Package: gridpattern (via r-universe)

October 28, 2026

Type Package

Title 'grid' Pattern Grobs

Version 1.5.1

Description Provides 'grid' grobs that fill in a user-defined area with various patterns. Includes enhanced versions of the geometric and image-based patterns originally contained in the 'ggpattern' package as well as original 'pch', 'polygon_tiling', 'regular_polygon', 'rose', 'text', 'wave', and 'weave' patterns plus support for custom user-defined patterns.

URL https://trevorldavis.com/R/gridpattern/, https://github.com/trevorld/gridpattern

BugReports https://github.com/trevorld/gridpattern/issues

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.4.0)

Imports glue, grDevices, grid, memoise, png, rlang, sf, utils

Suggests ambient, aqp, aRtsy, ggplot2 (>= 3.5.0), gtable, knitr, magick (>= 2.7.4), ragg (>= 1.2.0), rmarkdown, scales, svglite (>= 2.1.0), testthat, Unicode, vdiffr (>= 1.0.6), withr

VignetteBuilder knitr, rmarkdown

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Trevor L. Davis [aut, cre] (ORCID: <https://orcid.org/0000-0001-6341-4639>), Mike FC [aut] (Code/docs adapted from ggpattern), ggplot2 authors [ctb] (some utility functions copied from ggplot2)

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

Repository https://cran.r-universe.dev

Date/Publication 2026-10-28 11:10:08 UTC

RemoteUrl https://github.com/cran/gridpattern

RemoteRef HEAD

RemoteSha e9a7c3c86b5e6b51b8bd8e99f83c50d29c7d1fdc

Contents

gridpattern-package . . . 3
alphaMaskGrob . . . 4
clippingPathGrob . . . 5
grid.pattern . . . 7
grid.pattern_ambient . . . 10
grid.pattern_aRtsy . . . 13
grid.pattern_circle . . . 14
grid.pattern_crosshatch . . . 16
grid.pattern_fill . . . 18
grid.pattern_gradient . . . 19
grid.pattern_hatch . . . 21
grid.pattern_image . . . 24
grid.pattern_line . . . 26
grid.pattern_magick . . . 28
grid.pattern_none . . . 30
grid.pattern_pch . . . 31
grid.pattern_placeholder . . . 34
grid.pattern_plasma . . . 35
grid.pattern_polygon_tiling . . . 37
grid.pattern_regular_polygon . . . 40
grid.pattern_rose . . . 43
grid.pattern_stripe . . . 46
grid.pattern_text . . . 48
grid.pattern_wave . . . 50
grid.pattern_weave . . . 53
guess_has_R4.1_features . . . 56
local_pattern . . . 57
mean_col . . . 59
mix_col . . . 60
pattern_hex . . . 61
pattern_square . . . 62
pattern_weave . . . 64
patternFill . . . 66
prefer_pattern . . . 67
register_pattern . . . 69
registered_patterns . . . 72
reset_image_cache . . . 74
star_scale . . . 74
update_alpha . . . 75

Index **77**

`gridpattern-package` *gridpattern: 'grid' Pattern Grobs*

Description

Provides 'grid' grobs that fill in a user-defined area with various patterns. Includes enhanced versions of the geometric and image-based patterns originally contained in the 'ggpattern' package as well as original 'pch', 'polygon_tiling', 'regular_polygon', 'rose', 'text', 'wave', and 'weave' patterns plus support for custom user-defined patterns.

Package options

The following `gridpattern` options may be set globally via `base::options()`:

ggpattern_array_funcs Legacy option to set custom "array" pattern functions. Should use `register_pattern(kind = "array")` instead.

ggpattern_geometry_funcs Legacy option to set custom "geometry" pattern functions. Should use `register_pattern()` instead.

ggpattern_res Set custom raster image resolution (pixels per inch) for certain patterns.

ggpattern_use_R4.1_clipping If `TRUE` use the grid clipping path feature introduced in R v4.1.0. If `FALSE` do a `rasterGrob` approximation of the clipped pattern. If `NULL` try to guess an appropriate choice.

ggpattern_use_R4.1_features If `TRUE` sets the default for all the other `ggpattern_use_R4.1_*` options arguments to `TRUE`. If `FALSE` sets them to `FALSE`.

ggpattern_use_R4.1_gradients If `TRUE` use the grid gradient feature introduced in R v4.1.0. If `FALSE` do a `rasterGrob` approximation of the gradient pattern. If `NULL` try to guess an appropriate choice.

ggpattern_use_R4.1_masks If `TRUE` use the grid mask feature introduced in R v4.1.0. If `FALSE` do a `rasterGrob` approximation of the masked pattern. If `NULL` try to guess an appropriate choice.

ggpattern_use_R4.1_patterns If `TRUE` use the grid pattern feature introduced in R v4.1.0. Currently only used by a couple of examples.

Note to use the R v4.1.0 features one needs R be (at least) version 4.1 and not all graphic devices support any/all these features. See `https://www.stat.auckland.ac.nz/~paul/Reports/GraphicsEngine/definitions/definitions.html` for more information on these features.

Author(s)

Maintainer: Trevor L. Davis `<trevor.l.davis@gmail.com>` (ORCID)

Authors:

- Trevor L. Davis `<trevor.l.davis@gmail.com>` (ORCID)
- Mike FC (Code/docs adapted from ggpattern)

Other contributors:

- ggplot2 authors (some utility functions copied from ggplot2) [contributor]

See Also

Useful links:

- https://trevorldavis.com/R/gridpattern/
- https://github.com/trevorld/gridpattern
- Report bugs at https://github.com/trevorld/gridpattern/issues

---

`alphaMaskGrob` *Mask grob using another grob to specify the (alpha) mask*

Description

`alphaMaskGrob()` masks a grob using another grob to specify the (alpha) mask.

Usage

```
alphaMaskGrob(
  maskee,
  masker,
  use_R4.1_masks = getOption("ggpattern_use_R4.1_masks",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  res = getOption("ggpattern_res", 72),
  name = NULL,
  gp = gpar(),
  vp = NULL
)
```

Arguments

`maskee`	Grob to be masked
`masker`	Grob that defines masking region
`use_R4.1_masks`	If `TRUE` use the grid mask feature introduced in R v4.1.0. If `FALSE` do a `rasterGrob` approximation. If `NULL` try to guess an appropriate choice. Note not all graphic devices support the grid mask feature.
`png_device`	"png" graphics device to save intermediate raster data with if `use_R4.1_masks` is `FALSE`. If `NULL` and suggested package `ragg` is available and versions are high enough we directly capture masked raster via `ragg::agg_capture()`. Otherwise we will use `png_device` (default `ragg::agg_png()` if available else `grDevices::png()`) and `png::readPNG()` to manually compute a masked raster.
`res`	Resolution of desired `rasterGrob` in pixels per inch if `use_R4.1_masks` is `FALSE`.
`name`	A character identifier.
`gp`	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
`vp`	A Grid viewport object (or NULL).

Value

A grid grob

Examples

```
# May take more than 5 seconds on CRAN servers
if (capabilities("png") && require("grid")) {
  maskee <- patternGrob("circle", gp = gpar(col = "black", fill = "yellow"),
                        spacing = 0.1, density = 0.5)
  angle <- seq(2 * pi / 4, by = 2 * pi / 6, length.out = 7)
  x_hex_outer <- 0.5 + 0.5 * cos(angle)
  y_hex_outer <- 0.5 + 0.5 * sin(angle)
  x_hex_inner <- 0.5 + 0.25 * cos(rev(angle))
  y_hex_inner <- 0.5 + 0.25 * sin(rev(angle))
  gp <- gpar(lwd = 0, col = NA, fill = "white")
  masker <- grid::pathGrob(x = c(x_hex_outer, x_hex_inner),
                           y = c(y_hex_outer, y_hex_inner),
                           id = rep(1:2, each = 7),
                           rule = "evenodd", gp = gp)
  masked <- alphaMaskGrob(maskee, masker, use_R4.1_masks = FALSE)
  grid.draw(masked)
}
if (capabilities("png") && require("grid")) {
  maskee_transparent <- rectGrob(gp = gpar(col = NA, fill = "blue"))
  gp <- gpar(lwd = 20, col = "black", fill = grDevices::rgb(0, 0, 0, 0.5))
  masker_transparent <- editGrob(masker, gp = gp)
  masked_transparent <- alphaMaskGrob(maskee_transparent,
                                      masker_transparent,
                                      use_R4.1_masks = FALSE)
  grid.newpage()
  grid.draw(masked_transparent)
}
```

---

clippingPathGrob *Clip grob using another grob to specify the clipping path*

Description

clippingPathGrob() clips a grob using another grob to specify the clipping path

Usage

```
clippingPathGrob(
  clippee,
  clipper,
  use_R4.1_clipping = getOption("ggpattern_use_R4.1_clipping",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
```

```
  res = getOption("ggpattern_res", 72),
  name = NULL,
  gp = gpar(),
  vp = NULL
)
```

Arguments

clippee	Grob to be clipped
clipper	Grob that defines clipping region
use_R4.1_clipping	If TRUE use the grid clipping path feature introduced in R v4.1.0. If FALSE do a rasterGrob approximation. If NULL try to guess an appropriate choice. Note not all graphic devices support the grid clipping path feature and the grid clipping path feature does not nest.
png_device	"png" graphics device to save intermediate raster data with if use_R4.1_clipping is FALSE. If NULL and suggested package ragg is available and versions are high enough we directly capture clipped raster via ragg::agg_capture(). Otherwise we will use png_device (default ragg::agg_png() if available else grDevices::png()) and png::readPNG() to manually compute a clipped raster.
res	Resolution of desired rasterGrob in pixels per inch if use_R4.1_clipping is FALSE.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
vp	A Grid viewport object (or NULL).

Value

A grid grob

Examples

```
if (capabilities("png") && require("grid")) {
  clippee <- patternGrob("circle", gp = gpar(col = "black", fill = "yellow"),
                         spacing = 0.1, density = 0.5)
  angle <- seq(2 * pi / 4, by = 2 * pi / 6, length.out = 7)
  x_hex_outer <- 0.5 + 0.5 * cos(angle)
  y_hex_outer <- 0.5 + 0.5 * sin(angle)
  x_hex_inner <- 0.5 + 0.25 * cos(rev(angle))
  y_hex_inner <- 0.5 + 0.25 * sin(rev(angle))
  clipper <- grid::pathGrob(x = c(x_hex_outer, x_hex_inner),
                            y = c(y_hex_outer, y_hex_inner),
                            id = rep(1:2, each = 7),
                            rule = "evenodd")
  clipped <- clippingPathGrob(clippee, clipper, use_R4.1_clipping = FALSE)
  grid.newpage()
  grid.draw(clipped)
}
```

grid.pattern *Create patterned grobs by pattern name*

Description

grid.pattern() draws patterned shapes onto the graphic device. patternGrob() returns the grid grob objects. names_pattern is a character vector of builtin patterns.

Usage

```
grid.pattern(
  pattern = "stripe",
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  legend = FALSE,
  prefix = "pattern_",
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_pattern

patternGrob(
  pattern = "stripe",
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  legend = FALSE,
  prefix = "pattern_",
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

pattern	Name of pattern. See Details section for a list of supported patterns. Builtin patterns may also be referred to by their qualified name e.g. "gridpattern::stripe"

	and custom patterns registered by packages by "pkg::name". See register_pattern() for more information.
x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Pattern parameters.
legend	Whether this is intended to be drawn in a legend or not.
prefix	Prefix to prepend to the name of each of the pattern parameters in .... For compatibility with ggpattern most underlying functions assume parameters beginning with pattern_.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Details

Here is a list of the various patterns supported:

ambient Noise array patterns onto the graphic device powered by the ambient package. See grid.pattern_ambient() for more information.

aRtsy Patterns powered by the aRtsy package. See grid.pattern_aRtsy() for more information.

circle Circle geometry patterns. See grid.pattern_circle() for more information.

crosshatch Crosshatch geometry patterns. See grid.pattern_crosshatch() for more information.

gradient Gradient array/geometry patterns. See grid.pattern_gradient() for more information.

hatch Heraldic hatching patterns. See grid.pattern_hatch() for more information.

image Image array patterns. See grid.pattern_image() for more information.

line Line geometry patterns. See grid.pattern_line() for more information.

magick imagemagick array patterns. See grid.pattern_magick() for more information.

none Does nothing. See grid::grid.null() for more information.

pch Plotting character geometry patterns. See grid.pattern_pch() for more information.

placeholder Placeholder image array patterns. See grid.pattern_placeholder() for more information.

plasma Plasma array patterns. See grid.pattern_plasma() for more information.

polygon_tiling Polygon tiling patterns. See grid.pattern_polygon_tiling() for more information.

regular_polygon Regular polygon patterns. See `grid.pattern_regular_polygon()` for more information.

rose Rose array/geometry patterns. See `grid.pattern_rose()` for more information.

stripe Stripe geometry patterns. See `grid.pattern_stripe()` for more information.

text Text array/geometry patterns. See `grid.pattern_text()` for more information.

wave Wave geometry patterns. See `grid.pattern_wave()` for more information.

weave Weave geometry patterns. See `grid.pattern_weave()` for more information.

Custom geometry-based patterns See `register_pattern()` and the "Developing Patterns" vignette `vignette("developing-patterns", package = "gridpattern")` for more information.

Custom array-based patterns See `register_pattern()` and the "Developing Patterns" vignette `vignette("developing-patterns", package = "gridpattern")` for more information.

Value

A grid grob object (invisibly in the case of `grid.pattern()`). If `draw` is `TRUE` then `grid.pattern()` also draws to the graphic device as a side effect.

See Also

https://coolbutuseless.github.io/package/ggpattern/index.html for more details on the ggpattern package.

Examples

```
print(names_pattern)
# May take more than 5 seconds on CRAN servers
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))

# geometry-based patterns
# 'stripe' pattern
grid::grid.newpage()
grid.pattern("stripe", x_hex, y_hex,
             colour="black", fill=c("yellow", "blue"), density = 0.5)

# Can alternatively use "gpar()" to specify colour and line attributes
grid::grid.newpage()
grid.pattern("stripe", x_hex, y_hex,
             gp = grid::gpar(col="blue", fill="red", lwd=2))

# 'weave' pattern
grid::grid.newpage()
grid.pattern("weave", x_hex, y_hex, type = "satin",
             colour = "black", fill = "lightblue", fill2 =  "yellow",
             density = 0.3)

# 'regular_polygon' pattern
grid::grid.newpage()
```

```
grid.pattern_regular_polygon(x_hex, y_hex, colour = "black",
                             fill = c("blue", "yellow", "red"),
                             shape = c("convex4", "star8", "circle"),
                             density = c(0.45, 0.42, 0.4),
                             spacing = 0.08, angle = 0)

# can be used to achieve a variety of 'tiling' effects
grid::grid.newpage()
grid.pattern_regular_polygon(x_hex, y_hex, color = "transparent",
                             fill = c("white", "grey", "black"),
                             density = 1.0, spacing = 0.1,
                             shape = "convex6", grid = "hex")
if (suppressPackageStartupMessages(requireNamespace("magick", quietly = TRUE))) {
  # array-based patterns
  # 'image' pattern
  logo_filename <- system.file("img", "Rlogo.png" , package="png")
  grid::grid.newpage()
  grid.pattern("image", x_hex, y_hex, filename=logo_filename, type="fit")
}
if (suppressPackageStartupMessages(requireNamespace("magick", quietly = TRUE))) {
  # 'plasma' pattern
  grid::grid.newpage()
  grid.pattern("plasma", x_hex, y_hex, fill="green")
}
```

`grid.pattern_ambient` *Ambient patterned grobs*

Description

`grid.pattern_ambient()` draws noise patterns onto the graphic device powered by the `ambient` package.

Usage

```
grid.pattern_ambient(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  type = "simplex",
  fill = gp$fill %||% "grey80",
  fill2 = "#4169E1",
  frequency = 0.01,
  interpolator = "quintic",
  fractal = switch(type, worley = "none", "fbm"),
  octaves = 3,
  lacunarity = 2,
```

```
  gain = 0.5,
  pertubation = "none",
  pertubation_amplitude = 1,
  value = "cell",
  distance_ind = c(1, 2),
  jitter = 0.45,
  res = getOption("ggpattern_res", 72),
  alpha = NA_real_,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
...	Currently ignored.
type	Either cubic, perlin, simplex, value, white, or worley
fill	Colour.
fill2	Second colour.
frequency	Determines the granularity of the features in the noise.
interpolator	How should values between sampled points be calculated? Either `'linear'`, `'hermite'`, or `'quintic'` (default), ranging from lowest to highest quality.
fractal	The fractal type to use. Either `'none'`, `'fbm'` (default), `'billow'`, or `'rigid-multi'`. It is suggested that you experiment with the different types to get a feel for how they behaves.
octaves	The number of noise layers used to create the fractal noise. Ignored if `fractal = 'none'`. Defaults to `3`.
lacunarity	The frequency multiplier between successive noise layers when building fractal noise. Ignored if `fractal = 'none'`. Defaults to `2`.
gain	The relative strength between successive noise layers when building fractal noise. Ignored if `fractal = 'none'`. Defaults to `0.5`.
pertubation	The pertubation to use. Either `'none'` (default), `'normal'`, or `'fractal'`. Defines the displacement (warping) of the noise, with `'normal'` giving a smooth warping and `'fractal'` giving a more eratic warping.
pertubation_amplitude	The maximal pertubation distance from the origin. Ignored if `pertubation = 'none'`. Defaults to `1`.
value	The noise value to return. Either

- 'value' (default) A random value associated with the closest point
- 'distance' The distance to the closest point
- 'distance2' The distance to the nth closest point (n given by distance_ind[1])
- 'distance2add' Addition of the distance to the nth and mth closest point given in distance_ind
- 'distance2sub' Substraction of the distance to the nth and mth closest point given in distance_ind
- 'distance2mul' Multiplication of the distance to the nth and mth closest point given in distance_ind
- 'distance2div' Division of the distance to the nth and mth closest point given in distance_ind

distance_ind	Reference to the nth and mth closest points that should be used when calculating value.
jitter	The maximum distance a point can move from its start position during sampling of cell points.
res	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

For more information about the noise types please see the relevant ambient documentation: ambient::noise_cubic(), ambient::noise_perlin(), ambient::noise_simplex(), ambient::noise_value(), ambient::noise_white(), and ambient::noise_worley(). grid.pattern_plasma() provides an alternative noise pattern that depends on magick.

Examples

```
if (requireNamespace("ambient", quietly = TRUE)) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_ambient(x_hex, y_hex, fill = "green", fill2 = "blue")
}
if (requireNamespace("ambient")) {
  grid::grid.newpage()
  grid.pattern_ambient(x_hex, y_hex, fill = "green", fill2 = "blue", type = "cubic")
}
```

`grid.pattern_aRtsy` *Grobs with patterns powered by the aRtsy package*

Description

`grid.pattern_aRtsy()` draws patterns powered by the `aRtsy` package. `names_aRtsy()` returns character vector of supported types.

Usage

```
grid.pattern_aRtsy(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  type = "strokes",
  fill = gp$fill %||% "grey80",
  alpha = gp$alpha %||% NA_real_,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_aRtsy()
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored
`type`	Name of pattern.
`fill`	Passed to the underlying `aRtsy` function's `color` / `colors` argument.
`alpha`	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
`default.units`	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
`name`	A character identifier.
`gp`	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
`draw`	A logical value indicating whether graphics output should be produced.
`vp`	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

https://koenderks.github.io/aRtsy/ for more information about the `aRtsy` package.

Examples

```
if (requireNamespace("aRtsy", quietly = TRUE)) {
  print(names_aRtsy())
}

# Make take more than 5 seconds on CRAN servers
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
if (requireNamespace("aRtsy", quietly = TRUE) &&
    guess_has_R4.1_features("patterns")) {
  grid::grid.newpage()
  grid.pattern_aRtsy(x_hex, y_hex, type = "forest",
                     fill = c("black", "white", "grey"))
}
```

---

grid.pattern_circle *Circle patterned grobs*

Description

`grid.pattern_circle()` draws a circle pattern onto the graphic device.

Usage

```
grid.pattern_circle(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
```

```
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  grid = "square",
  type = NULL,
  subtype = NULL,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
colour	Stroke colour(s).
fill	Fill colour(s) or grid::pattern() / gradient object(s).
angle	Rotation angle in degrees.
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern (in units units).
xoffset	Shift pattern along x axis (in units units).
yoffset	Shift pattern along y axis (in units units).
units	grid::unit() units for spacing, xoffset, and yoffset parameters.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set linewidth.
grid	Adjusts placement and density of certain graphical elements. "square" (default) is a square grid. "hex" is a hexagonal grid suitable for hexagonal and triangular tiling. "hex_circle" is a hexagonal grid suitable for circle packing. "elongated_triangle" is a grid used for the "elongated triangle" tiling.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in names_hex, names_square, or names_weave. See for pattern_hex(), pattern_square(), and pattern_weave() for more information about supported type arguments.
subtype	See for pattern_hex(), pattern_square(), and pattern_weave() for more information about supported subtype arguments.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.

gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

See grid.pattern_regular_polygon() for a more general case of this pattern.

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
grid.pattern_circle(x_hex, y_hex, fill = c("blue", "yellow"), density = 0.5)
grid::grid.newpage()
grid.pattern_circle(x_hex, y_hex, density = 0.8, grid = "hex_circle",
                    gp = grid::gpar(fill = c("blue", "yellow", "red")))
grid::grid.newpage()
grid.pattern_circle(x_hex, y_hex, density = 1.2, grid = "hex_circle",
                    gp = grid::gpar(fill = c("blue", "yellow", "red")))
# using a "twill_zigzag" 'weave' pattern
grid::grid.newpage()
grid.pattern_circle(x_hex, y_hex, fill = "blue", density = 0.5, type = "twill_zigzag")
```

---

grid.pattern_crosshatch

Crosshatch patterned grobs

Description

grid.pattern_crosshatch() draws a crosshatch pattern onto the graphic device.

Usage

```
grid.pattern_crosshatch(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  fill2 = fill,
  angle = 30,
  density = 0.2,
```

```
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  grid = "square",
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
colour	Stroke colour(s).
fill	Fill colour(s) or grid::pattern() / gradient object(s).
fill2	The fill colour for the "top" crosshatch lines.
angle	Rotation angle in degrees.
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern (in units units).
xoffset	Shift pattern along x axis (in units units).
yoffset	Shift pattern along y axis (in units units).
units	grid::unit() units for spacing, xoffset, and yoffset parameters.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set linewidth.
grid	Adjusts placement and density of certain graphical elements. "square" (default) is a square grid. "hex" is a hexagonal grid suitable for hexagonal and triangular tiling. "hex_circle" is a hexagonal grid suitable for circle packing. "elongated_triangle" is a grid used for the "elongated triangle" tiling.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.

gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid.pattern_weave() which interweaves two sets of lines. For a single set of lines use grid.pattern_stripe().

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
grid.pattern_crosshatch(x_hex, y_hex, colour = "black", fill = "blue",
                        fill2 = "yellow", density = 0.5)
grid::grid.newpage()
grid.pattern_crosshatch(x_hex, y_hex, density = 0.3,
                        gp = grid::gpar(col = "blue", fill = "yellow"))
```

---

grid.pattern_fill *Grobs with a simple fill pattern*

Description

grid.pattern_fill() draws a simple fill pattern onto the graphics device.

Usage

```
grid.pattern_fill(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  fill = gp$fill %||% "grey80",
  alpha = gp$alpha %||% NA_real_,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored
fill	Fill colour(s) or grid::pattern() / gradient object(s).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid::grid.polygon()

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
grid.pattern_fill(x_hex, y_hex, fill = "blue")

if (guess_has_R4.1_features("patterns")) {
  grid::grid.newpage()
  stripe_fill <- patternFill("stripe", fill = c("red", "blue"))
  grid.pattern_fill(x_hex, y_hex, fill = stripe_fill)
}
```

---

grid.pattern_gradient *Gradient patterned grobs*

Description

grid.pattern_gradient() draws a gradient pattern onto the graphic device.

Usage

```
grid.pattern_gradient(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  fill = gp$fill %||% "grey80",
  fill2 = "#4169E1",
  orientation = "vertical",
  alpha = gp$alpha %||% NA_real_,
  use_R4.1_gradients = getOption("ggpattern_use_R4.1_gradients",
    getOption("ggpattern_use_R4.1_features")),
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
fill	Colour.
fill2	Second colour.
orientation	vertical, horizontal, or radial.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
use_R4.1_gradients	Whether to use the gradient feature introduced in R v4.1 or use a rasterGrob approximation. Note not all graphic devices support the grid gradient feature.
aspect_ratio	Override aspect ratio.
key_scale_factor	Additional scale factor for legend.
res	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.

gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

Examples

```
if (requireNamespace("magick") && capabilities("png")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_gradient(x_hex, y_hex, fill = "green")
}
if (requireNamespace("magick") && capabilities("png")) {
  grid::grid.newpage()
  grid.pattern_gradient(x_hex, y_hex, fill = "green", orientation = "radial")
}
```

---

grid.pattern_hatch *Heraldic color hatching patterned grobs*

Description

grid.pattern_hatch() draws a heraldic color hatching patterns onto the graphic device. names_hatch() returns a character vector of supported type values.

Usage

```
grid.pattern_hatch(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  type = "gules",
  subtype = "combinatorial",
  colour = gp$col %||% "grey20",
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
  alpha = gp$alpha %||% NA_real_,
  lineend = gp$lineend %||% "round",
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
```

```
  use_R4.1_masks = getOption("ggpattern_use_R4.1_masks",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_hatch(
  subtype = c("combinatorial", "fox-davies", "goodman", "unicode"),
  accent = FALSE
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
type	A tincture or color name. names_hatch() lists supported values. Both traditional tincture names (e.g. "gules") and modern color equivalents (e.g. "red") are accepted. Matching is case-insensitive and ignores hyphens and spaces.
subtype	A string with one of • "combinatorial" (default): an extension of the seven standard Petra Sancta hatchings with systematically derived mixed-color hatchings. • "fox-davies": the hatchings in Fox-Davies' *A Complete Guide to Heraldry*. • "goodman": the hatchings in David Goodman's *Heraldic Tincture* reference. • "unicode": the hatchings used in Unicode character charts. The string is case-insensitive and hyphens and spaces are ignored.
colour	Stroke colour(s).
spacing	Spacing between repetitions of pattern (in units units).
xoffset	Shift pattern along x axis (in units units).
yoffset	Shift pattern along y axis (in units units).
units	grid::unit() units for spacing, xoffset, and yoffset parameters.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
lineend	Line end style, one of "round" (default), "butt", or "square".
linewidth	Stroke linewidth.

size	For backwards compatibility can be used to set linewidth.
use_R4.1_masks	If TRUE use the grid mask feature introduced in R v4.1.0. If FALSE do a rasterGrob approximation. If NULL try to guess an appropriate choice. Note not all graphic devices support the grid mask feature.
png_device	"png" graphics device to save intermediate raster data with if use_R4.1_masks is FALSE. If NULL and suggested package ragg is available and versions are high enough we directly capture masked raster via ragg::agg_capture(). Otherwise we will use png_device (default ragg::agg_png() if available else grDevices::png()) and png::readPNG() to manually compute a masked raster.
res	Resolution of desired rasterGrob in pixels per inch if use_R4.1_masks is FALSE.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).
accent	If TRUE, return tincture names using their traditional accented spellings where applicable (e.g. "tenn\u00e9", "brun\u00e2tre"). Defaults to FALSE.

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid.pattern_line() for single-direction lines, grid.pattern_crosshatch() for perpendicular lines. vignette("hatching", package = "gridpattern") for a visual overview of all subtypes. https://en.wikisource.org/wiki/A_Complete_Guide_to_Heraldry/Chapter_7#74 for Fox-Davies' *A Complete Guide to Heraldry*. https://david.goodman.graphics/portfolio/item/crests-heraldry-and-coats-of-arms/ for David Goodman's Heraldic Tincture reference.

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  grid::grid.newpage()
  grid.pattern_hatch(x_hex, y_hex, type = "azure", colour = "blue")
}
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  grid::grid.newpage()
  grid.pattern_hatch(x_hex, y_hex, type = "cendree", colour = "grey", spacing = 0.1)
}
print(names_hatch())
print(names_hatch("fox-davies"))
print(names_hatch("goodman"))
print(names_hatch("unicode"))
```

`grid.pattern_image` *Image patterned grobs*

Description

`grid.pattern_image()` draws an image pattern onto the graphic device.

Usage

```
grid.pattern_image(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  filename = "",
  type = "fit",
  scale = 1,
  gravity = switch(type, tile = "southwest", "center"),
  filter = "lanczos",
  alpha = gp$alpha %||% NA_real_,
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored.
`filename`	Image of filename or URL
`type`	Image scaling type
`scale`	Extra scaling
`gravity`	Position of image within area. `magick::gravity_types()` returns a vector of supported values.
`filter`	Filter to use when scaling. `magick::filter_types()` returns a vector of supported values.

alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
aspect_ratio	Override aspect ratio.
key_scale_factor	Additional scale factor for legend.
res	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Details

Here is a description of the type arguments:

expand Scale the image beyond the bounding box and crop it such that the image fully covers the width and the height of the region.

fit Scale the image such that either the width or the height of the image fits in the bounding box. Affected by gravity

none Position a single image in the region without attempting to scale to the bounding box size. Affected by scale and gravity.

squish Distort the image to cover the bounding box of the region.

tile Repeat the image to cover the bounding box. Affected by tile.

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid.pattern_placeholder() is an image pattern that uses images downloaded from the internet.
reset_image_cache() resets the image cache used by grid.pattern_image() and grid.pattern_placeholder().

Examples

```
# May emit a "CPU time > 2.5 times elapsed time" NOTE in a CRAN check
if (requireNamespace("magick")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  logo_filename <- system.file("img", "Rlogo.png" , package = "png")
  grid.pattern_image(x_hex, y_hex, filename = logo_filename, type = "fit")
}
if (requireNamespace("magick")) {
  # "tile" `type` image pattern depends on `magick` functionality
```

```
  # which is not reliable across platforms
  grid::grid.newpage()
  try(grid.pattern_image(x_hex, y_hex, filename = logo_filename,
                         type = "tile"))
}
```

`grid.pattern_line` *Line patterned grobs*

Description

`grid.pattern_line()` draws a line pattern onto the graphic device. Unlike `grid.pattern_stripe()` which fills bands with solid colour, this pattern draws stroked lines using the device's native line rendering, enabling all of R's built-in `linetype` values (including `"dotdash"`, `"twodash"`, and custom line types specified as hex strings per `?par`).

Usage

```
grid.pattern_line(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  angle = 30,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
  alpha = gp$alpha %||% NA_real_,
  lineend = gp$lineend %||% "round",
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  stagger = FALSE,
  use_R4.1_masks = getOption("ggpattern_use_R4.1_masks",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
colour	Stroke colour(s).
angle	Rotation angle in degrees.
spacing	Spacing between repetitions of pattern (in units units).
xoffset	Shift pattern along x axis (in units units).
yoffset	Shift pattern along y axis (in units units).
units	grid::unit() units for spacing, xoffset, and yoffset parameters.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
lineend	Line end style, one of "round" (default), "butt", or "square".
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set linewidth.
stagger	If TRUE, alternate lines are shifted by half the dash period so that dashes of adjacent lines interleave. Computed from linetype and linewidth per ?par. Default FALSE.
use_R4.1_masks	If TRUE use the grid mask feature introduced in R v4.1.0. If FALSE do a rasterGrob approximation. If NULL try to guess an appropriate choice. Note not all graphic devices support the grid mask feature.
png_device	"png" graphics device to save intermediate raster data with if use_R4.1_masks is FALSE. If NULL and suggested package ragg is available and versions are high enough we directly capture masked raster via ragg::agg_capture(). Otherwise we will use png_device (default ragg::agg_png() if available else grDevices::png()) and png::readPNG() to manually compute a masked raster.
res	Resolution of desired rasterGrob in pixels per inch if use_R4.1_masks is FALSE.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid.pattern_stripe() for filled bands, grid.pattern_wave() for wavy bands, grid.pattern_crosshatch() for two sets of lines.

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  grid::grid.newpage()
  grid.pattern_line(x_hex, y_hex, colour = "black", linetype = "dotdash",
                    angle = 45, spacing = 0.1, stagger = TRUE)
}

# more intricate dashed lines are possible with hex strings
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  grid::grid.newpage()
  grid.pattern_line(x_hex, y_hex, gp = grid::gpar(col = "darkred", lty = "23632E"))
}
```

---

grid.pattern_magick *Magick patterned grobs*

Description

grid.pattern_magick() draws a imagemagick pattern onto the graphic device. names_magick, names_magick_intensity, and names_magick_stripe are character vectors of supported type values plus subsets for shaded intensity and stripes.

Usage

```
grid.pattern_magick(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  type = "hexagons",
  fill = "grey20",
  scale = 1,
  filter = "box",
  alpha = gp$alpha %||% NA_real_,
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
```

```
  vp = NULL
)

names_magick

names_magick_intensity

names_magick_stripe
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
type	Magick pattern types. names_magick, names_magick_intensity, and names_magick_stripe are character vectors of supported type values plus subsets for shaded intensity and stripes.
fill	Fill colour
scale	Extra scaling
filter	Filter to use when scaling. magick::filter_types() returns a vector of supported values.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
aspect_ratio	Override aspect ratio.
key_scale_factor	Additional scale factor for legend.
res	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

The imagemagick documentation https://imagemagick.org/script/formats.php/ for more information.

Examples

```
if (requireNamespace("magick")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_magick(x_hex, y_hex, type="octagons", fill="blue", scale=2)
}

# supported magick pattern names
print(names_magick)
```

---

grid.pattern_none *Grobs without any pattern*

Description

`grid.pattern_none()` draws nothing onto the graphic device.

Usage

```
grid.pattern_none(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored
`default.units`	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
`name`	A character identifier.
`gp`	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
`draw`	A logical value indicating whether graphics output should be produced.
`vp`	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

`grid::grid.null()`

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
grid.pattern_none(x_hex, y_hex)
```

---

grid.pattern_pch *Plotting character patterned grobs*

Description

`grid.pattern_pch()` draws a plotting character pattern onto the graphic device.

Usage

```
grid.pattern_pch(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
  scale = 0.5,
  shape = 1L,
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  default.units = "npc",
  name = NULL,
```

```
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
colour	Stroke colour(s).
fill	Fill colour(s) or grid::pattern() / gradient object(s).
angle	Rotation angle in degrees.
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern (in units units).
xoffset	Shift pattern along x axis (in units units).
yoffset	Shift pattern along y axis (in units units).
units	grid::unit() units for spacing, xoffset, and yoffset parameters.
scale	For star polygons, multiplier (between 0 and 1) applied to exterior radius to get interior radius.
shape	An integer from 0 to 25 or NA. See graphics::points() for more details. Note we only support these shapes and do not support arbitrary ASCII / Unicode characters.
grid	Adjusts placement and density of certain graphical elements. "square" (default) is a square grid. "hex" is a hexagonal grid suitable for hexagonal and triangular tiling. "hex_circle" is a hexagonal grid suitable for circle packing. "elongated_triangle" is a grid used for the "elongated triangle" tiling.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in names_hex, names_square, or names_weave. See for pattern_hex(), pattern_square(), and pattern_weave() for more information about supported type arguments.
subtype	See for pattern_hex(), pattern_square(), and pattern_weave() for more information about supported subtype arguments.
rot	Angle to rotate regular polygon (degrees, counter-clockwise).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set linewidth.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.

gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid.pattern_regular_polygon() which is used to implement this pattern.

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
gp <- grid::gpar(col = "black", fill = "lightblue")

if (capabilities("png") || guess_has_R4.1_features("masks")) {
  # pch 0-6 are simple shapes with no fill
  grid.pattern_pch(x_hex, y_hex, shape = 0:6, gp = gp,
                   spacing = 0.1, density = 0.4, angle = 0)
}
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  # pch 7-14 are compound shapes with no fill
  grid::grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = 7:14, gp = gp,
                   spacing = 0.1, density = 0.4, angle = 0)
}
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  # pch 15-20 are filled with 'col'
  grid::grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = 15:20, gp = gp,
                   spacing = 0.1, density = 0.4, angle = 0)
}
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  # pch 21-25 are filled with 'fill'
  grid::grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = 21:25, gp = gp,
                   spacing = 0.1, density = 0.4, angle = 0)
}
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  # using a 'basket' weave `type` with two shapes
  grid::grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = c(1,4), gp = gp,
                   type = "basket",
                   spacing = 0.1, density = 0.4, angle = 0)
}
```

grid.pattern_placeholder *Placeholder image patterned grobs*

Description

`grid.pattern_placeholder()` draws a placeholder image pattern onto the graphic device. `names_placeholder` are character vectors of supported placeholder types.

Usage

```
grid.pattern_placeholder(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  type = "bear",
  alpha = gp$alpha %||% NA_real_,
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_placeholder
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored.
`type`	Image source. `names_placeholder` is a vector of supported values. If you would like only greyscale images append `bw` to the name.
`alpha`	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
`aspect_ratio`	Override aspect ratio.
`key_scale_factor`	Additional scale factor for legend.
`res`	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.

default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

reset_image_cache() resets the image cache used by grid.pattern_image() and grid.pattern_placeholder().

Examples

```
if (requireNamespace("magick")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  # requires internet connection to download from placeholder image websites
  try(grid.pattern_placeholder(x_hex, y_hex, type="bear"))
}

print(names_placeholder)
```

---

grid.pattern_plasma *Plasma patterned grobs*

Description

grid.pattern_plasma() draws a plasma pattern onto the graphic device.

Usage

```
grid.pattern_plasma(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  fill = gp$fill %||% "grey80",
  scale = 1,
  alpha = gp$alpha %||% NA_real_,
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
```

```
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
fill	Colour.
scale	Extra scaling
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
aspect_ratio	Override aspect ratio.
key_scale_factor	Additional scale factor for legend.
res	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid.pattern_ambient() provides a noise pattern using the ambient package. Pseudorandom seeds for the plasma pattern may be set via magick::magick_set_seed().

Examples

```
if (requireNamespace("magick")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_plasma(x_hex, y_hex, fill = "green")
}
```

grid.pattern_polygon_tiling *Polygon tiling patterned grobs*

Description

grid.pattern_polygon_tiling() draws a specified polygon tiling pattern onto the graphic device. names_polygon_tiling lists all supported types.

Usage

```
grid.pattern_polygon_tiling(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
  type = "square",
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_polygon_tiling
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
colour	Stroke colour(s).

fill	Fill colour(s) or grid::pattern() / gradient object(s).
angle	Rotation angle in degrees.
spacing	Spacing between repetitions of pattern (in units units).
xoffset	Shift pattern along x axis (in units units).
yoffset	Shift pattern along y axis (in units units).
units	grid::unit() units for spacing, xoffset, and yoffset parameters.
type	Name of polygon tiling to draw. See Details.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value). Not supported for all polygon tiling type.
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set linewidth.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Details

grid.pattern_polygon_tiling() supports 1, 2, or 3 fill colors with the first colors (weakly) covering a larger area. Size of the pattern is controlled by spacing. We support the following polygon tiling types:

elongated_triangular Creates an elongated triangular tiling made of squares and triangles.

herringbone Creates a herringbone tiling made of rectangles.

hexagonal Creates a hexagonal tiling made of hexagons.

pythagorean Creates a Pythagorean tiling made of squares of two different sizes.

rhombille Creates a rhombille tiling made of rhombi.

rhombitrihexagonal Creates a rhombitrihexagonal tiling made out of dodecagons, hexagons, and squares.

snub_square Creates a snub square tiling made of squares and triangles.

snub_trihexagonal Creates a snub trihexagonal tiling made of hexagons and triangles.

square Creates a square tiling made of squares.

tetrakis_square Creates a tetrakis square tiling made of isosceles right triangles.

triangular Creates a triangular tiling made of equilateral triangles.

trihexagonal Creates a trihexagonal tiling made of hexagons and triangles.

truncated_square Creates a truncated square tiling made of octagons and squares.

`truncated_hexagonal` Creates a truncated hexagonal tiling made of dodecagons and triangles.

`truncated_trihexagonal` Creates a truncated trihexagonal tiling made of hexagons, squares, and triangles.

`2*.2**.2*.2**` Creates a polygon tiling made of rhombi.

`2**.3**.12*` Creates a polygon tiling made of rhombi, triangles, and twelve-pointed stars.

`3.3.3.3**` Creates a polygon tiling made of triangles.

`3.3*.3.3**` Creates a regular (star) polygon tiling made of triangles and three-pointed stars.

`3.3.3.12*.3.3.12*` Creates a regular (star) polygon tiling made of triangles and twelve-pointed stars.

`3.3.8*.3.4.3.8*` Creates a regular (star) polygon tiling made of triangles, squares, and eight-pointed stars.

`3.3.8*.4**.8*` Creates a regular (star) polygon tiling made of triangles, four-pointed stars, and eight-pointed stars.

`3.4.6.3.12*` Creates a regular (star) polygon tiling made of triangles, squares, hexagons, and twelve-pointed stars.

`3.4.8.3.8*` Creates a regular (star) polygon tiling made of triangles, squares, octagons, and eight-pointed stars.

`3.6*.6**` Creates a regular (star) polygon tiling made of triangles and six-pointed stars.

`4.2*.4.2**` Creates a polygon tiling made of squares and rhombi.

`4.4*.4**` Creates a regular (star) polygon tiling made of squares and four-pointed stars.

`4.6.4*.6` Creates a regular (star) polygon tiling made of squares, hexagons, and four-pointed stars.

`4.6*.4.6*.4.6*` Creates a regular (star) polygon tiling made of squares and six-pointed stars.

`4.8*.4**.8*` Creates a polygon tiling of squares and eight-pointed stars.

`6.6*.6.6*` Creates a regular (star) polygon tiling made of hexagons and six-pointed stars.

`8.4*.8.4*` Creates a regular (star) polygon tiling made of octagons and four-pointed stars.

`9.3.9.3*` Creates a regular (star) polygon tiling made of triangles, nonagons, and three-pointed stars.

`12.3*.12.3*` Creates a regular (star) polygon tiling made of dodecagons and three-pointed stars.

`12.12.4*` Creates a regular (star) polygon tiling made of dodecagons and four-pointed stars.

`18.18.3*` Creates a regular (star) polygon tiling made of eighteen-sided polygons and three-pointed stars.

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

The tiling vignette `vignette("tiling", package = "gridpattern")` for more information about these tilings as well as more examples of polygon tiling using the `grid.pattern_regular_polygon()` function.

Examples

```
print(names_polygon_tiling)

x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
gp1 <- grid::gpar(fill = "yellow", col = "black")
gp2 <- grid::gpar(fill = c("yellow", "red"), col = "black")
gp3 <- grid::gpar(fill = c("yellow", "red", "blue"), col = "black")

grid.pattern_polygon_tiling(x_hex, y_hex, type = "herringbone", gp = gp1)

grid::grid.newpage()
grid.pattern_polygon_tiling(x_hex, y_hex, type = "hexagonal",
                            spacing = 0.2, gp = gp3)

grid::grid.newpage()
grid.pattern_polygon_tiling(x_hex, y_hex, type = "pythagorean",
                            spacing = 0.2, gp = gp2)

grid::grid.newpage()
grid.pattern_polygon_tiling(x_hex, y_hex, type = "snub_trihexagonal",
                            spacing = 0.2, gp = gp3)

grid::grid.newpage()
grid.pattern_polygon_tiling(x_hex, y_hex, type = "rhombille",
                              spacing = 0.2, gp = gp3)
```

---

grid.pattern_regular_polygon

Regular polygon patterned grobs

Description

`grid.pattern_regular_polygon()` draws a regular polygon pattern onto the graphic device.

Usage

```
grid.pattern_regular_polygon(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
```

```
  yoffset = 0,
  units = "snpc",
  scale = 0.5,
  shape = "convex4",
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
...	Currently ignored.
colour	Stroke colour(s).
fill	Fill colour(s) or `grid::pattern()` / gradient object(s).
angle	Rotation angle in degrees.
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern (in `units` units).
xoffset	Shift pattern along x axis (in `units` units).
yoffset	Shift pattern along y axis (in `units` units).
units	`grid::unit()` units for `spacing`, `xoffset`, and `yoffset` parameters.
scale	For star polygons, multiplier (between 0 and 1) applied to exterior radius to get interior radius.
shape	Either "convex" or "star" followed by the number of exterior vertices or alternatively "circle", "square", "null", "rhombille_rhombus", "tetrakis_left", or "tetrakis_right". For example "convex5" corresponds to a pentagon and "star6" corresponds to a six-pointed star. The "square" shape is larger than the "convex4" shape and is rotated an extra 45 degrees, it can be used to generate a multi-colored “checkers” effect when density is 1. The "null" shape is not drawn, it can be used to create holes within multiple-element patterns. The "rhombille_rhombus" shape draws a rhombus while the "tetrakis_left" or "tetrakis_right" shapes draw an isosceles right triangle. These latter three non-regular-polygon shapes are intended to help generate rhombille and tetrakis square tilings.

grid	Adjusts placement and density of certain graphical elements. "square" (default) is a square grid. "hex" is a hexagonal grid suitable for hexagonal and triangular tiling. "hex_circle" is a hexagonal grid suitable for circle packing. "elongated_triangle" is a grid used for the "elongated triangle" tiling.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in names_hex, names_square, or names_weave. See for pattern_hex(), pattern_square(), and pattern_weave() for more information about supported type arguments.
subtype	See for pattern_hex(), pattern_square(), and pattern_weave() for more information about supported subtype arguments.
rot	Angle to rotate regular polygon (degrees, counter-clockwise).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set linewidth.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid.pattern_circle() for a special case of this pattern. The tiling vignette features more examples of regular polygon tiling using this function vignette("tiling", package = "gridpattern").

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))

# 'density', 'rot', and 'shape' are vectorized
grid.pattern_regular_polygon(x_hex, y_hex, colour = "black",
                             fill = c("blue", "yellow", "red"),
                             shape = c("convex4", "star8", "circle"),
                             density = c(0.45, 0.42, 0.4),
                             spacing = 0.08, angle = 0)

# checker pattern using "square" shape
grid::grid.newpage()
grid.pattern_regular_polygon(x_hex, y_hex, shape = "square",
                             colour = "transparent",
```

```
                              fill = c("black", "red", "blue", "yellow"),
                              angle = 0, density = 1.0, spacing = 0.2)

# checker pattern using the default "convex4" shape
grid::grid.newpage()
grid.pattern_regular_polygon(x_hex, y_hex, density = 1.0,
                              colour = "black", fill = "blue")

# using a "twill_zigzag" 'weave' pattern
grid::grid.newpage()
grid.pattern_regular_polygon(x_hex, y_hex, fill = c("blue", "yellow"),
                              shape = c("circle", "star8"),
                              density = c(0.5, 0.6), type = "twill_zigzag")

# hexagon tiling
grid::grid.newpage()
grid.pattern_regular_polygon(x_hex, y_hex, color = "transparent",
                              fill = c("white", "grey", "black"),
                              density = 1.0, spacing = 0.1,
                              shape = "convex6", grid = "hex")

# triangle tiling
grid::grid.newpage()
grid.pattern_regular_polygon(x_hex, y_hex, fill = "green",
                              density = 1.0, spacing = 0.1,
                              shape = "convex3", grid = "hex")
```

`grid.pattern_rose` *Rose curve patterned grobs*

Description

`grid.pattern_rose()` draws a rose curve pattern onto the graphic device.

Usage

```
grid.pattern_rose(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
```

```
  frequency = 0.1,
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  use_R4.1_masks = getOption("ggpattern_use_R4.1_masks",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
...	Currently ignored.
colour	Stroke colour(s).
fill	Fill colour(s) or `grid::pattern()` / gradient object(s).
angle	Rotation angle in degrees.
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern (in `units` units).
xoffset	Shift pattern along x axis (in `units` units).
yoffset	Shift pattern along y axis (in `units` units).
units	`grid::unit()` units for `spacing`, `xoffset`, and `yoffset` parameters.
frequency	The "angular frequency" parameter of the rose pattern.
grid	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in `names_hex`, `names_square`, or `names_weave`. See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `type` arguments.
subtype	See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `subtype` arguments.

rot	Angle to rotate rose (degrees, counter-clockwise).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set linewidth.
use_R4.1_masks	If TRUE use the grid mask feature introduced in R v4.1.0. If FALSE do a rasterGrob approximation. If NULL try to guess an appropriate choice. Note not all graphic devices support the grid mask feature.
png_device	"png" graphics device to save intermediate raster data with if use_R4.1_masks is FALSE. If NULL and suggested package ragg is available and versions are high enough we directly capture masked raster via ragg::agg_capture(). Otherwise we will use png_device (default ragg::agg_png() if available else grDevices::png()) and png::readPNG() to manually compute a masked raster.
res	Resolution of desired rasterGrob in pixels per inch if use_R4.1_masks is FALSE.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

See https://en.wikipedia.org/wiki/Rose_(mathematics) for more information.

Examples

```
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  gp <- grid::gpar(fill = c("blue", "red", "yellow", "green"), col = "black")
  grid.pattern_rose(x_hex, y_hex,
                    spacing = 0.15, density = 0.5, angle = 0,
                    frequency = 1:4, gp = gp)
}
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  grid::grid.newpage()
  grid.pattern_rose(x_hex, y_hex,
                    spacing = 0.15, density = 0.5, angle = 0,
                    frequency = 1/1:4, gp = gp)
}
```

```
if (capabilities("png") || guess_has_R4.1_features("masks")) {
  grid::grid.newpage()
  grid.pattern_rose(x_hex, y_hex,
                    spacing = 0.18, density = 0.5, angle = 0,
                    frequency = c(3/2, 7/3, 5/4, 3/7), gp = gp)
}
```

`grid.pattern_stripe` *Stripe patterned grobs*

Description

`grid.pattern_stripe()` draws a stripe pattern onto the graphic device.

Usage

```
grid.pattern_stripe(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  grid = "square",
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.

...	Currently ignored.
colour	Stroke colour(s).
fill	Fill colour(s) or grid::pattern() / gradient object(s).
angle	Rotation angle in degrees.
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern (in units units).
xoffset	Shift pattern along x axis (in units units).
yoffset	Shift pattern along y axis (in units units).
units	grid::unit() units for spacing, xoffset, and yoffset parameters.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set linewidth.
grid	Adjusts placement and density of certain graphical elements. "square" (default) is a square grid. "hex" is a hexagonal grid suitable for hexagonal and triangular tiling. "hex_circle" is a hexagonal grid suitable for circle packing. "elongated_triangle" is a grid used for the "elongated triangle" tiling.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

[grid.pattern_crosshatch()] and [grid.pattern_weave()] for overlaying stripes. Use grid.pattern_line() for stroked lines that support all native linetype values. Use grid.pattern_wave() for wavy bands instead of straight ones.

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
grid.pattern_stripe(x_hex, y_hex, colour = "black",
                    fill = c("red", "blue"), density = 0.4)

# Can alternatively use "gpar()" to specify colour and line attributes
grid::grid.newpage()
grid.pattern_stripe(x_hex, y_hex, density = 0.3,
                    gp = grid::gpar(col = "blue", fill = "yellow"))
```

`grid.pattern_text` *Text character patterned grobs*

Description

`grid.pattern_text()` draws a text character pattern onto the graphic device.

Usage

```
grid.pattern_text(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  angle = 30,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
  scale = 0.5,
  shape = "X",
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  size = gp$fontsize %||% 12,
  fontfamily = gp$fontfamily %||% "sans",
  fontface = gp$fontface %||% "plain",
  use_R4.1_masks = getOption("ggpattern_use_R4.1_masks",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.

...	Currently ignored.
colour	Stroke colour(s).
angle	Rotation angle in degrees.
spacing	Spacing between repetitions of pattern (in units units).
xoffset	Shift pattern along x axis (in units units).
yoffset	Shift pattern along y axis (in units units).
units	grid::unit() units for spacing, xoffset, and yoffset parameters.
scale	For star polygons, multiplier (between 0 and 1) applied to exterior radius to get interior radius.
shape	A character or expression vector. See label argument of grid::textGrob() for more details.
grid	Adjusts placement and density of certain graphical elements. "square" (default) is a square grid. "hex" is a hexagonal grid suitable for hexagonal and triangular tiling. "hex_circle" is a hexagonal grid suitable for circle packing. "elongated_triangle" is a grid used for the "elongated triangle" tiling.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in names_hex, names_square, or names_weave. See for pattern_hex(), pattern_square(), and pattern_weave() for more information about supported type arguments.
subtype	See for pattern_hex(), pattern_square(), and pattern_weave() for more information about supported subtype arguments.
rot	Angle to rotate regular polygon (degrees, counter-clockwise).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
size	Fontsize
fontfamily	The font family. See grid::gpar() for more details.
fontface	The font face. See grid::gpar() for more details.
use_R4.1_masks	If TRUE use the grid mask feature introduced in R v4.1.0. If FALSE do a rasterGrob approximation. If NULL try to guess an appropriate choice. Note not all graphic devices support the grid mask feature.
png_device	"png" graphics device to save intermediate raster data with if use_R4.1_masks is FALSE. If NULL and suggested package ragg is available and versions are high enough we directly capture masked raster via ragg::agg_capture(). Otherwise we will use png_device (default ragg::agg_png() if available else grDevices::png()) and png::readPNG() to manually compute a masked raster.
res	Resolution of desired rasterGrob in pixels per inch if use_R4.1_masks is FALSE.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

Examples

```
if (capabilities("png") &&
    gridpattern:::device_supports_unicode()) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))

  playing_card_symbols <- c("\u2660", "\u2665", "\u2666", "\u2663")
  grid.pattern_text(x_hex, y_hex,
                    shape = playing_card_symbols,
                    colour = c("black", "red", "red", "black"),
                    size = 18, spacing = 0.1, angle = 0)
}
```

---

grid.pattern_wave *Wave patterned grobs*

Description

`grid.pattern_wave()` draws a wave pattern onto the graphic device. `names_wave` is a character vector of supported `type` values.

Usage

```
grid.pattern_wave(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
  amplitude = 0.5 * spacing,
  frequency = 1/spacing,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  grid = "square",
  type = "indented",
```

```
  reverse = FALSE,
  stagger = FALSE,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_wave
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
...	Currently ignored.
colour	Stroke colour(s).
fill	Fill colour(s) or `grid::pattern()` / gradient object(s).
angle	Rotation angle in degrees.
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern (in `units` units).
xoffset	Shift pattern along x axis (in `units` units).
yoffset	Shift pattern along y axis (in `units` units).
units	`grid::unit()` units for `amplitude`, `frequency`, `spacing`, `xoffset`, and `yoffset` parameters.
amplitude	Wave amplitude (in `units` units)
frequency	Linear frequency (in inverse `units` units)
alpha	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set `linewidth`.
grid	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
type	One of the following (see `names_wave` for the canonical list): `"dovetailed"` A wave with diagonal strokes connecting the crests and troughs. Alias: `"dovetail"`. `"embattled"` Square wave. Alias: `"square"`. `"embattled_grady"` Graduated stepped wave: two ascending steps followed by two descending steps per period.

"engrailed" Repeating arches curving downward (non-positive half of a sine wave per period).

"indented" **(default)** Triangular wave with equal rise and fall. Alias: "triangle".

"invected" Repeating arches curving upward (non-negative half of a sine wave per period).

"nebuly" Smooth cloud-like wave.

"potenty" A stepped wave with T-shaped crenellations.

"raguly" Oblique stepped wave. Use reverse = TRUE for the horizontally mirror image.

"sawtoothed" Sawtooth wave with a gradual rise and sharp fall. Use reverse = TRUE for a sharp rise and gradual fall. Aliases: "sawlike", "sawtooth".

"urdy" A wave with pointed crests and troughs.

"wavy" Smooth sinusoidal wave. Aliases: "sine", "undy".

reverse	If TRUE, horizontally mirror the wave. Currently affects "sawtoothed" and "raguly" only. Default FALSE.
stagger	If TRUE, alternate wave rows are shifted by half a wavelength so that crests of one row align with troughs of adjacent rows, creating an interlocking effect. Default FALSE.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

Use grid.pattern_stripe() for straight filled bands or grid.pattern_line() for stroked lines instead of waves. See https://en.wikipedia.org/wiki/Line_(heraldry) and https://en.wikipedia.org/wiki/Waveform for more information about the supported wave types.

Examples

```
print(names_wave)

# visual table of all wave types
grid::grid.newpage()
n <- length(names_wave)
nc <- 2L
nr <- ceiling(n / nc)
grid::pushViewport(grid::viewport(layout = grid::grid.layout(nr, nc)))
```

```
for (i in seq_len(n)) {
    grid::pushViewport(grid::viewport(
        layout.pos.row = (i - 1L) %/% nc + 1L,
        layout.pos.col = (i - 1L) %% nc + 1L
    ))
    grid.pattern_wave(colour = "black", fill = c("gold", "steelblue"),
                      type = names_wave[i], density = 0.18, spacing = 0.45,
                      angle = 0, amplitude = 0.100, frequency = 1 / 0.45)
    grid::grid.rect(x = 0.5, y = 0.86, width = 0.5, height = 0.28,
                    just = "centre", gp = grid::gpar(fill = "grey80", col = "black"))
    grid::grid.text(names_wave[i], x = 0.5, y = 0.88,
                    gp = grid::gpar(fontsize = 11))
    grid::grid.rect(gp = grid::gpar(fill = "transparent", col = "black", lwd = 6))
    grid::popViewport()
}
grid::popViewport()

# stagger shifts alternate rows by half a wavelength
grid::grid.newpage()
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
grid.pattern_wave(x_hex, y_hex, colour = "black", type = "urdy",
                  fill = c("red", "blue"), density = 0.3,
                  spacing = 0.15, angle = 0,
                  amplitude = 0.045, frequency = 1 / 0.15, stagger = TRUE)
```

`grid.pattern_weave` *Weave patterned grobs*

Description

`grid.pattern_weave()` draws a weave pattern onto the graphic device.

Usage

```
grid.pattern_weave(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  fill2 = fill,
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  units = "snpc",
```

```
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  grid = "square",
  type = "plain",
  subtype = NA,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored.
colour	Stroke colour(s).
fill	The fill colour for the horizontal "weft" lines.
fill2	The fill colour for the vertical "warp" lines.
angle	Rotation angle in degrees.
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern (in units units).
xoffset	Shift pattern along x axis (in units units).
yoffset	Shift pattern along y axis (in units units).
units	grid::unit() units for spacing, xoffset, and yoffset parameters.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype.
linewidth	Stroke linewidth.
size	For backwards compatibility can be used to set linewidth.
grid	Adjusts placement and density of certain graphical elements. "square" (default) is a square grid. "hex" is a hexagonal grid suitable for hexagonal and triangular tiling. "hex_circle" is a hexagonal grid suitable for circle packing. "elongated_triangle" is a grid used for the "elongated triangle" tiling.
type	The weave type. See pattern_weave() for more details.
subtype	The weave subtype. See pattern_weave() for more details.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.

name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

pattern_weave()

Examples

```
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
gp <- grid::gpar(colour = "black", fill = "lightblue", lwd=0.5)

# Plain weave (default weave)
grid.pattern_weave(x_hex, y_hex, fill2 = "yellow",
                   gp = gp, spacing = 0.1, density = 0.3)

# Irregular matt weave
grid::grid.newpage()
grid.pattern_weave(x_hex, y_hex,  type = "matt_irregular",
                   fill2 = "yellow", gp = gp, spacing = 0.1, density = 0.3)

# Twill weave
grid::grid.newpage()
grid.pattern_weave(x_hex, y_hex, type = "twill",
                   fill2 = "yellow", gp = gp, spacing = 0.1, density = 0.3)

# Zig-zag twill
grid::grid.newpage()
grid.pattern_weave(x_hex, y_hex, type = "twill_zigzag",
                   fill2 = "yellow", gp = gp, spacing = 0.05, density = 0.7)

# Herringbone twill with density 1
grid::grid.newpage()
gp$col <- NA
grid.pattern_weave(x_hex, y_hex, type = "twill_herringbone",
                   fill2 = "yellow", gp = gp, spacing = 0.05, density = 1.0)
```

guess_has_R4.1_features *Guess whether "active" graphics device supports the grid graphics features introduced in R v4.1.*

Description

`guess_has_R4.1_features()` guesses whether "active" graphics device supports the grid graphics features introduced in R v4.1. If it guesses it does it returns `TRUE` else `FALSE`.

Usage

```
guess_has_R4.1_features(
  features = c("clippingPaths", "gradients", "masks", "patterns")
)
```

Arguments

`features`	Character vector of features to guess support for. Will return `TRUE` only if guesses support for all requested features. **"clippingPaths"** Supports clipping path feature **"gradients"** Supports (both linear and radial) gradient feature **"masks"** Supports (alpha) mask feature **"patterns"** Supports (tiling) pattern feature

Value

`TRUE` if we guess all `features` are supported else `FALSE`

Usage in other packages

To avoid taking a dependency on `gridpattern` you may copy the source of `guess_has_R4.1_features()` into your own package under the permissive MIT No Attribution (MIT-0) license. Either use `usethis::use_standalone("trevorld/gridpattern", "standalone-guess_has_R4.1_features.R")` or copy the file `standalone-guess_has_R4.1_features.R` into your `R` directory and add `grDevices` and `utils` to the `Imports` of your `DESCRIPTION` file.

See Also

https://www.stat.auckland.ac.nz/~paul/Reports/GraphicsEngine/definitions/definitions.html for more info about the new grid graphics features introduced in R v4.1.

Examples

```
# If R version (weakly) greater than 4.1 should be TRUE
pdf(tempfile(fileext = ".pdf"))
print(guess_has_R4.1_features())
invisible(dev.off())

# Should be FALSE
postscript(tempfile(fileext = ".ps"))
print(guess_has_R4.1_features())
invisible(dev.off())
```

---

local_pattern *Temporarily register a pattern*

Description

`local_pattern()` registers a pattern (see `register_pattern()`) until the current function (or test, see `.local_envir`) exits and then restores the previous registration (if any). Requires the suggested `{withr}` package.

Usage

```
local_pattern(
  name,
  fn,
  kind = c("geometry", "array"),
  defaults = list(),
  unprefixed = character(),
  env = caller_env(),
  .local_envir = parent.frame()
)
```

Arguments

`name`	Pattern name. Must not contain `":"`.
`fn`	Pattern function. Its required signature depends on `kind`:

geometry `function(params, boundary_df, aspect_ratio, legend)` returning a grid grob that lies within the boundary (e.g. clipped to it). `params` is a list of (prefixed) pattern parameters e.g. `params$pattern_fill`, `boundary_df` is a data frame with `x`, `y`, and `id` columns describing the boundary polygon(s) in "npc" coordinates, `aspect_ratio` is the (best guess) aspect ratio of the viewport, and `legend` is whether the pattern is being drawn in a `{ggpattern}` legend key.

array `function(width, height, params, legend)` returning a 3D array of RGBA values (all values in the range [0, 1]) which {gridpattern} will mask to the boundary. `width` and `height` are the dimensions (in pixels) of the boundary's bounding box and `params` and `legend` are as for geometry patterns.

See the "Developing Patterns" vignette `vignette("developing-patterns", package = "gridpattern")` for more details and examples.

`kind` Either `"geometry"` or `"array"`.

`defaults` A named list of pattern-specific default parameter values e.g. `list(pattern_type = "running", pattern_spacing = 0.1)`. Names are used as-is and must be `pattern_` prefixed unless listed in `unprefixed`. Values may be constants or functions of the form `function(params, gp)` where `params` is the list of pattern parameters filled in so far and `gp` is the grob's `grid::gpar()` object e.g. `list(pattern_amplitude = function(params, gp) 0.5 * params$pattern_spacing)` or `list(pattern_size = function(params, gp) gp$fontsize %||% 12)`. These replace the generic defaults for parameters that weren't supplied. The remaining generic defaults are filled in first and then these in the order given so a function default may use any generic parameter and any earlier pattern-specific default. Note {ggpattern} supplies its own (generic) defaults for most of its pattern aesthetics so these will mainly be used with `grid.pattern()` / `patternGrob()`.

`unprefixed` A character vector of parameter names without the `pattern_` prefix that the pattern reads so that `defaults` may set them. The usual case is a plain {ggplot2} aesthetic that {ggpattern} passes to pattern functions alongside the `pattern_` aesthetics e.g. `defaults = list(fill = "black"), unprefixed = "fill"` lets a pattern that reads `params$fill` under {ggpattern} also work with `grid.pattern()`. Since such patterns are rare any other unprefixed name (e.g. `fill` or `spacing` instead of `pattern_fill` or `pattern_spacing`) is an error of class `"gridpattern_error_unprefixed_de`

`env` The environment used to determine the "owner" of the pattern. If writing a function that registers patterns on behalf of its caller add an `env = rlang::caller_env()` argument to that function and pass it on.

`.local_envir` The environment whose exit should undo `local_pattern()`'s registration.

Value

`local_pattern()` invisibly returns the previous registration (a list) or `NULL` if there wasn't one.

See Also

`register_pattern()` to (permanently) register patterns.

Examples

```
create_pattern_polygon <- function(params, boundary_df, aspect_ratio, legend = FALSE) {
  grid::polygonGrob(boundary_df$x, boundary_df$y, boundary_df$id,
                    gp = grid::gpar(fill = params$pattern_fill))
}
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
```

```
draw_polygon <- function() {
  local_pattern("polygon", create_pattern_polygon)
  grid::grid.newpage()
  grid.pattern("polygon", x_hex, y_hex, fill = "red")
}
# don't clobber (or be confused by) any pre-existing "polygon" pattern
if (requireNamespace("withr", quietly = TRUE) && !has_pattern("polygon")) {
  draw_polygon()
}
print(has_pattern("polygon"))
```

`mean_col` *Compute average color*

Description

`mean_col()` computes an average color.

Usage

```
mean_col(...)
```

Arguments

`...`	Colors to average

Details

We currently compute an average color by using the quadratic mean of the colors' RGBA values.

Value

A color string of 9 characters: `"#"` followed by the red, blue, green, and alpha values in hexadecimal.

See Also

`mix_col()` for subtractive (pigment) mixing via Munsell color space.

Examples

```
mean_col("black", "white")
mean_col(c("black", "white"))
mean_col("red", "blue")
```

`mix_col` *Mix colors via Munsell color space*

Description

`mix_col()` simulates subtractive (pigment) color mixing by converting input colors to Munsell notation via `aqp::col2Munsell()`, mixing them with `aqp::mixMunsell()`, and converting the result back to an R color string via `aqp::parseMunsell()`.

Usage

```
mix_col(..., w = NULL, mixingMethod = "adaptive")
```

Arguments

`...`	Colors to mix. Can be individual color strings or character vectors; all are combined into a single vector.
`w`	A numeric vector of weights or proportions the same length as the combined color vector. Defaults to equal weights.
`mixingMethod`	Mixing method passed to `aqp::mixMunsell()`.

Value

A single R color string in hex notation.

See Also

`mean_col()` for a simpler quadratic-mean RGB approach (no extra packages required).

Examples

```
should_run_ex <- requireNamespace("aqp", quietly = TRUE) &&
    (interactive() || identical(Sys.getenv("IN_PKGDOWN"), "true"))
if (should_run_ex) {
  mix_col(c("red", "blue"))
}
if (should_run_ex) {
  mix_col("red", "yellow", "blue", w = c(2, 1, 1))
}
```

pattern_hex *Hex pattern matrix*

Description

pattern_hex() returns an integer matrix indicating where each color (or other graphical element) should be drawn on a (horizontal) hex grid for a specified hex pattern type and subtype. names_hex lists the currently supported hex types.

Usage

```
pattern_hex(type = "hex", subtype = NULL, nrow = 5L, ncol = 5L)

names_hex
```

Arguments

type	Currently just supports "hex".
subtype	An integer indicating number of colors (or other graphical elements).
nrow	Number of rows (height).
ncol	Number of columns (width).

Details

"hex" Attempts to use a uniform coloring if it exists. For subtype 1L, 2L, and 3L we use the "hex1" pattern. For subtype 4L we use the "hex2" pattern. For subtype 7L we use the "hex3" pattern. Else a uniform coloring does not exist and we use the "hex_skew" pattern.

"hex1" Provides the 1-uniform colorings of a hexagonal tiling. Only exists for subtype 1L, 2L, or 3L.

"hex2" Provides the 2-uniform colorings of a hexagonal tiling. Only exists for subtype 2L or 4L.

"hex3" Provides the 3-uniform colorings of a hexagonal tiling. Only exists for subtype 2L or 7L.

"hex_skew" For the "hex_skew" type we cycle through subtype elements on the horizontal line and "main" diagonal line. For some subtype numbers this may lead to noticeable color repeats on the "skew" diagonal line. If subtype is strictly greater than 2L then a hexagon should never touch another hexagon of the same color.

Value

A matrix of integer values indicating where the each color or other graphical elements should be drawn on a horizontal hex grid (i.e. hexagons are assumed to be pointy side up). Indices [1,1] of the matrix corresponds to the bottom-left of the grid while indices [1,ncol] corresponds to the bottom-right of the grid. The even rows are assumed to be on the **left** of the ones on the odd rows (for those in the same column in the matrix). This matrix has a "pattern_hex" subclass which supports a special print() method.

See Also

grid.pattern_regular_polygon() for drawing to a graphics device hexagons, triangles, circles, etc. in hexagon patterns. The tiling vignette features several examples of regular polygon tiling using this both the "hex" and "hex_circle" types `vignette("tiling", package = "gridpattern")`. For more information on uniform colorings of a hexagonal tiling see https://en.wikipedia.org/wiki/Hexagonal_tiling#Uniform_colorings.

Examples

```
# supported hex names
print(names_hex)

# 1-uniform 3-color
hex_3color <- pattern_hex("hex1", 3L, nrow = 7L, ncol = 9L)
print(hex_3color)

# 2-uniform 4-color
hex_4color <- pattern_hex("hex2", 4L, nrow = 7L, ncol = 9L)
print(hex_4color)
```

---

pattern_square *Square pattern matrix*

Description

`pattern_square()` returns an integer matrix indicating where each color (or other graphical element) should be drawn on a rectangular grid for a specified square pattern type and subtype. `names_square` lists the currently supported square types (excluding those in `names_weave`).

Usage

```
pattern_square(type = "diagonal", subtype = NULL, nrow = 5L, ncol = 5L)

names_square
```

Arguments

type	Either "diagonal" (default), "diagonal_skew", "horizontal", "vertical", or any type in `names_weave`. See Details.
subtype	See Details. For "diagonal", "diagonal_skew", "horizontal", or "vertical" an integer of the desired number of colors (or other graphical elements).
nrow	Number of rows (height).
ncol	Number of columns (width).

Details

"horizontal", "vertical" "horizontal" and "vertical" simply cycle through the colors either horizontally or vertically. Use `subtype` to indicate the (integer) number of colors (or other graphical elements). "horizontal" will produce horizontal stripes of color whereas "vertical" will produce vertical stripes.

"diagonal", "diagonal_skew" "diagonal" and "diagonal_skew" simply cycle through the colors both horizontally and vertically. Use `subtype` to indicate the (integer) number of colors (or other graphical elements). If two colors are requested this provides the standard two-color checkerboard pattern. If there are more than three colors than "diagonal" will have colored diagonals going from top left to bottom right while "diagonal_skew" will have them going form bottom left to top right.

"square" "square" attempts a uniform coloring using "square_tiling" before falling falling back on "diagonal". If `subtype` is `1L`, `2L`, `3L`, or `4L` uses "square_tiling" else uses "diagonal".

"square_tiling" "square_tiling" supports uniform coloring for (non-staggered) square tilings. Use `subtype` to either indicate the (integer) number of colors or a string with four integers such as `"1231"` (will fill in a 2x2 matrix by row which will then be tiled). Supports up to a max of four colors.

any pattern from `names_weave` We simply convert the logical matrix returned by `pattern_weave()` into an integer matrix by having any `TRUE` set to `1L` and `FALSE` set to `2L`. Hence the various weave patterns only support (up to) two-color patterns. See `pattern_weave()` for more details about supported `type` and `subtype`.

Value

A matrix of integer values indicating where the each color (or other graphical element) should be drawn on a rectangular grid. Indices `[1,1]` of the matrix corresponds to the bottom-left of the grid while indices `[1,ncol]` corresponds to the bottom-right of the grid. This matrix has a "pattern_square" subclass which supports a special `print()` method.

See Also

`grid.pattern_regular_polygon()` for drawing to a graphics device polygons in multiple color/size/shape patterns. `pattern_weave()` for more information on "weave" patterns.

Examples

```
# supported square names
print(names_square)

# (main) diagonal has colors going from top left to bottom right
diagonal <- pattern_square("diagonal", 4L, nrow = 7L, ncol = 9L)
print(diagonal)

# skew diagonal has colors going from bottom left to top right
skew <- pattern_square("diagonal_skew", 4L, nrow = 7L, ncol = 9L)
print(skew)

horizontal <- pattern_square("horizontal", 4L, nrow = 8L, ncol = 8L)
```

```
print(horizontal)

vertical <- pattern_square("vertical", 4L, nrow = 8L, ncol = 8L)
print(vertical)

# uniform coloring using 4 colors
color4 <- pattern_square("square_tiling", 4L, nrow = 7L, ncol = 9L)
print(color4)

# uniform coloring using 3 colors
color3 <- pattern_square("square_tiling", 3L, nrow = 7L, ncol = 9L)
print(color3)

# also supports the various 'weave' patterns
zigzag <- pattern_square("twill_zigzag", nrow = 15L, ncol = 9L)
print(zigzag)
```

`pattern_weave` *Weave pattern matrix*

Description

`pattern_weave()` returns a logical matrix indicating where the warp lines should be "up" for a specified weave pattern type and subtype. `names_weave` is a character vector listing supported weave pattern types.

Usage

```
pattern_weave(type = "plain", subtype = NULL, nrow = 5L, ncol = 5L)

names_weave
```

Arguments

`type`	Type of weave. See Details.
`subtype`	Subtype of weave. See Details.
`nrow`	Number of rows (length of warp).
`ncol`	Number of columns (length of weft).

Details

Here is a list of the various weave `type`s supported:

basket A simple criss-cross pattern using two threads at a time. Same as the "matt_irregular" weave but with a default `subtype` of `2L`.

matt A simple criss-cross pattern using 3 (or more) threads at a time. Same as the "matt_irregular" weave but with a default `subtype` of `3L`.

matt_irregular A generalization of the "plain" weave. A character `subtype` `"U/D(L+R)"` is a standard matt weave specification: `U` indicates number warp up, `D` indicates number warp down, `L` indicates number of warp up in repeat, and `R` indicates number of warp down in repeat. An integer `subtype` `N` will be interpreted as a `"N/N(N+N)"` irregular matt weave. A character `subtype` `"U/D"` will be interpreted as a `"U/D(U+D)"` irregular matt weave. Has a default `subtype` of `"3/2(4+2)"`.

plain A simple criss-cross pattern. Same as the "matt_irregular" weave but with a default `subtype` of `1L`.

rib_warp A plain weave variation that emphasizes vertical lines. An integer `subtype` `N` will be interpreted as a "matt_irregular" `"N/N(1+1)"` weave. A character `subtype` `"U/D"` will be interpreted as a "matt_irregular" `"U/D(1+1)"` weave. Default `subtype` of `2L`.

satin A "regular" satin weave is a special type of the elongated twill weave with a move number carefully chosen so no twill line is distinguishable. Same as the "twill_elongated" weave but with a default `subtype` of `5L`.

twill A simple diagonal pattern. Same as the "twill_elongated" weave but with a default `subtype` of `"2/1"`.

twill_elongated A generalization of the "twill" weave. A character `subtype` `"U/D(M)"` is a standard twill weave specification: `U` indicates number warp up, `D` indicates number warp down, and `M` indicates the "move" number. A character `subtype` `"U/D"` will be interpreted as a `"U/D(1)"` elongated twill weave. An integer `subtype` `N` will provide a `"{N-1}/1(1)"` elongated twill weave if `N` is less than 5, 6, or greater than 14 otherwise it will provide a `"{N-1}/1(M)"` weave where `M` is the largest possible regular "satin" move number. Default `subtype` of `"4/3(2)"`.

twill_herringbone Adds a (vertical) "herringbone" effect to the specified "twill_elongated" weave. Default `subtype` of `"4/3(2)"`.

twill_zigzag Adds a (vertical) "zig-zag" effect to the specified "twill_elongated" weave. Default `subtype` of `"4/3(2)"`.

For both "matt" and "twill" weaves the `U/D` part of the subtype can be further extended to `U1/D1*U2/D2`, `U1/D1*U2/D2*U3/D3`, etc. For the "matt" weave the "(L+R)" part of the subtype can be further extended to `(L1+R1+L2+R2)`, `(L1+R1+L2+R2+L3+R3)`, etc.

Value

A matrix of logical values indicating where the "warp" is "up" (if `TRUE`) or "down" (if `FALSE`). Indices `[1,1]` of the matrix corresponds to the bottom-left of the weave while indices `[1,ncol]` corresponds to the bottom-right of the weave. This matrix has a "pattern_weave" subclass which supports a special `print()` method.

See Also

`grid.pattern_weave()` for drawing weaves onto a graphics device. See https://textilestudycenter.com/derivatives-of-plain-weave/ for further information on the "matt" family of weaves, https://textilelearner.net/twill-weave-features-classification-derivatives-and-uses/ for further information on the "twill" family of weaves, and https://texwiz101.blogspot.com/2012/03/features-and-classification-of-satin.html for further information on "satin" weaves.

Examples

```
# supported weave names
print(names_weave)

plain <- pattern_weave("plain", nrow = 7, ncol = 9)
print(plain)

matt_irregular <- pattern_weave("matt_irregular", nrow = 9, ncol = 11)
print(matt_irregular)

satin <- pattern_weave("satin", nrow = 9, ncol = 11)
print(satin)

twill <- pattern_weave("twill", nrow = 9, ncol = 11)
print(twill)

twill_zigzag <- pattern_weave("twill_zigzag", nrow = 18, ncol = 11)
print(twill_zigzag)
```

---

patternFill *Create patterned fills by pattern name*

Description

`patternFill()` returns `grid::pattern()` fill objects. It is a wrapper around `patternGrob()`.

Usage

```
patternFill(
  ...,
  x = 0.5,
  y = 0.5,
  width = 1,
  height = 1,
  default.units = "npc",
  just = "centre",
  hjust = NULL,
  vjust = NULL,
  group = TRUE
)
```

Arguments

`...`	Passed to `patternGrob()`.
`x`, `y`, `width`, `height`	The size of the `grid::pattern()` tile.
`default.units`	The default `grid::unit()` unit to use for `x`, `y`, `width`, and `height`.

`just`, `hjust`, `vjust`
: The justification of the tile relative to its location.

`group`
: A logical indicating whether the pattern is relative to the bounding box of the grob or whether it is relative to individual shapes within the grob. Ignored if R is less than version 4.2.

Value

A `grid::pattern()` fill object.

Examples

```
if (guess_has_R4.1_features("patterns") &&
    require("grid", quietly = TRUE)) {
  grid.newpage()
  stripe_fill <- patternFill("stripe", fill = c("red", "blue"))
  grid.circle(gp = gpar(fill = stripe_fill))
}

if (guess_has_R4.1_features("patterns") &&
    require("ggplot2", quietly = TRUE) &&
    (getRversion() >= "4.2")) {
  grid.newpage()
  weave_fill <- patternFill("weave", fill = "red", fill2 = "blue",
                            colour = "transparent")
  hex_fill <- patternFill("polygon_tiling", type = "hexagonal",
                          fill = c("black", "white", "grey"),
                          colour = "transparent")
  df <- data.frame(trt = c("a", "b"), outcome = c(1.9, 3.2))
  gg <- ggplot(df, aes(trt, outcome)) +
    geom_col(fill = list(weave_fill, hex_fill))
  plot(gg)
}
```

---

prefer_pattern *Resolve pattern name conflicts*

Description

`prefer_pattern()` sets which pattern a bare pattern name resolves to, taking priority over all other patterns with that name. `unprefer_pattern()` removes such preferences. `conflicting_patterns()` returns the available patterns whose bare name is shared by more than one pattern (see `register_pattern()` for how bare names are resolved). Using such a bare name throws an error unless a preference has been set.

Usage

```
prefer_pattern(pattern)

unprefer_pattern(pattern)

conflicting_patterns()
```

Arguments

pattern	A character vector. For `prefer_pattern()` the qualified names `"owner::name"` of the preferred patterns where `owner` is either the package that registered it (`"gridpattern"` for the builtin patterns) or `"R_GlobalEnv"` for patterns registered by the user e.g. `c("gridpattern::stripe", "R_GlobalEnv::wave")`. Can't contain more than one pattern with the same bare name. If length zero (e.g. `character(0)` or `NULL`) then `prefer_pattern()` does nothing. For `unprefer_pattern()` bare pattern names e.g. `"stripe"` (removes any preference for that name) and/or qualified names (only removes the preference if it is for that pattern).

Details

`prefer_pattern()` and `unprefer_pattern()` are vectorized and return the previous preferences (if any) so they may be used to temporarily change preferences:

```
old <- prefer_pattern(new)
# ... use patterns ...
unprefer_pattern(new)
prefer_pattern(old) # does nothing if `old` is empty
```

Preferences only last for the current R session. Users may put `prefer_pattern()` calls in their `.Rprofile` (or scripts) to make them persistent. Packages should not call `prefer_pattern()` and should instead refer to patterns by their qualified name `"pkg::name"`.

Value

`prefer_pattern()` and `unprefer_pattern()` invisibly return a character vector of the qualified names of the previous preferences (for the bare names in `pattern`) that existed (`character(0)` if there weren't any). `conflicting_patterns()` returns (for just the conflicting patterns) a data frame with columns

name The pattern name.

owner The package that registered the pattern or `"R_GlobalEnv"` for the user.

kind Either `"geometry"` or `"array"`.

source One of `"builtin"`, `"register_pattern"`, or `"options"`.

active Whether the bare `name` currently resolves to this pattern.

Examples

```
create_pattern_polygon <- function(params, boundary_df, aspect_ratio, legend = FALSE) {
  grid::polygonGrob(boundary_df$x, boundary_df$y, boundary_df$id,
                    gp = grid::gpar(fill = params$pattern_fill))
}
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
# don't clobber any pre-existing user "stripe" pattern
run_example <- !has_pattern("R_GlobalEnv::stripe")
if (run_example) {
  # temporarily remove any pre-existing "stripe" preference
  old <- unprefer_pattern("stripe")
  register_pattern("stripe", create_pattern_polygon)
}

if (run_example) {
  print(conflicting_patterns())
}

if (run_example) {
  # bare "stripe" is now ambiguous
  grid::grid.newpage()
  try(grid.pattern("stripe", x_hex, y_hex))
}

if (run_example) {
  # prefer the user's "stripe" pattern over the builtin "stripe" pattern
  prefer_pattern("R_GlobalEnv::stripe")
  grid::grid.newpage()
  grid.pattern("stripe", x_hex, y_hex, fill = "blue")
}

if (run_example) {
  print(conflicting_patterns())
}

if (run_example) {
  # remove the preference
  unprefer_pattern("stripe")
  # remove "R_GlobalEnv::stripe" (not "gridpattern::stripe")
  unregister_pattern("stripe")
  # restore any pre-existing preference
  prefer_pattern(old)
}
```

`register_pattern` *Register patterns*

Description

`register_pattern()` registers a pattern so it can be used by `grid.pattern()` / `patternGrob()` (and {ggpattern}). `unregister_pattern()` removes a pattern previously registered by the same owner (as determined by `env`); it never removes patterns registered by other packages or the builtin {gridpattern} patterns.

Usage

```
register_pattern(
  name,
  fn,
  kind = c("geometry", "array"),
  defaults = list(),
  unprefixed = character(),
  env = caller_env()
)

unregister_pattern(name, env = caller_env())
```

Arguments

`name`	Pattern name. Must not contain `":"`.
`fn`	Pattern function. Its required signature depends on `kind`: **geometry** `function(params, boundary_df, aspect_ratio, legend)` returning a grid grob that lies within the boundary (e.g. clipped to it). `params` is a list of (prefixed) pattern parameters e.g. `params$pattern_fill`, `boundary_df` is a data frame with `x`, `y`, and `id` columns describing the boundary polygon(s) in "npc" coordinates, `aspect_ratio` is the (best guess) aspect ratio of the viewport, and `legend` is whether the pattern is being drawn in a {ggpattern} legend key. **array** `function(width, height, params, legend)` returning a 3D array of RGBA values (all values in the range [0, 1]) which {gridpattern} will mask to the boundary. `width` and `height` are the dimensions (in pixels) of the boundary's bounding box and `params` and `legend` are as for geometry patterns. See the "Developing Patterns" vignette `vignette("developing-patterns", package = "gridpattern")` for more details and examples.
`kind`	Either `"geometry"` or `"array"`.
`defaults`	A named list of pattern-specific default parameter values e.g. `list(pattern_type = "running", pattern_spacing = 0.1)`. Names are used as-is and must be `pattern_` prefixed unless listed in `unprefixed`. Values may be constants or functions of the form `function(params, gp)` where `params` is the list of pattern parameters filled in so far and `gp` is the grob's `grid::gpar()` object e.g. `list(pattern_amplitude = function(params, gp) 0.5 * params$pattern_spacing)` or `list(pattern_size = function(params, gp) gp$fontsize %||% 12)`. These replace the generic defaults for parameters that weren't supplied. The remaining generic defaults are filled in first and then these in the order given so a function

default may use any generic parameter and any earlier pattern-specific default. Note {ggpattern} supplies its own (generic) defaults for most of its pattern aesthetics so these will mainly be used with grid.pattern() / patternGrob().

unprefixed — A character vector of parameter names without the pattern_ prefix that the pattern reads so that defaults may set them. The usual case is a plain {ggplot2} aesthetic that {ggpattern} passes to pattern functions alongside the pattern_ aesthetics e.g. defaults = list(fill = "black"), unprefixed = "fill" lets a pattern that reads params$fill under {ggpattern} also work with grid.pattern(). Since such patterns are rare any other unprefixed name (e.g. fill or spacing instead of pattern_fill or pattern_spacing) is an error of class "gridpattern_error_unprefixed_de

env — The environment used to determine the "owner" of the pattern. If writing a function that registers patterns on behalf of its caller add an env = rlang::caller_env() argument to that function and pass it on.

Details

Each pattern is "owned" by either the package that registered it or by the user (if registered from the global environment e.g. in a script or at the console). Patterns may always be referred to by their qualified name "owner::name" e.g. "gridpattern::stripe", "pkg::name", or "R_GlobalEnv::name". A bare pattern name like "stripe" is resolved as follows:

1. If the user has set a preference with prefer_pattern() then the preferred pattern.
2. If exactly one owner (the user, {gridpattern}, or a loaded package) has a pattern with that name then that pattern. Patterns set by the user with the (legacy) ggpattern_geometry_funcs / ggpattern_array_funcs options are owned by the user. If the user has more than one pattern with that name (from register_pattern() and/or the options) an error is thrown asking them to remove all but one of them (since they share the same qualified name "R_GlobalEnv::name" a preference can't choose between them).
3. Otherwise there is a conflict and an error is thrown, asking for either a qualified "owner::name" or a prefer_pattern() preference. In particular, a user pattern with the same name as a builtin {gridpattern} pattern does not mask the builtin pattern without a preference.

conflicting_patterns() lists the patterns whose bare names are shared by more than one pattern.

patternGrob() resolves the pattern name when the grob is created (if the pattern is registered) so redrawing the grob (e.g. when resizing an interactive graphics device) still works even if the pattern is later unregistered or conflicted (e.g. by local_pattern() going out of scope). Changing the grob's pattern with grid::editGrob() resolves the new pattern name when drawn.

Packages should register their patterns in their .onLoad() function. We recommend wrapping the registration in rlang::on_package_load() so the pattern is registered whenever {gridpattern} is loaded (allowing {gridpattern} to be in Suggests and surviving {gridpattern} being reloaded). Since a Suggests version requirement isn't enforced when installing or loading packages, also check that the installed {gridpattern} has register_pattern() (and otherwise fall back to setting the legacy ggpattern_geometry_funcs option):

```
.onLoad <- function(libname, pkgname) {
  rlang::on_package_load("gridpattern", {
   if (exists("register_pattern", envir = asNamespace("gridpattern"), inherits = FALSE)) {
```

```
    gridpattern::register_pattern("brick", create_pattern_brick)
  } else {
    funcs <- getOption("ggpattern_geometry_funcs", list())
    funcs[["brick"]] <- create_pattern_brick
    options(ggpattern_geometry_funcs = funcs)
  }
 })
}
```

Package code that relies on a particular pattern should use its qualified name (e.g. `"gridpattern::stripe"`) since other packages (or users) may register patterns with the same bare name.

Value

`register_pattern()` and `unregister_pattern()` invisibly return the previous registration (a list) or `NULL` if there wasn't one.

See Also

`registered_patterns()` and `has_pattern()` to list available patterns. `local_pattern()` to temporarily register a pattern. `prefer_pattern()` and `conflicting_patterns()` to resolve pattern name conflicts.

Examples

```
create_pattern_polygon <- function(params, boundary_df, aspect_ratio, legend = FALSE) {
  grid::polygonGrob(boundary_df$x, boundary_df$y, boundary_df$id,
                    gp = grid::gpar(fill = params$pattern_fill))
}
x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
# don't clobber any pre-existing "polygon" pattern
if (!has_pattern("polygon")) {
  register_pattern("polygon", create_pattern_polygon)
  grid::grid.newpage()
  grid.pattern("polygon", x_hex, y_hex, fill = "blue")
  unregister_pattern("polygon")
}
```

---

registered_patterns *List available patterns*

Description

`registered_patterns()` returns a data frame of all available patterns. `has_pattern()` returns whether pattern names currently resolve to a pattern i.e. whether `grid.pattern()` / `patternGrob()` could use them. See `register_pattern()` for how bare pattern names are resolved.

Usage

```
registered_patterns()

has_pattern(pattern)
```

Arguments

pattern	A character vector of bare pattern names "name" and/or qualified pattern names "owner::name".

Value

registered_patterns() returns a data frame with columns

name The pattern name.

owner The package that registered the pattern or "R_GlobalEnv" for the user.

kind Either "geometry" or "array".

source One of "builtin", "register_pattern", or "options".

active Whether the bare name currently resolves to this pattern.

has_pattern() returns a logical vector the same length as pattern. A bare name with an unresolved conflict (see conflicting_patterns()) returns FALSE.

See Also

register_pattern() to register patterns.

Examples

```
head(registered_patterns())
has_pattern(c("stripe", "gridpattern::stripe", "polygon"))

create_pattern_polygon <- function(params, boundary_df, aspect_ratio, legend = FALSE) {
  grid::polygonGrob(boundary_df$x, boundary_df$y, boundary_df$id,
                    gp = grid::gpar(fill = params$pattern_fill))
}
# don't clobber any pre-existing "polygon" pattern
run_example <- !has_pattern("polygon")
if (run_example) {
  register_pattern("polygon", create_pattern_polygon)
  print(subset(registered_patterns(), name == "polygon"))
}
if (run_example) {
  print(has_pattern("polygon"))
}
if (run_example) {
  unregister_pattern("polygon")
}
```

reset_image_cache *Reset 'gridpattern' image cache*

Description

grid.pattern_image() and grid.pattern_placeholder() store images in a cache (so we won't download image URLs over and over). reset_image_cache() resets this cache.

Usage

```
reset_image_cache()
```

star_scale *Compute regular star polygon scale or angles*

Description

star_scale() computes star scale value given an internal or external angle. star_angle() computes star angle (internal or external) given a scale value.

Usage

```
star_scale(n_vertices, angle, external = FALSE)

star_angle(n_vertices, scale, external = FALSE)
```

Arguments

n_vertices	Number of exterior vertices.
angle	Angle in degrees.
external	If TRUE angle should be considered an external angle.
scale	Scale from 0 to 1.

Details

grid.pattern_regular_polygon() parameterizes regular star polygons with the number of its external vertices and a scale that equals the fraction of the radius of the circle that circumscribes the interior vertices divided by the radius of the circle that circumscribes the exterior vertices. These helper functions help convert between that parameterization and either the internal or external angle of the regular star polygon.

Value

star_scale() returns a numeric value between 0 and 1 intended for use as the scale argument in grid.pattern_regular_polygon(). star_angle() returns a numeric value between 0 and 360 (degrees).

Examples

```
# |8/3| star has internal angle 45 degrees and external angle 90 degrees
scale <- star_scale(8, 45)
scale2 <- star_scale(8, 90, external = TRUE)
all.equal(scale, scale2)
star_angle(8, scale)
star_angle(8, scale, external = TRUE)

grid.pattern_regular_polygon(shape = "star8", scale = scale, angle = 0,
                             spacing = 0.2, density = 0.8)
```

`update_alpha` *Update colour and/or pattern transparency*

Description

`update_alpha()` modifies the transparency of colours and/or patterns.

Usage

```
update_alpha(fill, alpha)
```

Arguments

`fill`	A fill colour given as a `character` or `integer` vector, or as a (list of) `<GridPattern>` object(s) and/or colour(s).
`alpha`	A transparency value between 0 (transparent) and 1 (opaque), parallel to `fill`.

Details

- This is a fork of pattern utilities mainly added to `{ggplot2}` by Teun van den Brand.
- `update_alpha()` does not depend on `{ggplot2}` or `{scales}`.
- Like `ggplot2::fill_alpha()` but unlike `scales::alpha()` it also attempts to set the transparency of `<GridPattern>` objects.
- Unlike `ggplot2::fill_alpha()` it will work on a list of length one containing a vector of color strings.

Value

A `character` vector of colours or list of `<GridPattern>` objects.

Usage in other packages

To avoid taking a dependency on `gridpattern` you may copy the source of `update_alpha()` into your own package under the permissive MIT license. Either use `usethis::use_standalone("trevorld/gridpattern", "standalone-update_alpha.R")` or copy the file `update_alpha.R` into your `R` directory and add `grDevices`, `grid`, and `rlang` to the `Imports` of your `DESCRIPTION` file.

Examples

```
# Typical color input
update_alpha("red", 0.5)

# Pattern input
if (getRversion() >= "4.2" && requireNamespace("grid", quietly = TRUE)) {
  update_alpha(list(grid::linearGradient()), 0.5)
}
```

Index

alphaMaskGrob, 4
ambient::noise_cubic(), *12*
ambient::noise_perlin(), *12*
ambient::noise_simplex(), *12*
ambient::noise_value(), *12*
ambient::noise_white(), *12*
ambient::noise_worley(), *12*
aqp::col2Munsell(), *60*
aqp::mixMunsell(), *60*
aqp::parseMunsell(), *60*

base::options(), *3*

clippingPathGrob, 5
conflicting_patterns *(*prefer_pattern*)*, *67*
conflicting_patterns(), *71–73*

ggplot2::fill_alpha(), *75*
gpar, *4*, *6*, *8*, *12, 13*, *16*, *18, 19*, *21*, *23*, *25*, *27*, *29, 30*, *33*, *35, 36*, *38*, *42*, *45*, *47*, *49*, *52*, *55*
graphics::points(), *32*
grDevices::png(), *4*, *6*, *23*, *27*, *45*, *49*
grid.pattern, 7
grid.pattern(), *58*, *70–72*
grid.pattern_ambient, 10
grid.pattern_ambient(), *8*, *36*
grid.pattern_aRtsy, 13
grid.pattern_aRtsy(), *8*
grid.pattern_circle, 14
grid.pattern_circle(), *8*, *42*
grid.pattern_crosshatch, 16
grid.pattern_crosshatch(), *8*, *23*, *28*
grid.pattern_fill, 18
grid.pattern_gradient, 19
grid.pattern_gradient(), *8*
grid.pattern_hatch, 21
grid.pattern_hatch(), *8*
grid.pattern_image, 24
grid.pattern_image(), *8*, *35*, *74*
grid.pattern_line, 26
grid.pattern_line(), *8*, *23*, *47*, *52*
grid.pattern_magick, 28
grid.pattern_magick(), *8*
grid.pattern_none, 30
grid.pattern_pch, 31
grid.pattern_pch(), *8*
grid.pattern_placeholder, 34
grid.pattern_placeholder(), *8*, *25*, *74*
grid.pattern_plasma, 35
grid.pattern_plasma(), *8*, *12*
grid.pattern_polygon_tiling, 37
grid.pattern_polygon_tiling(), *8*
grid.pattern_regular_polygon, 40
grid.pattern_regular_polygon(), *9*, *16*, *33*, *39*, *62, 63*, *74*
grid.pattern_rose, 43
grid.pattern_rose(), *9*
grid.pattern_stripe, 46
grid.pattern_stripe(), *9*, *18*, *26*, *28*, *52*
grid.pattern_text, 48
grid.pattern_text(), *9*
grid.pattern_wave, 50
grid.pattern_wave(), *9*, *28*, *47*
grid.pattern_weave, 53
grid.pattern_weave(), *9*, *18*, *65*
grid::editGrob(), *71*
grid::gpar(), *49*, *58*, *70*
grid::grid.null(), *8*, *31*
grid::grid.polygon(), *19*
grid::pattern(), *15*, *17*, *19*, *32*, *38*, *41*, *44*, *47*, *51*, *66, 67*
grid::textGrob(), *49*
grid::unit(), *15*, *17*, *22*, *27*, *32*, *38*, *41*, *44*, *47*, *49*, *51*, *54*, *66*
gridpattern *(*gridpattern-package*)*, 3
gridpattern-package, 3
guess_has_R4.1_features, 56

has_pattern *(registered_patterns)*, 72
has_pattern(), *72*

local_pattern, 57
local_pattern(), *71, 72*

magick::magick_set_seed(), *36*
mean_col, 59
mean_col(), *60*
mix_col, 60
mix_col(), *59*

names_aRtsy *(grid.pattern_aRtsy)*, 13
names_hatch *(grid.pattern_hatch)*, 21
names_hex *(pattern_hex)*, 61
names_magick *(grid.pattern_magick)*, 28
names_magick_intensity
(grid.pattern_magick), 28
names_magick_stripe
(grid.pattern_magick), 28
names_pattern *(grid.pattern)*, 7
names_placeholder
(grid.pattern_placeholder), 34
names_polygon_tiling
(grid.pattern_polygon_tiling), 37
names_square *(pattern_square)*, 62
names_wave *(grid.pattern_wave)*, 50
names_weave *(pattern_weave)*, 64

pattern_hex, 61
pattern_hex(), *15*, *32*, *42*, *44*, *49*
pattern_square, 62
pattern_square(), *15*, *32*, *42*, *44*, *49*
pattern_weave, 64
pattern_weave(), *15*, *32*, *42*, *44*, *49*, *54, 55*, *63*
patternFill, 66
patternGrob *(grid.pattern)*, 7
patternGrob(), *58*, *66*, *70–72*
png::readPNG(), *4*, *6*, *23*, *27*, *45*, *49*
prefer_pattern, 67
prefer_pattern(), *71, 72*

ragg::agg_capture(), *4*, *6*, *23*, *27*, *45*, *49*
ragg::agg_png(), *4*, *6*, *23*, *27*, *45*, *49*
register_pattern, 69
register_pattern(), *3*, *8, 9*, *57, 58*, *67*, *72, 73*
registered_patterns, 72
registered_patterns(), *72*
reset_image_cache, 74
reset_image_cache(), *25*, *35*
rlang::on_package_load(), *71*

scales::alpha(), *75*
star_angle *(star_scale)*, 74
star_scale, 74

unprefer_pattern *(prefer_pattern)*, 67
unregister_pattern *(register_pattern)*, 69
update_alpha, 75