")), "axis")
  )
```

`primitive_ticks` *Guide primitive: line*

Description

This function constructs a ticks guide primitive.

Usage

```
primitive_ticks(key = NULL, bidi = FALSE, theme = NULL, position = waiver())
```

Arguments

`key`	A standard key specification. See more information in the linked topic.
`bidi`	A `<logical[1]>`: whether ticks should be drawn bidirectionally (`TRUE`) or in a single direction (`FALSE`, default).
`theme`	A `<theme>` object to style the guide individually or differently from the plot's theme settings. The `theme` argument in the guide overrides and is combined with the plot's theme.
`position`	A `<character[1]>` giving the location of the guide. Can be one of `"top"`, `"bottom"`, `"left"` or `"right"`.

Details

Styling options:

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

The ticks can come in three variants: major, minor and minimal. Which variants are drawn depends on the keys: `key_minor()` draws major and minor ticks, whereas `key_log()` also has minimal ticks. Each variant has a corresponding length setting.

The possible `{position}` suffixes mentioned below are `x`, `x.top`, `x.bottom`, `y`, `y.left`, `y.right`. The `theta` and `r` position suffixes in **ggplot2** are *not* obeyed in **legendry**.

Theme setting	Context	Type	Description
`axis.ticks.{position}`	Axis	`element_line()`	Major tick lines
`axis.ticks.length.{position}`	Axis	`unit()`	Major tick length
`axis.minor.ticks.{position}`	Axis	`element_line()`	Minor tick lines
`axis.minor.ticks.length.{position}`	Axis	`unit()`	Minor tick length

legendry.axis.mini.ticks	Axis	element_line()	Minimal tick lines
legendry.axis.mini.ticks.length	Axis	unit()	Minimal tick length
legend.ticks	Legend	element_line()	Major tick lines
legend.ticks.length	Legend	unit()	Major ticks length
legendry.legend.minor.ticks	Legend	element_line()	Minor tick lines
legendry.legend.minor.ticks.length	Legend	unit()	Minor ticks length
legendry.legend.mini.ticks	Legend	element_line()	Minimal tick lines
legendry.legend.mini.ticks.length	Legend	unit()	Minimal tick length

Styling options *per break* can be set in the key. The `line` and prefixed properties are prioritised for the tick lines. These override theme settings.

The context-agnostic alternative to using `theme()` is to use `theme_guide()`:

```
primitive_ticks(theme = theme_guide(
  ticks = element_line(),
  ticks.length = unit(5, "mm"),
  minor.ticks = element_line(),
  minor.ticks.length = unit(4, "mm"),
  mini.ticks = element_line(),
  mini.ticks.length = unit(3, "mm")
))
```

Value

A `PrimitiveTicks` primitive guide that can be used inside other guides.

See Also

Other primitives: `primitive_box()`, `primitive_bracket()`, `primitive_fence()`, `primitive_labels()`, `primitive_line()`, `primitive_segments()`, `primitive_spacer()`, `primitive_title()`

Examples

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

# Adding as secondary guides
p + guides(x.sec = primitive_ticks(), y.sec = primitive_ticks())
```

---

primitive_title *Guide primitive: title*

Description

This function constructs a title guide primitive.

Usage

```
primitive_title(
  title = waiver(),
  angle = waiver(),
  theme = NULL,
  position = waiver()
)
```

Arguments

title	One of the following to indicate the title of the guide: • A <character[1]> or <expression[1]> to set a custom title. • NULL to not display any title. • waiver() (default) to take the name of the scale object or the name specified in labs() as the title.
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Styling options:

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

Theme setting	Context	Type	Description
axis.title.{x/y}.{position}	Axis	element_text()	The title itself.
legend.title	Legend	element_text()	The title itself.

There are no further styling options.

The context-agnostic alternative to using theme() is to use theme_guide():

```
primitive_title(theme = theme_guide(
  title = element_text(),
))
```

Value

A <PrimitiveTitle> primitive guide that can be used inside other guides.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_fence(), primitive_labels(), primitive_line(), primitive_segments(), primitive_spacer(), primitive_ticks()

Examples

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

# Adding as secondary guides
p + guides(
  x.sec = primitive_title("Horizontal Title"),
  y.sec = primitive_title(c("along vertical", "Multiple tiles"))
)

# 'Real' titles occur once per plot.
# Primitive titles repeat over facets and hide the 'real' title.
p + facet_wrap(~ drv) +
  guides(x = primitive_title("I am a repeated subtitle")) +
  labs(x = "I am the hidden real title")
```

---

scale_x_dendro *Dendrogram scales*

Description

These are speciality scales for use with hierarchically clustered data. The scale automatically orders the limits according to the clustering result and comes with a dendrogram axis.

Usage

```
scale_x_dendro(
  clust,
  ...,
  expand = waiver(),
  guide = "axis_dendro",
  position = "bottom"
)

scale_y_dendro(
  clust,
  ...,
  expand = waiver(),
```

```
  guide = "axis_dendro",
  position = "left"
)
```

Arguments

clust	A data structure that can be coerced to an <hclust> object through as.hclust().
...	Arguments passed on to ggplot2::discrete_scale aesthetics The names of the aesthetics that this scale works with. 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. 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 the options below. Please note that when labels is a vector, it is highly recommended to also set the breaks argument as a vector to protect against unintended mismatches. • NULL for no labels • waiver() for the default labels computed by the transformation object • A character vector giving labels (must be same length as breaks) • An expression vector (must be the same length as breaks). See ?plotmath for details. • A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation. 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. fallback.palette Function to use when palette = NULL and the palette is not represented in the theme. call The call used to construct the scale for reporting messages. super The super class to use for the constructed scale

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.
guide	A function used to create a guide or its name. See `guides()` for more information.
position	For position scales, The position of the axis. `left` or `right` for y axes, `top` or `bottom` for x axes.

Details

The scale limits are determined by the order and labels in the `clust` argument. While `limits` is not an argument in these scales, the `breaks` argument can still be used to selectively omit some breaks and the `labels` can be used for formatting purposes.

Value

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

See Also

`guide_axis_dendro()`

Examples

```
# Hierarchically cluster data, separately for rows and columns
car_clust <- hclust(dist(scale(mtcars)), "ave")
var_clust <- hclust(dist(scale(t(mtcars))), "ave")

long_mtcars <- data.frame(
  car = rownames(mtcars)[row(mtcars)],
  var = colnames(mtcars)[col(mtcars)],
  value = as.vector(scale(mtcars))
)

# A standard heatmap adorned with dendrograms
p <- ggplot(long_mtcars, aes(var, car, fill = value)) +
  geom_tile() +
  scale_x_dendro(var_clust) +
  scale_y_dendro(car_clust)
p

# Styling the dendrograms
p +
  guides(
    y = guide_axis_dendro(key_dendro(type = "triangle")),
    x = guide_axis_dendro(space = rel(5))
  ) +
  theme(
    axis.text.y.left = element_text(margin = margin(r = 3, l = 3)),
```

```
    axis.ticks.y = element_line("red"),
    axis.ticks.x = element_line(linetype = "dotted")
  )

# In polar coordinates, plus some formatting
p +
  coord_radial(
    theta = "y", inner.radius = 0.5,
    start = 0.25 * pi, end = 1.75 * pi
  ) +
  guides(
    theta = primitive_labels(angle = 90),
    theta.sec = primitive_segments("dendro", vanish = TRUE),
    r = guide_axis_dendro(angle = 0)
  )
```

---

theme_guide *Theme wrapper for guides*

Description

This function does three things different than `theme()`.

Usage

```
theme_guide(
  text = NULL,
  line = NULL,
  title = NULL,
  subtitle = NULL,
  text.position = NULL,
  title.position = NULL,
  subtitle.position = NULL,
  ticks = NULL,
  minor.ticks = NULL,
  mini.ticks = NULL,
  ticks.length = NULL,
  minor.ticks.length = NULL,
  mini.ticks.length = NULL,
  spacing = NULL,
  group.spacing = NULL,
  table.spacing = NULL,
  key = NULL,
  key.size = NULL,
  key.width = NULL,
  key.height = NULL,
  key.spacing = NULL,
  key.spacing.x = NULL,
```

```
  key.spacing.y = NULL,
  key.margin = NULL,
  key.justification = NULL,
  frame = NULL,
  byrow = NULL,
  background = NULL,
  margin = NULL,
  bracket = NULL,
  bracket.size = NULL,
  box = NULL,
  fence = NULL,
  fence.post = NULL,
  fence.rail = NULL,
  zebra.light = NULL,
  zebra.dark = NULL,
  point = NULL,
  connector = NULL
)
```

Arguments

`text` An <element_text> setting both `legend.text` and `axis.text` elements.

`line` An <element_line> setting both `legend.axis.line` and `axis.line` elements.

`title` An <element_text> setting both `legend.title` and `axis.title` elements.

`subtitle` An <element_text> setting both `legendry.legend.subtitle` and `legendry.axis.subtitle` elements.

`text.position`, `title.position`, `subtitle.position`
One of `"top"`, `"right"`, `"bottom"` or `"right"` setting the following elements:

- `text.position`: sets only `legend.text.position`.
- `title.position`: sets only `legend.title.position`.
- `subtitle.position` sets both `legendry.legend.subtitle.position` and `legendry.axis.subtitle.position`

`ticks` An <element_line> setting both `axis.ticks` and `legend.ticks` elements.

`minor.ticks` An <element_line> setting `legendry.legend.minor.ticks` and all 6 of the `axis.ticks.minor.{r/theta/x.top/x.bottom/y.left/y.right}` elements.

`mini.ticks` An <element_line> setting both `legendry.legend.mini.ticks` and `legendry.axis.mini.ticks` elements.

`ticks.length`, `minor.ticks.length`, `mini.ticks.length`
A [<unit[1]>][grid::unit()] setting the following elements:

- `ticks.length`: sets both `legend.ticks.length` and `axis.ticks.length`.
- `minor.ticks.length` sets both `axis.minor.ticks.length` and `legendry.legend.minor.ticks.`
- `mini.ticks.length` sets both `legendry.axis.mini.ticks.length` and `legendry.legend.mini.ticks.length`.

`spacing`, `group.spacing`, `table.spacing`
A [<unit[1]>][grid::unit()] settingthe `legendry.guide.spacing`, `legendry.group.spacing` and `legendry.table.spacing` theme elements respectively.

key An <element_rect> setting the legend.key element.

key.size, key.width, key.height
A <unit> setting the legend.key.size, legend.key.width and legend.key.height elements respectively.

key.spacing, key.spacing.x, key.spacing.y
A [<unit[1]>][grid::unit()] setting the legend.key.spacing, legend.key.spacing.x and legend.key.spacing.y elements respectively.

key.margin A <margin> setting the margin around legend glyphs.

key.justification
A [<numeric[2]>] passed to the legend.key.justification setting.

frame An <element_rect> setting the legend.frame element.

byrow A <logical[1]> setting the legend.byrow element.

background An <element_rect> setting the legend.background element.

margin A <margin> setting the legend.margin element.

bracket An <element_line> setting the legendry.bracket element.

bracket.size A [<unit[1]>][grid::unit()] setting the legendry.bracket.size element.

box An <element_rect> setting the legendry.box element.

fence, fence.post, fence.rail
An <element_line> setting the legendry.fence, legendry.fence.post and legendry.fence.rail respectively.

zebra.light, zebra.dark
An <element_rect> setting the legendry.zebra.light and legendry.zebra.dark settings respectively.

point An <element_point> setting the legendry.point element.

connector An <elemenet_line> setting the legendry.connector element.

Details

1. It has shorthand names for various guide settings, similar to theme_sub_legend() and theme_sub_axis()
2. It simultaneously sets theme elements for axes and legends. This makes it contextually agnostic. For example, setting theme_guide(text) will populate legend.text and also axis.text.
3. It includes **legendry** specific settings.

The first two things make it very good for setting the guide_*(theme) arguments. The second thing makes it very bad for setting plot-wide or global themes.

Value

A <theme> object that can be provided to a guide.

Examples

```
red_ticks <- theme_guide(
  ticks = element_line(colour = "red", linewidth = 0.5)
)

# Both axis and colourbar gain red ticks
ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point() +
  guides(
    colour = guide_colourbar(theme = red_ticks),
    x = guide_axis(theme = red_ticks)
  )
```

Index