# Package 'ggpattern'

June 4, 2024

**Type** Package

**Title** 'ggplot2' Pattern Geoms

**Version** 1.1.1

**Description**

Provides 'ggplot2' geoms filled with various patterns. Includes a patterned version of every 'ggplot2' geom that has a region that can be filled with a pattern. Provides a suite of 'ggplot2' aesthetics and scales for controlling pattern appearances. Supports over a dozen builtin patterns (every pattern implemented by 'gridpattern') as well as allowing custom user-defined patterns.

**URL** https://github.com/trevorld/ggpattern,

https://trevorldavis.com/R/ggpattern/dev/

**BugReports** https://github.com/trevorld/ggpattern/issues

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.1

**Imports** cli, ggplot2 (>= 3.5.1), glue, grid, gridpattern (>= 1.2.2), lifecycle, rlang (>= 1.1.3), scales, vctrs

**Suggests** ambient, dplyr, gganimate, knitr (>= 1.47), magick, mapproj, maps, png, ragg (>= 1.2.0), readr, rmarkdown (>= 2.27), sf (>= 0.7-3), svglite (>= 2.1.3), testthat (>= 2.1.0), vdiffr (>= 1.0.6)

**VignetteBuilder** knitr, ragg, rmarkdown

**Collate** 'a-geom-docs.R' 'aaa-ggplot2-compat-plyr.R'
'aaa-ggplot2-ggplot-global.R' 'aaa-ggplot2-performance.R'
'aaa-ggplot2-scale-manual.R' 'aaa-ggplot2-utilities-grid.R'
'aaa-ggplot2-utilities.R' 'geom-.R' 'geom-rect.R' 'geom-bar.R'
'geom-bin2d.R' 'geom-boxplot.R' 'geom-col.R' 'geom-crossbar.R'
'geom-ribbon.R' 'geom-density.R' 'geom-histogram.R'
'geom-polygon.R' 'geom-map.R' 'geom-sf.R' 'geom-tile.R'
'geom-violin.R' 'ggpattern-defunct.R' 'ggpattern-deprecated.R'
'ggpattern-package.R' 'pattern.R' 'polygon_df.R'
'scale-pattern-alpha.R' 'scale-pattern-auto.R'

'scale-pattern-brewer.R' 'scale-pattern-colour.R'
'scale-pattern-gradient.R' 'scale-pattern-grey.R'
'scale-pattern-hue.R' 'scale-pattern-linetype.R'
'scale-pattern-shape.R' 'scale-pattern-size.R'
'scale-pattern-viridis.R' 'scale-pattern.R' 'zxx.r' 'zzz.R'

**NeedsCompilation** no

**Author** Mike FC [aut],
Trevor L. Davis [aut, cre] (<<https://orcid.org/0000-0001-6341-4639>>),
ggplot2 authors [aut]

**Maintainer** Trevor L. Davis <trevor.l.davis@gmail.com>

# Contents

| create_polygon_df | *Create a* polygon_df *object from the given coordinates* |
|---|---|

## Description

code using polygon_df should not assume that the first and last point within each id are the same.
i.e. they may have to manulaly set a final point equal to the initial point if that is what their graphics
system desires

## Usage

```
create_polygon_df(x, y, id = 1L)
```

## Arguments

| | |
|---|---|
| x, y | coordinates of polygon. not necessarily closed. |
| id | a numeric vector used to separate locations in x,y into multiple polygons |

## Value

data.frame with x, y, id columns.

## Examples

```
df <- create_polygon_df(x = c(0, 0, 1, 1), y = c(0, 1, 1, 0))
is_polygon_df(df)
```

---

draw_key_polygon_pattern

*Key glyphs for legends*

---

## Description

Each geom has an associated function that draws the key when the geom needs to be displayed in a legend. These functions are called draw_key_*(), where * stands for the name of the respective key glyph. The key glyphs can be customized for individual geoms by providing a geom with the key_glyph argument (see [layer()](#) or examples below.)

## Usage

```
draw_key_polygon_pattern(data, params, size, aspect_ratio = get_aspect_ratio())

draw_key_boxplot_pattern(data, params, size, aspect_ratio = get_aspect_ratio())

draw_key_crossbar_pattern(
  data,
  params,
  size,
  aspect_ratio = get_aspect_ratio()
)
```

## Arguments

| | |
|---|---|
| data | A single row data frame containing the scaled aesthetics to display in this key |
| params | A list of additional parameters supplied to the geom. |
| size | Width and height of key in mm. |
| aspect_ratio | the geom's best guess at what the aspect ratio might be. |

**Value**

A grid grob.

**Examples**

```
if (require("ggplot2")) {

  # 'stripe' pattern example
  df <- data.frame(level = c("a", "b", "c", 'd'), outcome = c(2.3, 1.9, 3.2, 1))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = level),
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black',
      key_glyph = draw_key_polygon_pattern
    ) +
    theme_bw(18) +
    theme(legend.position = 'none') +
    labs(
      title    = "ggpattern::geom_col_pattern()",
      subtitle = "pattern = 'stripe'"
    )
  plot(gg)
}
```

---

geom-docs                          *ggplot2 geoms with support for pattern fills*

---

**Description**

All geoms in this package are identical to their counterparts in ggplot2 except that they can be filled with patterns.

**Usage**

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

```
geom_bar_pattern(
  mapping = NULL,
  data = NULL,
  stat = "count",
  position = "stack",
  ...,
  just = 0.5,
  width = NULL,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_bin_2d_pattern(
  mapping = NULL,
  data = NULL,
  stat = "bin2d",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_bin2d_pattern(
  mapping = NULL,
  data = NULL,
  stat = "bin2d",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_boxplot_pattern(
  mapping = NULL,
  data = NULL,
  stat = "boxplot",
  position = "dodge2",
  ...,
  outliers = TRUE,
  outlier.colour = NULL,
  outlier.color = NULL,
  outlier.fill = NULL,
  outlier.shape = 19,
  outlier.size = 1.5,
```

```
  outlier.stroke = 0.5,
  outlier.alpha = NULL,
  notch = FALSE,
  notchwidth = 0.5,
  staplewidth = 0,
  varwidth = FALSE,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_col_pattern(
  mapping = NULL,
  data = NULL,
  position = "stack",
  ...,
  just = 0.5,
  width = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_crossbar_pattern(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  fatten = 2.5,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_ribbon_pattern(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE,
  outline.type = "both"
```

```
)

geom_area_pattern(
  mapping = NULL,
  data = NULL,
  stat = "align",
  position = "stack",
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  outline.type = "upper"
)

geom_density_pattern(
  mapping = NULL,
  data = NULL,
  stat = "density",
  position = "identity",
  ...,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE,
  outline.type = "upper"
)

geom_histogram_pattern(
  mapping = NULL,
  data = NULL,
  stat = "bin",
  position = "stack",
  ...,
  binwidth = NULL,
  bins = NULL,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_polygon_pattern(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  rule = "evenodd",
```

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

geom_map_pattern(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  ...,
  map,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

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

geom_tile_pattern(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_violin_pattern(
  mapping = NULL,
  data = NULL,
  stat = "ydensity",
  position = "dodge",
  ...,
  draw_quantiles = NULL,
  trim = TRUE,
```

```
    bounds = c(-Inf, Inf),
    scale = "area",
    na.rm = FALSE,
    orientation = NA,
    show.legend = NA,
    inherit.aes = TRUE
)
```

## Arguments

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

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

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

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

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

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

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

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

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

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

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

- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.

- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.

- The key_glyph argument of [layer()](#) may also be passed on through .... This can be one of the functions described as [key glyphs](#), to change the display of the layer in the legend.

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

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

show.legend      logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

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

just             Adjustment for column placement. Set to 0.5 by default, meaning that columns will be centered about axis breaks. Set to 0 or 1 to place columns to the left/right of axis breaks. Note that this argument may have unintended behaviour when used with alternative positions, e.g. position_dodge().

width            Bar width. By default, set to 90% of the [resolution()](#) of the data.

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

outliers         Whether to display (TRUE) or discard (FALSE) outliers from the plot. Hiding or discarding outliers can be useful when, for example, raw data points need to be displayed on top of the boxplot. By discarding outliers, the axis limits will adapt to the box and whiskers only, not the full data range. If outliers need to be hidden and the axes needs to show the full data range, please use outlier.shape = NA instead.

outlier.colour,     outlier.color,     outlier.fill,     outlier.shape,
outlier.size, outlier.stroke, outlier.alpha
                 Default aesthetics for outliers. Set to NULL to inherit from the aesthetics used for the box.

| | |
|---|---|
| | In the unlikely event you specify both US and UK spellings of colour, the US spelling will take precedence. |
| notch | If `FALSE` (default) make a standard box plot. If `TRUE`, make a notched box plot. Notches are used to compare groups; if the notches of two boxes do not overlap, this suggests that the medians are significantly different. |
| notchwidth | For a notched box plot, width of the notch relative to the body (defaults to `notchwidth = 0.5`). |
| staplewidth | The relative width of staples to the width of the box. Staples mark the ends of the whiskers with a line. |
| varwidth | If `FALSE` (default) make a standard box plot. If `TRUE`, boxes are drawn with widths proportional to the square-roots of the number of observations in the groups (possibly weighted, using the `weight` aesthetic). |
| fatten | A multiplicative factor used to increase the size of the middle bar in `geom_crossbar()` and the middle point in `geom_pointrange()`. |
| outline.type | Type of the outline of the area; `"both"` draws both the upper and lower lines, `"upper"`/`"lower"` draws the respective lines only. `"full"` draws a closed polygon around the area. |
| binwidth | The width of the bins. Can be specified as a numeric value or as a function that calculates width from unscaled x. Here, "unscaled x" refers to the original x values in the data, before application of any scale transformation. When specifying a function along with a grouping structure, the function will be called once per group. The default is to use the number of bins in `bins`, covering the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data. |
| | The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds. |
| bins | Number of bins. Overridden by `binwidth`. Defaults to 30. |
| rule | Either `"evenodd"` or `"winding"`. If polygons with holes are being drawn (using the `subgroup` aesthetic) this argument defines how the hole coordinates are interpreted. See the examples in [`grid::pathGrob()`](#) for an explanation. |
| map | Data frame that contains the map coordinates. This will typically be created using [`fortify()`](#) on a spatial object. It must contain columns x or long, y or lat, and region or id. |
| draw_quantiles | If `not(NULL)` (default), draw horizontal lines at the given quantiles of the density estimate. |
| trim | If `TRUE` (default), trim the tails of the violins to the range of the data. If `FALSE`, don't trim the tails. |
| bounds | Known lower and upper bounds for estimated data. Default `c(-Inf, Inf)` means that there are no (finite) bounds. If any bound is finite, boundary effect of default density estimation will be corrected by reflecting tails outside bounds around their closest edge. Data points outside of bounds are removed with a warning. |
| scale | if "area" (default), all violins have the same area (before trimming the tails). If "count", areas are scaled proportionally to the number of observations. If "width", all violins have the same maximum width. |

**Value**

A [ggplot2::Geom](ggplot2::Geom) object.

**Pattern Arguments**

Not all arguments apply to all patterns.

`pattern` Pattern name string e.g. 'stripe' (default), 'crosshatch', 'point', 'circle', 'none'

`pattern_alpha` Alpha transparency for pattern. default: 1

`pattern_angle` Orientation of the pattern in degrees. default: 30

`pattern_aspect_ratio` Aspect ratio adjustment.

`pattern_colour` Colour used for strokes and points. default: 'black'

`pattern_density` Approximate fill fraction of the pattern. Usually in range [0, 1], but can be higher. default: 0.2

`pattern_filename` Image filename/URL.

`pattern_fill` Fill colour (or [grid::pattern()](grid::pattern())/gradient fill). default: 'grey80'

`pattern_fill2` Second fill colour (or [grid::pattern()](grid::pattern())/gradient fill). default: '#4169E1'

`pattern_filter` (Image scaling) filter. default: 'lanczos'

`pattern_frequency` Frequency. default: 0.1

`pattern_gravity` Image placement. default: 'center'

`pattern_grid` Pattern grid type. default: 'square'

`pattern_key_scale_factor` Scale factor for pattern in legend. default: 1

`pattern_linetype` Stroke linetype. default: 1

`pattern_option_1` Generic user value for custom patterns.

`pattern_option_2` Generic user value for custom patterns.

`pattern_option_3` Generic user value for custom patterns.

`pattern_option_4` Generic user value for custom patterns.

`pattern_option_5` Generic user value for custom patterns.

`pattern_orientation` 'vertical', 'horizontal', or 'radial'. default: 'vertical'

`pattern_res` Pattern resolution (pixels per inch).

`pattern_rot` Rotation angle (shape within pattern). default: 0

`pattern_scale` Scale. default: 1

`pattern_shape` Plotting shape. default: 1

`pattern_size` Stroke line width. default: 1

`pattern_spacing` Spacing of the pattern as a fraction of the plot size. default: 0.05

`pattern_type` Generic control option

`pattern_subtype` Generic control option

`pattern_xoffset` Offset the origin of the pattern. Range [0, 1]. default: 0. Use this to slightly shift the origin of the pattern. For most patterns, the user should limit the offset value to be less than the pattern spacing.

`pattern_yoffset` Offset the origin of the pattern. Range [0, 1]. default: 0. Use this to slightly shift the origin of the pattern. For most patterns, the user should limit the offset value to be less than the pattern spacing.

## Examples

```
if (require("ggplot2")) {

  # 'stripe' pattern example
  df <- data.frame(level = c("a", "b", "c", 'd'), outcome = c(2.3, 1.9, 3.2, 1))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = level),
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black'
    ) +
    theme_bw(18) +
    theme(legend.position = 'none') +
    labs(
      title    = "ggpattern::geom_col_pattern()",
      subtitle = "pattern = 'stripe'"
    )
  plot(gg)

  # 'pch' pattern example
  gg <- ggplot(mtcars, aes(as.factor(cyl), mpg)) +
    geom_violin_pattern(aes(fill = as.factor(cyl),
                            pattern_shape = as.factor(cyl)),
      pattern = 'pch',
      pattern_density = 0.3,
      pattern_angle = 0,
      colour  = 'black'
    ) +
    theme_bw(18) +
    theme(legend.position = 'none') +
    labs(
      title    = "ggpattern::geom_violin_pattern()",
      subtitle = "pattern = 'pch'"
    )
  plot(gg)

  # 'polygon_tiling' pattern example
  gg <- ggplot(mtcars) +
    geom_density_pattern(
      aes(
        x           = mpg,
        pattern_fill = as.factor(cyl),
        pattern_type = as.factor(cyl)
      ),
      pattern = 'polygon_tiling',
      pattern_key_scale_factor = 1.2
    ) +
    scale_pattern_type_manual(values = c("hexagonal", "rhombille",
                              "pythagorean")) +
    theme_bw(18) +
    theme(legend.key.size = unit(2, 'cm')) +
```

```
    labs(
      title   = "ggpattern::geom_density_pattern()",
      subtitle = "pattern = 'polygon_tiling'"
    )
  plot(gg)
}
```

GeomRectPattern            *Geom ggproto objects*

### Description

Geom ggproto objects that could be extended to create a new geom.

### See Also

[ggplot2::Geom](#)

ggpattern-defunct          *Defunct data/functions*

### Description

These data/functions are Defunct in this release of ggpattern.

1. For `magick_filter_names` use `magick::filter_types()` instead.
2. For `magick_gravity_names` use `magick::gravity_types()` instead.
3. For `magick_pattern_intensity_names` use `gridpattern::names_magick_intensity`.
4. For `magick_pattern_names` use `gridpattern::names_magick`.
5. For `magick_pattern_stripe_names` use `gridpattern::names_magick_stripe`.
6. For `placeholder_names` use `gridpattern::names_placeholder`.

### Usage

```
calculate_bbox_polygon_df(...)

convert_img_to_array(...)

convert_polygon_df_to_alpha_channel(...)

convert_polygon_df_to_polygon_grob(...)

convert_polygon_df_to_polygon_sf(...)
```

```
convert_polygon_sf_to_polygon_df(...)

create_gradient_img(...)

fetch_placeholder_img(...)

fill_area_with_img(...)

rotate_polygon_df(...)
```

## Arguments

...            Ignored

---

is_polygon_df            *Test if object is polygon_df or NULL*

---

## Description

Test if object is polygon_df or NULL

## Usage

```
is_polygon_df(x)
```

## Arguments

x            object

## Value

TRUE if object is polygon_df or NULL

## Examples

```
df <- create_polygon_df(x = c(0, 0, 1, 1), y = c(0, 1, 1, 0))
is_polygon_df(df)
```

---

scale_continuous          *Scales for continuous pattern aesthetics*

---

### Description

Scales for continuous pattern aesthetics

### Usage

```
scale_pattern_angle_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(0, 90),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_angle_discrete(..., range = c(0, 90))

scale_pattern_density_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(0, 0.5),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_density_discrete(..., range = c(0, 0.5))

scale_pattern_spacing_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(0.01, 0.1),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
```

```
)

scale_pattern_spacing_discrete(..., range = c(0.01, 0.1))

scale_pattern_xoffset_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(0.01, 0.1),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_xoffset_discrete(..., range = c(0.01, 0.1))

scale_pattern_yoffset_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(0.01, 0.1),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_yoffset_discrete(..., range = c(0.01, 0.1))

scale_pattern_aspect_ratio_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(0.5, 2),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_aspect_ratio_discrete(..., range = c(0.5, 2))

scale_pattern_key_scale_factor_continuous(
  name = waiver(),
```

```
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(0.5, 2),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_key_scale_factor_discrete(..., range = c(0.5, 2))

scale_pattern_scale_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(0.5, 2),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_scale_discrete(..., range = c(0.5, 2))

scale_pattern_phase_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = NULL,
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_phase_discrete(..., range = NULL)

scale_pattern_frequency_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = NULL,
  trans = deprecated(),
  guide = "legend",
```

```
  ...,
  transform = "identity"
)

scale_pattern_frequency_discrete(..., range = NULL)

scale_pattern_res_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = NULL,
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_res_discrete(..., range = NULL)

scale_pattern_rot_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(0, 360),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_rot_discrete(..., range = c(0, 360))
```

## Arguments

name, breaks, labels, limits, range, trans, guide, ..., transform
              See {ggplot2} documentation for more information on scales.

## Value

A [ggplot2::Scale](#) object.

## Examples

```
if (require('ggplot2')) {

  # 'stripe' pattern example
  df <- data.frame(level = c('a', 'b', 'c', 'd'),
                   outcome = c(2.3, 1.9, 3.2, 1))
```

```
    gg <- ggplot(df) +
      geom_col_pattern(
        aes(level, outcome, pattern_fill = level,
            pattern_density = outcome),
        pattern = 'stripe',
        fill    = 'white',
        colour  = 'black'
      ) +
      theme_bw(18) +
      theme(legend.position = 'none') +
      scale_pattern_density_continuous(range = c(0.1, 0.6)) +
      labs(
        title    = 'ggpattern::geom_col_pattern()',
        subtitle = 'pattern = \'stripe\''
      )
    plot(gg)
  }
```

---

| scale_discrete | *Scales for discrete pattern aesthetics* |
| --- | --- |

---

### Description

Scales for discrete pattern aesthetics

### Usage

```
scale_pattern_type_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  choices = NULL,
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_type_discrete(..., choices = NULL, guide = "legend")

scale_pattern_subtype_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  choices = NULL,
  trans = deprecated(),
```

```
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_subtype_discrete(..., choices = NULL, guide = "legend")

scale_pattern_filename_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  choices = NULL,
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_filename_discrete(..., choices = NULL, guide = "legend")

scale_pattern_filter_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  choices = c("lanczos", "box", "spline", "cubic"),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_filter_discrete(
  ...,
  choices = c("lanczos", "box", "spline", "cubic"),
  guide = "legend"
)

scale_pattern_gravity_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
 choices = c("center", "north", "south", "east", "west", "northeast", "northwest",
    "southeast", "southwest"),
  trans = deprecated(),
  guide = "legend",
```

```
  ...,
  transform = "identity"
)

scale_pattern_gravity_discrete(
  ...,
 choices = c("center", "north", "south", "east", "west", "northeast", "northwest",
    "southeast", "southwest"),
  guide = "legend"
)

scale_pattern_orientation_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  choices = c("horizontal", "vertical", "radial"),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_orientation_discrete(
  ...,
  choices = c("horizontal", "vertical", "radial"),
  guide = "legend"
)

scale_pattern_grid_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  choices = c("square", "hex"),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_grid_discrete(
  ...,
  choices = c("square", "hex"),
  guide = "legend"
)

scale_pattern_units_continuous(
```

```
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  choices = c("snpc", "cm", "inches"),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_units_discrete(
  ...,
  choices = c("snpc", "cm", "inches"),
  guide = "legend"
)

scale_pattern_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  choices = c("stripe", "crosshatch", "circle"),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity",
  na.value = "none"
)

scale_pattern_discrete(
  ...,
  choices = c("stripe", "crosshatch", "circle"),
  guide = "legend",
  na.value = "none"
)
```

### Arguments

name, breaks, labels, limits, trans, guide, ..., transform, na.value
                See {ggplot2} documentation for more information on scales.

choices          vector of values to choose from.

### Value

A [ggplot2::Scale](#) object.

## Examples

```
if (require('ggplot2')) {
  gg <- ggplot(mtcars) +
    geom_density_pattern(
      aes(
        x           = mpg,
        pattern_fill = as.factor(cyl),
        pattern_type = as.factor(cyl)
      ),
      pattern = 'polygon_tiling',
      pattern_key_scale_factor = 1.2
    ) +
    scale_pattern_type_discrete(choices = gridpattern::names_polygon_tiling) +
    theme_bw(18) +
    theme(legend.key.size = unit(2, 'cm')) +
    labs(
      title    = 'ggpattern::geom_density_pattern()',
      subtitle = 'pattern = \'polygon_tiling\''
    )
  plot(gg)
}
```

scale_pattern_alpha_continuous
                            *Alpha transparency scales*

---

## Description

See ggplot2::scale_alpha() for details.

## Usage

```
scale_pattern_alpha_continuous(..., range = c(0.1, 1))

scale_pattern_alpha(..., range = c(0.1, 1))

scale_pattern_alpha_discrete(...)

scale_pattern_alpha_ordinal(..., range = c(0.1, 1))
```

## Arguments

| | |
|---|---|
| ... | Other arguments passed on to continuous_scale(), binned_scale, or discrete_scale() as appropriate, to control name, limits, breaks, labels and so forth. |
| range | Output range of alpha values. Must lie between 0 and 1. |

## Value

A ggplot2::Scale object.

## Examples

```
if (require("ggplot2")) {
  # 'stripe' pattern example
  df <- data.frame(level = c("a", "b", "c", 'd'), outcome = c(2.3, 1.9, 3.2, 1))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = level, pattern_alpha = outcome),
      pattern_density = 0.6,
      pattern_size = 1.5,
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black',
      size = 1.5
    ) +
    theme_bw(18) +
    theme(legend.position = 'none') +
    scale_pattern_alpha() +
    labs(
      title    = "ggpattern::geom_col_pattern()",
      subtitle = "pattern = 'stripe'"
    )
  plot(gg)
}
```

---

scale_pattern_colour_brewer

*Sequential, diverging and qualitative colour scales from color-brewer.org*

---

## Description

The brewer scales provides sequential, diverging and qualitative colour schemes from ColorBrewer. These are particularly well suited to display discrete values on a map. See https://colorbrewer2. org for more information.

## Usage

```
scale_pattern_colour_brewer(
  ...,
  type = "seq",
  palette = 1,
  direction = 1,
  aesthetics = "pattern_colour"
)

scale_pattern_fill_brewer(
  ...,
  type = "seq",
```

```
  palette = 1,
  direction = 1,
  aesthetics = "pattern_fill"
)

scale_pattern_fill2_brewer(
  ...,
  type = "seq",
  palette = 1,
  direction = 1,
  aesthetics = "pattern_fill2"
)

scale_pattern_colour_distiller(
  ...,
  type = "seq",
  palette = 1,
  direction = -1,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_colour"),
  aesthetics = "pattern_colour"
)

scale_pattern_fill_distiller(
  ...,
  type = "seq",
  palette = 1,
  direction = -1,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill"),
  aesthetics = "pattern_fill"
)

scale_pattern_fill2_distiller(
  ...,
  type = "seq",
  palette = 1,
  direction = -1,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill2"),
  aesthetics = "pattern_fill2"
)
```

## Arguments

| | |
|---|---|
| `...` | Other arguments passed on to [discrete_scale()](#), [continuous_scale()](#), or [binned_scale()](#), for `brewer`, `distiller`, and `fermenter` variants respectively, to control name, limits, breaks, labels and so forth. |
| `palette` | If a string, will use that named palette. If a number, will index into the list of palettes of appropriate `type`. The list of available palettes can found in the Palettes section. |
| `direction, type, aesthetics, values, space, na.value, guide` | |
| | See `ggplot2::scale_colour_brewer` for more information. |

## Details

The `brewer` scales were carefully designed and tested on discrete data. They were not designed to be extended to continuous data, but results often look good. Your mileage may vary.

## Value

A [ggplot2::Scale](#) object.

## Palettes

The following palettes are available for use with these scales:

**Diverging** BrBG, PiYG, PRGn, PuOr, RdBu, RdGy, RdYlBu, RdYlGn, Spectral

**Qualitative** Accent, Dark2, Paired, Pastel1, Pastel2, Set1, Set2, Set3

**Sequential** Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu, YlOrBr, YlOrRd

Modify the palette through the `palette` arguement.

## Note

The `distiller` scales extend brewer to continuous scales by smoothly interpolating 7 colours from any palette to a continuous scale. The `fermenter` scales provide binned versions of the brewer scales.

## Examples

```
if (require("ggplot2")) {
  df <- data.frame(level = c("a", "b", "c", "d"),
                   outcome = c(2.3, 1.9, 3.2, 1))
  # discrete 'brewer' palette
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = level),
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black'
    ) +
    theme_bw(18) +
```

```
      scale_pattern_fill_brewer()
    plot(gg)

    # continuous 'distiller' palette
    gg <- ggplot(df) +
      geom_col_pattern(
        aes(level, outcome, pattern_fill = outcome),
        pattern = 'stripe',
        fill    = 'white',
        colour  = 'black'
      ) +
      theme_bw(18) +
      scale_pattern_fill_distiller()
    plot(gg)
  }
```

---

scale_pattern_colour_continuous
                        *Continuous and binned colour scales*

---

### Description

See ggplot2::scale_colour_continuous() for more information

### Usage

```
scale_pattern_colour_continuous(
  ...,
  type = getOption("ggplot2.continuous.colour", default = "gradient")
)

scale_pattern_fill_continuous(
  ...,
  type = getOption("ggplot2.continuous.fill", default = "gradient")
)

scale_pattern_fill2_continuous(
  ...,
  type = getOption("ggplot2.continuous.fill", default = "gradient")
)
```

### Arguments

| | |
|---|---|
| ... | Additional parameters passed on to the scale type |
| type | One of "gradient" (the default) or "viridis" indicating the colour scale to use |

### Value

A [ggplot2::Scale](#) object.

## Examples

```
if (require("ggplot2")) {
  df <- data.frame(level = c("a", "b", "c", "d"),
                   outcome = c(2.3, 1.9, 3.2, 1))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = outcome),
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black'
    ) +
    theme_bw(18) +
    scale_pattern_fill_continuous()
  plot(gg)
}
```

---

```
scale_pattern_colour_gradient
```
*Gradient colour scales*

---

## Description

See `ggplot2::scale_colour_gradient()` for more information

## Usage

```
scale_pattern_colour_gradient(
  ...,
  low = "#132B43",
  high = "#56B1F7",
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_colour"),
  aesthetics = "pattern_colour"
)

scale_pattern_fill_gradient(
  ...,
  low = "#132B43",
  high = "#56B1F7",
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill"),
  aesthetics = "pattern_fill"
)

scale_pattern_fill2_gradient(
  ...,
```

```
  low = "#132B43",
  high = "#56B1F7",
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill2"),
  aesthetics = "pattern_fill2"
)

scale_pattern_colour_gradient2(
  ...,
  low = muted("red"),
  mid = "white",
  high = muted("blue"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_colour"),
  aesthetics = "pattern_colour"
)

scale_pattern_fill_gradient2(
  ...,
  low = muted("red"),
  mid = "white",
  high = muted("blue"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill"),
  aesthetics = "pattern_fill"
)

scale_pattern_fill2_gradient2(
  ...,
  low = muted("red"),
  mid = "white",
  high = muted("blue"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill2"),
  aesthetics = "pattern_fill2"
)

scale_pattern_colour_gradientn(
  ...,
  colours,
  values = NULL,
```

```
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_colour"),
  aesthetics = "pattern_colour",
  colors
)

scale_pattern_fill_gradientn(
  ...,
  colours,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill"),
  aesthetics = "pattern_fill",
  colors
)

scale_pattern_fill2_gradientn(
  ...,
  colours,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill2"),
  aesthetics = "pattern_fill2",
  colors
)
```

## Arguments

| | |
|---|---|
| `low, high` | Colours for low and high ends of the gradient. |
| `space, ..., na.value, aesthetics` | |
| | See `scales::seq_gradient_pal`, `scale_colour_hue`, `ggplot2::continuous_scale` |
| `guide` | Type of legend. Use `"colourbar"` for continuous colour bar, or `"legend"` for discrete colour legend. |
| `mid` | colour for mid point |
| `midpoint` | The midpoint (in data value) of the diverging scale. Defaults to 0. |
| `colours, colors` | Vector of colours to use for n-colour gradient. |
| `values` | if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the `colours` vector. See [rescale()](#) for a convenience function to map an arbitrary range to between 0 and 1. |

## Details

`scale_*_gradient` creates a two colour gradient (low-high), `scale_*_gradient2` creates a diverging colour gradient (low-mid-high), `scale_*_gradientn` creates a n-colour gradient.

## Value

A [ggplot2::Scale](#) object.

## Examples

```
if (require("ggplot2")) {
  df <- data.frame(level = c("a", "b", "c", "d"),
                    outcome = c(2.3, 1.9, 3.2, 1))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = outcome),
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black'
    ) +
    theme_bw(18) +
    scale_pattern_fill_gradient()
  plot(gg)
}
```

---

scale_pattern_colour_grey

*Sequential grey colour scales*

---

## Description

Based on [gray.colors()](#). This is black and white equivalent of [scale_pattern_colour_gradient()](#).

## Usage

```
scale_pattern_colour_grey(
  ...,
  start = 0.2,
  end = 0.8,
  na.value = "red",
  aesthetics = "pattern_colour"
)

scale_pattern_fill_grey(
  ...,
  start = 0.2,
  end = 0.8,
  na.value = "red",
  aesthetics = "pattern_fill"
)

scale_pattern_fill2_grey(
  ...,
```

```
    start = 0.2,
    end = 0.8,
    na.value = "red",
    aesthetics = "pattern_fill2"
 )
```

## Arguments

```
..., start, end, na.value, aesthetics
```
                See ggplot2::scale_colour_grey for more information

## Value

A [ggplot2::Scale](ggplot2::Scale) object.

## Examples

```
if (require("ggplot2")) {
  df <- data.frame(level = c("a", "b", "c", "d"),
                   outcome = c(2.3, 1.9, 3.2, 1))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = level),
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black'
    ) +
    theme_bw(18) +
    scale_pattern_fill_grey()
  plot(gg)
}
```

---

scale_pattern_colour_hue

*Evenly spaced colours for discrete data*

---

## Description

This is the default colour scale for categorical variables. It maps each level to an evenly spaced hue
on the colour wheel. It does not generate colour-blind safe palettes.

## Usage

```
scale_pattern_colour_hue(
  ...,
  h = c(0, 360) + 15,
  c = 100,
  l = 65,
  h.start = 0,
```

```
    direction = 1,
    na.value = "grey50",
    aesthetics = "pattern_colour"
)

scale_pattern_fill_hue(
    ...,
    h = c(0, 360) + 15,
    c = 100,
    l = 65,
    h.start = 0,
    direction = 1,
    na.value = "grey50",
    aesthetics = "pattern_fill"
)

scale_pattern_fill2_hue(
    ...,
    h = c(0, 360) + 15,
    c = 100,
    l = 65,
    h.start = 0,
    direction = 1,
    na.value = "grey50",
    aesthetics = "pattern_fill2"
)
```

## Arguments

h, c, l, h.start, direction, ...

                See `ggplot2::scale_colour_hue`

na.value       Colour to use for missing values

aesthetics     Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the `colour` and `fill` aesthetics at the same time, via `aesthetics = c("colour", "fill")`.

## Value

A [ggplot2::Scale](#) object.

## Examples

```
if (require("ggplot2")) {
  df <- data.frame(level = c("a", "b", "c", "d"),
                   outcome = c(2.3, 1.9, 3.2, 1))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = level),
```

```
        pattern = 'stripe',
        fill    = 'white',
        colour  = 'black'
      ) +
      theme_bw(18) +
      scale_pattern_fill_hue()
    plot(gg)
  }
```

---

scale_pattern_colour_viridis_d
                    *Viridis colour scales from viridisLite*

---

### Description

The `viridis` scales provide colour maps that are perceptually uniform in both colour and black-and-white. They are also designed to be perceived by viewers with common forms of colour blindness. See also https://bids.github.io/colormap/.

### Usage

```
scale_pattern_colour_viridis_d(
  ...,
  alpha = 1,
  begin = 0,
  end = 1,
  direction = 1,
  option = "D",
  aesthetics = "pattern_colour"
)

scale_pattern_fill_viridis_d(
  ...,
  alpha = 1,
  begin = 0,
  end = 1,
  direction = 1,
  option = "D",
  aesthetics = "pattern_fill"
)

scale_pattern_fill2_viridis_d(
  ...,
  alpha = 1,
  begin = 0,
  end = 1,
  direction = 1,
```

```
  option = "D",
  aesthetics = "pattern_fill2"
)

scale_pattern_colour_viridis_c(
  ...,
  alpha = 1,
  begin = 0,
  end = 1,
  direction = 1,
  option = "D",
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_colour"),
  aesthetics = "pattern_colour"
)

scale_pattern_fill_viridis_c(
  ...,
  alpha = 1,
  begin = 0,
  end = 1,
  direction = 1,
  option = "D",
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill"),
  aesthetics = "pattern_fill"
)

scale_pattern_fill2_viridis_c(
  ...,
  alpha = 1,
  begin = 0,
  end = 1,
  direction = 1,
  option = "D",
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = guide_colourbar(available_aes = "pattern_fill2"),
  aesthetics = "pattern_fill2"
)
```

## Arguments

| | |
|---|---|
| `...` | Other arguments passed on to [discrete_scale()](), [continuous_scale()](), or [binned_scale]() to control name, limits, breaks, labels and so forth. |
| `begin, end, alpha, direction, option, values, space, na.value, guide` | |
| | See `ggplot2::scale_colour_viridis_d` for more information |
| `aesthetics` | Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the `colour` and `fill` aesthetics at the same time, via `aesthetics = c("colour", "fill")`. |

## Value

A [ggplot2::Scale]() object.

## Examples

```
if (require("ggplot2")) {
  df <- data.frame(level = c("a", "b", "c", "d"),
                   outcome = c(2.3, 1.9, 3.2, 1))
  # discrete 'viridis' palette
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = level),
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black'
    ) +
    theme_bw(18) +
    scale_pattern_fill_viridis_d()
  plot(gg)

  # continuous 'viridis' palette
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = outcome),
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black'
    ) +
    theme_bw(18) +
    scale_pattern_fill_viridis_c()
  plot(gg)
}
```

---

scale_pattern_identity

*Use values without scaling*

---

**Description**

Use values without scaling

**Usage**

```
scale_pattern_type_identity(..., guide = "none")

scale_pattern_subtype_identity(..., guide = "none")

scale_pattern_angle_identity(..., guide = "none")

scale_pattern_density_identity(..., guide = "none")

scale_pattern_spacing_identity(..., guide = "none")

scale_pattern_xoffset_identity(..., guide = "none")

scale_pattern_yoffset_identity(..., guide = "none")

scale_pattern_alpha_identity(..., guide = "none")

scale_pattern_linetype_identity(..., guide = "none")

scale_pattern_size_identity(..., guide = "none")

scale_pattern_shape_identity(..., guide = "none")

scale_pattern_colour_identity(..., guide = "none")

scale_pattern_fill_identity(..., guide = "none")

scale_pattern_fill2_identity(..., guide = "none")

scale_pattern_aspect_ratio_identity(..., guide = "none")

scale_pattern_key_scale_factor_identity(..., guide = "none")

scale_pattern_filename_identity(..., guide = "none")

scale_pattern_filter_identity(..., guide = "none")

scale_pattern_gravity_identity(..., guide = "none")

scale_pattern_scale_identity(..., guide = "none")

scale_pattern_orientation_identity(..., guide = "none")

scale_pattern_phase_identity(..., guide = "none")
```

```
scale_pattern_frequency_identity(..., guide = "none")

scale_pattern_grid_identity(..., guide = "none")

scale_pattern_res_identity(..., guide = "none")

scale_pattern_rot_identity(..., guide = "none")

scale_pattern_units_identity(..., guide = "none")

scale_pattern_identity(..., guide = "none")
```

## Arguments

| | |
|---|---|
| `..., guide` | See ggplot2 for documentation on identity scales. e.g. `ggplot2::scale_alpha_identity()` |

## Value

A [ggplot2::Scale](#) object.

## Examples

```
if (require('ggplot2')) {
  df <- data.frame(outcome = c(2.3, 1.9, 3.2, 1),
                   pattern_type = sample(gridpattern::names_polygon_tiling, 4))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(pattern_type, outcome, pattern_fill = pattern_type,
          pattern_type = pattern_type),
      colour = 'black',
      pattern = 'polygon_tiling',
      pattern_key_scale_factor = 1.2
    ) +
    scale_pattern_type_identity() +
    theme_bw(18) +
    theme(legend.position = 'none') +
    labs(
      x       = 'level',
      title   = 'ggpattern::geom_col_pattern()',
      subtitle = 'pattern = \'polygon_tiling\''
    )
  plot(gg)
}
```

---

```
scale_pattern_linetype
```
                      *Scale for line patterns*

---

**Description**

Default line types based on a set supplied by Richard Pearson, University of Manchester. Continuous values can not be mapped to line types.

**Usage**

```
scale_pattern_linetype(..., na.value = "blank")

scale_pattern_linetype_continuous(...)

scale_pattern_linetype_discrete(..., na.value = "blank")
```

**Arguments**

| ... | see ggplot2::scale_linetype for more information |
|---|---|
| na.value | The linetype to use for NA values. |

**Value**

A [ggplot2::Scale](#) object.

**Examples**

```
if (require("ggplot2")) {
  # 'stripe' pattern example
  df <- data.frame(level = c("a", "b", "c", 'd'), outcome = c(2.3, 1.9, 3.2, 1))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = level, pattern_linetype = level),
      pattern_density = 0.6,
      pattern_size = 1.5,
      pattern = 'stripe',
      fill    = 'white',
      colour  = 'black',
      size = 1.5
    ) +
    theme_bw(18) +
    theme(legend.position = 'none') +
    scale_pattern_linetype() +
    labs(
      title    = "ggpattern::geom_col_pattern()",
      subtitle = "pattern = 'stripe'"
    )
  plot(gg)
}
```

scale_pattern_manual      *Create your own discrete scale*

### Description

Create your own discrete scale

### Usage

```
scale_pattern_type_manual(..., values, breaks = waiver())

scale_pattern_subtype_manual(..., values, breaks = waiver())

scale_pattern_angle_manual(..., values, breaks = waiver())

scale_pattern_density_manual(..., values, breaks = waiver())

scale_pattern_spacing_manual(..., values, breaks = waiver())

scale_pattern_xoffset_manual(..., values, breaks = waiver())

scale_pattern_yoffset_manual(..., values, breaks = waiver())

scale_pattern_alpha_manual(..., values, breaks = waiver())

scale_pattern_linetype_manual(..., values, breaks = waiver())

scale_pattern_size_manual(..., values, breaks = waiver())

scale_pattern_shape_manual(..., values, breaks = waiver())

scale_pattern_colour_manual(..., values, breaks = waiver())

scale_pattern_fill_manual(..., values, breaks = waiver())

scale_pattern_fill2_manual(..., values, breaks = waiver())

scale_pattern_aspect_ratio_manual(..., values, breaks = waiver())

scale_pattern_key_scale_factor_manual(..., values, breaks = waiver())

scale_pattern_filename_manual(..., values, breaks = waiver())

scale_pattern_filter_manual(..., values, breaks = waiver())

scale_pattern_gravity_manual(..., values, breaks = waiver())
```

```
scale_pattern_scale_manual(..., values, breaks = waiver())

scale_pattern_orientation_manual(..., values, breaks = waiver())

scale_pattern_phase_manual(..., values, breaks = waiver())

scale_pattern_frequency_manual(..., values, breaks = waiver())

scale_pattern_grid_manual(..., values, breaks = waiver())

scale_pattern_res_manual(..., values, breaks = waiver())

scale_pattern_rot_manual(..., values, breaks = waiver())

scale_pattern_units_manual(..., values, breaks = waiver())

scale_pattern_manual(..., values, breaks = waiver(), na.value = "none")
```

## Arguments

`..., values, breaks, na.value`

                 See `ggplot2` for documentation on manual scales. e.g. `ggplot2::scale_colour_manual()`

## Value

A [ggplot2::Scale](#) object.

## Examples

```
if (require('ggplot2')) {
  gg <- ggplot(mtcars) +
    geom_density_pattern(
      aes(
        x           = mpg,
        pattern_fill = as.factor(cyl),
        pattern_type = as.factor(cyl)
      ),
      pattern = 'polygon_tiling',
      pattern_key_scale_factor = 1.2
    ) +
    scale_pattern_type_manual(values = c('hexagonal', 'rhombille',
                                 'pythagorean')) +
    theme_bw(18) +
    theme(legend.key.size = unit(2, 'cm')) +
    labs(
      title   = 'ggpattern::geom_density_pattern()',
      subtitle = 'pattern = \'polygon_tiling\''
    )
  plot(gg)
}
```

scale_pattern_shape    *Scales for shapes, aka glyphs*

---

### Description

scale_pattern_shape maps discrete variables to six easily discernible shapes. If you have more than six levels, you will get a warning message, and the seventh and subsequence levels will not appear on the plot. Use scale_pattern_shape_manual() to supply your own values. You can not map a continuous variable to shape unless scale_pattern_shape_binned() is used. Still, as shape has no inherent order, this use is not advised..

### Usage

```
scale_pattern_shape(..., solid = TRUE)

scale_pattern_shape_discrete(..., solid = TRUE)

scale_pattern_shape_ordinal(...)

scale_pattern_shape_continuous(...)
```

### Arguments

| | |
|---|---|
| ... | other arguments passed to discrete_scale() |
| solid | Should the shapes be solid, TRUE, or hollow, FALSE? |

### Details

Scales for area or radius

### Value

A ggplot2::Scale object.

### Examples

```
if (require("ggplot2")) {
  # 'pch' pattern example
  gg <- ggplot(mtcars, aes(as.factor(cyl), mpg)) +
    geom_violin_pattern(aes(fill = as.factor(cyl),
                            pattern_shape = as.factor(cyl)),
      pattern = 'pch',
      pattern_density = 0.3,
      pattern_angle = 0,
      colour  = 'black'
    ) +
    theme_bw(18) +
    theme(legend.position = 'none') +
```

```
        scale_pattern_shape() +
        labs(
          title    = "ggpattern::geom_violin_pattern()",
          subtitle = "pattern = 'pch'"
        )
    plot(gg)
  }
```

scale_pattern_size_continuous

*Scales for area or radius*

### Description

Scales for area or radius

### Usage

```
scale_pattern_size_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(1, 6),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)

scale_pattern_size(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(1, 6),
  trans = deprecated(),
  guide = "legend",
  ...,
  transform = "identity"
)
```

### Arguments

name, breaks, labels, limits, trans, guide, ..., transform
                See [ggplot2::scale_size()](#) for more information

range           a numeric vector of length 2 that specifies the minimum and maximum size of
                the plotting symbol after transformation.

**Value**

A [ggplot2::Scale](#) object.

**Examples**

```
if (require("ggplot2")) {
  # 'circle' pattern example
  df <- data.frame(level = c("a", "b", "c", 'd'), outcome = c(2.3, 1.9, 3.2, 1))
  gg <- ggplot(df) +
    geom_col_pattern(
      aes(level, outcome, pattern_fill = level,
          size = outcome, pattern_size = outcome),
      pattern_density = 0.4,
      pattern_spacing = 0.3,
      pattern = 'circle',
      fill    = 'white',
      colour  = 'black'
    ) +
    theme_bw(18) +
    theme(legend.position = 'none') +
    scale_pattern_size() +
    labs(
      title    = "ggpattern::geom_col_pattern()",
      subtitle = "pattern = 'circle'"
    )
  plot(gg)
}
```

# Index