

# Package ‘gridpattern’

July 17, 2021

**Type** Package

**Title** 'grid' Pattern Grobs

**Version** 0.2.1

**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

**RoxygenNote** 7.1.1

**Depends** R (>= 3.4.0)

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

**Suggests** ambient, knitr, ragg, rmarkdown, testthat, vdiff

**VignetteBuilder** knitr, rmarkdown

**NeedsCompilation** no

**Author** Mike FC [aut] (Patterns extracted from ggpattern),  
Trevor L Davis [aut, cre],  
Thomas Lin Pedersen [ctb] (new\_data\_frame() copied from ggplot2)

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

**Repository** CRAN

**Date/Publication** 2021-07-17 04:30:06 UTC

**R topics documented:**

clippingPathGrob . . . . .	2
grid.pattern . . . . .	3
grid.pattern_ambient . . . . .	7
grid.pattern_circle . . . . .	9
grid.pattern_crosshatch . . . . .	11
grid.pattern_gradient . . . . .	13
grid.pattern_image . . . . .	15
grid.pattern_magick . . . . .	17
grid.pattern_pch . . . . .	19
grid.pattern_placeholder . . . . .	21
grid.pattern_plasma . . . . .	23
grid.pattern_polygon_tiling . . . . .	25
grid.pattern_regular_polygon . . . . .	27
grid.pattern_rose . . . . .	30
grid.pattern_stripe . . . . .	33
grid.pattern_text . . . . .	35
grid.pattern_wave . . . . .	37
grid.pattern_weave . . . . .	39
mean_col . . . . .	41
pattern_hex . . . . .	42
pattern_square . . . . .	43
pattern_weave . . . . .	45
star_scale . . . . .	47

<b>Index</b>	<b>49</b>
--------------	-----------

---

clippingPathGrob	<i>Clip grob using another grob to specify the clipping path</i>
------------------	--

---

**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 = 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 use if use_R4.1_clipping is FALSE. If NULL (default) will use <code>ragg::agg_png()</code> if the suggested package <code>ragg</code> is available else <code>grDevices::png()</code> .
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 <code>gpar</code> . 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),
    rule = "evenodd")
  clipped <- clippingPathGrob(clippee, clipper, use_R4.1_clipping = FALSE)
  grid.newpage()
  grid.draw(clipped)
}

```

---

grid.pattern

*Create patterned grobs*

---

**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.
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 <code>gpar</code> . 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).

### Format

An object of class character of length 17.

### 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.
- 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.
- image** Image array patterns. See `grid.pattern_image()` 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.
- 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 <https://coolbutuseless.github.io/package/ggpattern/articles/developing-patterns-2.html> for more information.
- Custom array-based patterns** See <https://coolbutuseless.github.io/package/ggpattern/articles/developing-patterns-3.html> 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 patterns and their parameters.

**Examples**

```
print(names_pattern)
if (require("grid")) {
  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.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.newpage()
  grid.pattern("stripe", x_hex, y_hex, gp = gpar(col="blue", fill="red", lwd=2))

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

  # 'regular_polygon' pattern
  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.newpage()
  grid.pattern_regular_polygon(x_hex, y_hex, color = "transparent",
                               fill = c("white", "grey", "black"),
                               density = 1.0, spacing = 0.1,
                               shape = "convex6", type = "hex")

  # array-based patterns
  # 'image' pattern
  logo_filename <- system.file("img", "Rlogo.png", package="png")
  grid.newpage()
  grid.pattern("image", x_hex, y_hex, filename=logo_filename, type="fit")

  # 'plasma' pattern
  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 = 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

<code>fill</code>	Fill colour
<code>fill2</code>	Second colour
<code>frequency</code>	Determines the granularity of the features in the noise.
<code>interpolator</code>	How should values between sampled points be calculated? Either 'linear', 'hermite', or 'quintic' (default), ranging from lowest to highest quality.
<code>fractal</code>	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.
<code>octaves</code>	The number of noise layers used to create the fractal noise. Ignored if <code>fractal = 'none'</code> . Defaults to 3.
<code>lacunarity</code>	The frequency multiplier between successive noise layers when building fractal noise. Ignored if <code>fractal = 'none'</code> . Defaults to 2.
<code>gain</code>	The relative strength between successive noise layers when building fractal noise. Ignored if <code>fractal = 'none'</code> . Defaults to 0.5.
<code>perturbation</code>	The perturbation 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 erratic warping.
<code>perturbation_amplitude</code>	The maximal perturbation distance from the origin. Ignored if <code>perturbation = 'none'</code> . Defaults to 1.
<code>value</code>	The noise value to return. Either <ul style="list-style-type: none"> <li>• 'value' (default) A random value associated with the closest point</li> <li>• 'distance' The distance to the closest point</li> <li>• 'distance2' The distance to the nth closest point (n given by <code>distance_ind[1]</code>)</li> <li>• 'distance2add' Addition of the distance to the nth and mth closest point given in <code>distance_ind</code></li> <li>• 'distance2sub' Substraction of the distance to the nth and mth closest point given in <code>distance_ind</code></li> <li>• 'distance2mul' Multiplication of the distance to the nth and mth closest point given in <code>distance_ind</code></li> <li>• 'distance2div' Division of the distance to the nth and mth closest point given in <code>distance_ind</code></li> </ul>
<code>distance_ind</code>	Reference to the nth and mth closest points that should be used when calculating value.
<code>jitter</code>	The maximum distance a point can move from its start position during sampling of cell points.
<code>res</code>	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
<code>alpha</code>	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
<code>default.units</code>	A string indicating the default units to use if x or y are only given as numeric vectors.
<code>name</code>	A character identifier.



gp	An object of class "gpar", typically the output from a call to the function <code>gpar</code> . 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()`

**Examples**

```
if (requireNamespace("ambient")) {
  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")
  grid::grid.newpage()
  grid.pattern_ambient(x_hex, y_hex, fill = "green", fill2 = "blue", type = "cubic")
}
```

---

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,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
```

```

size = gp$lwd %||% 1,
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
fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype
size	Stroke 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.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in names_hex, names_square, or names_weave. See for <a href="#">pattern_hex()</a> , <a href="#">pattern_square()</a> , and <a href="#">pattern_weave()</a> for more information about supported type arguments.
subtype	See for <a href="#">pattern_hex()</a> , <a href="#">pattern_square()</a> , and <a href="#">pattern_weave()</a> 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 <a href="#">gpar</a> . 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 and ggpattern documentation: <https://coolbutuseless.github.io/package/ggpattern/articles/pattern-circle.html>

**Examples**

```
if (require("grid")) {
  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.newpage()
  grid.pattern_circle(x_hex, y_hex, density = 0.8, grid = "hex_circle",
                    gp = gpar(fill = c("blue", "yellow", "red")))
  grid.newpage()
  grid.pattern_circle(x_hex, y_hex, density = 1.2, grid = "hex_circle",
                    gp = gpar(fill = c("blue", "yellow", "red")))

  # using a "twill_zigzag" 'weave' pattern
  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,
alpha = gp$alpha %||% NA_real_,
linetype = gp$lty %||% 1,
size = gp$lwd %||% 1,
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
fill	Fill colour
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 (‘snpc’ units between 0 and 1).
xoffset	Shift pattern along x axis (‘snpc’ units between 0 and 1).
yoffset	Shift pattern along y axis (‘snpc’ units between 0 and 1).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors’ alpha value).
linetype	Stroke linetype
size	Stroke 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.
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 <code>gpar</code> . 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 as well as the ggpattern documentation: <https://coolbutuseless.github.io/package/ggpattern/articles/pattern-crosshatch.html>

**Examples**

```
if (require("grid")) {
  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.newpage()
  grid.pattern_crosshatch(x_hex, y_hex, density = 0.3,
                        gp = gpar(col = "blue", fill = "yellow"))
}
```

---

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 = 72,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
```

```

    vp = NULL
  )

```

### Arguments

<code>x</code>	A numeric vector or unit object specifying x-locations of the pattern boundary.
<code>y</code>	A numeric vector or unit object specifying y-locations of the pattern boundary.
<code>id</code>	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
<code>...</code>	Currently ignored
<code>fill</code>	Fill colour
<code>fill2</code>	Second colour
<code>orientation</code>	vertical, horizontal, or radial
<code>alpha</code>	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
<code>use_R4.1_gradients</code>	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.
<code>aspect_ratio</code>	Override aspect ratio
<code>key_scale_factor</code>	Additional scale factor for legend
<code>res</code>	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
<code>default.units</code>	A string indicating the default units to use if x or y are only given as numeric vectors.
<code>name</code>	A character identifier.
<code>gp</code>	An object of class "gpar", typically the output from a call to the function <code>gpar</code> . This is basically a list of graphical parameter settings.
<code>draw</code>	A logical value indicating whether graphics output should be produced.
<code>vp</code>	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 `ggpattern` documentation: <https://coolbutuseless.github.io/package/ggpattern/articles/pattern-gradient.html>

**Examples**

```

if (require("grid") && 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")
  grid.newpage()
  grid.pattern_gradient(x_hex, y_hex, fill = "green", orientation = "radial")
}

```

---

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 = "center",
  filter = "lanczos",
  alpha = gp$alpha %||% NA_real_,
  aspect_ratio = 1,
  key_scale_factor = 1,
  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. <code>magick::gravity_types()</code> returns a vector of supported values.
filter	Filter to use when scaling. <code>magick::filter_types()</code> 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 <code>gpar</code> . 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

The ggpattern documentation: <https://coolbutuseless.github.io/package/ggpattern/articles/pattern-image.html>



**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))
logo_filename <- system.file("img", "Rlogo.png", package = "png")
grid.pattern_image(x_hex, y_hex, filename = logo_filename, type = "fit")

# "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\_magick    *Magick patterned grobs*

---

**Description**

grid.pattern\_image() 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 = 72,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

names\_magick

names\_magick\_intensity

names\_magick\_stripe

**Arguments**

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

**Format**

An object of class character of length 54.

An object of class character of length 21.

An object of class character of length 19.

**Value**

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

**See Also**

The `ggpattern` documentation <https://coolbutuseless.github.io/package/ggpattern/articles/pattern-magick.html> and `imagemagick` documentation <http://www.imagemagick.org/script/formats.php> for more information.

**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_magick(x_hex, y_hex, type="octagons", fill="blue", scale=2)
# supported magick pattern names
print(names_magick)
```

---

grid.pattern_pch	<i>Plotting character patterned grobs</i>
------------------	---

---

**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,
  scale = 0.5,
  shape = 1L,
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  size = gp$lwd %||% 1,
  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
fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
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 <a href="#">graphics::points()</a> 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.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in names_hex, names_square, or names_weave. See for <a href="#">pattern_hex()</a> , <a href="#">pattern_square()</a> , and <a href="#">pattern_weave()</a> for more information about supported type arguments.
subtype	See for <a href="#">pattern_hex()</a> , <a href="#">pattern_square()</a> , and <a href="#">pattern_weave()</a> 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
size	Stroke 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 <a href="#">gpar</a> . 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**

```
if (require("grid")) {
  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 <- gpar(col = "black", fill = "lightblue")

  # 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)

  # pch 7-14 are compound shapes with no fill
  grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = 7:14, gp = gp,
                  spacing = 0.1, density = 0.4, angle = 0)

  # pch 15-20 are filled with 'col'
  grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = 15:20, gp = gp,
                  spacing = 0.1, density = 0.4, angle = 0)

  # pch 21-25 are filled with 'fill'
  grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = 21:25, gp = gp,
                  spacing = 0.1, density = 0.4, angle = 0)

  # using a 'basket' weave `type` with two shapes
  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 = "kitten",
alpha = gp$alpha %||% NA_real_,
aspect_ratio = 1,
key_scale_factor = 1,
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. <code>names_placeholder</code> is a vector of supported values. If you would like only greyscale images append <code>bw</code> 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 <code>gpar</code> . 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).

### Format

An object of class character of length 26.

**Value**

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

**See Also**

The ggpattern documentation: <https://coolbutuseless.github.io/package/ggpattern/articles/pattern-placeholder.html>

**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))
# 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 = 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	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 <code>gpar</code> . 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 ggpattern documentation: <https://coolbutuseless.github.io/package/ggpattern/articles/pattern-plasma.html>

**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_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,
  type = "square",
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  size = gp$lwd %||% 1,
  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
fill	Fill colour
angle	Rotation angle in degrees

spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
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
size	Stroke 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 <code>gpar</code> . 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).

**Format**

An object of class character of length 12.

**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:

**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.

**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.

**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.

**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)
if (require("grid")) {
  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 <- gpar(fill = "yellow", col = "black")
  gp2 <- gpar(fill = c("yellow", "red"), col = "black")
  gp3 <- gpar(fill = c("yellow", "red", "blue"), col = "black")

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

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

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

  grid.newpage()
  grid.pattern_polygon_tiling(x_hex, y_hex, type = "truncated_square", 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,
scale = 0.5,
shape = "convex4",
grid = "square",
type = NULL,
subtype = NULL,
rot = 0,
alpha = gp$alpha %||% NA_real_,
linetype = gp$lty %||% 1,
size = gp$lwd %||% 1,
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
fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
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", "null", or "square". 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.
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.

type	Adjusts the repeating of certain aesthetics such as color. Can use any type in names_hex, names_square, or names_weave. See for <a href="#">pattern_hex()</a> , <a href="#">pattern_square()</a> , and <a href="#">pattern_weave()</a> for more information about supported type arguments.
subtype	See for <a href="#">pattern_hex()</a> , <a href="#">pattern_square()</a> , and <a href="#">pattern_weave()</a> 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
size	Stroke 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 <a href="#">gpar</a> . 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**

```
if (require("grid")) {
  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.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)

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

# using a "twill_zigzag" 'weave' pattern
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.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.newpage()
grid.pattern_regular_polygon(x_hex, y_hex, fill = "green",
                             density = 1.0, spacing = 0.1,
                             shape = "convex3", grid = "hex")

}

```

---

grid.pattern_rose	<i>Rose curve patterned grobs</i>
-------------------	-----------------------------------

---

## 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,
  frequency = 0.1,
  grid = "square",
  type = NULL,

```

```

    subtype = NULL,
    rot = 0,
    alpha = gp$alpha %||% NA_real_,
    linetype = gp$lty %||% 1,
    size = gp$lwd %||% 1,
    use_R4.1_clipping = getOption("ggpattern_use_R4.1_clipping",
    getOption("ggpattern_use_R4.1_features")),
    png_device = NULL,
    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
fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
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.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in names_hex, names_square, or names_weave. See for <a href="#">pattern_hex()</a> , <a href="#">pattern_square()</a> , and <a href="#">pattern_weave()</a> for more information about supported type arguments.
subtype	See for <a href="#">pattern_hex()</a> , <a href="#">pattern_square()</a> , and <a href="#">pattern_weave()</a> 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
size	Stroke linewidth

<code>use_R4.1_clipping</code>	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.
<code>png_device</code>	“png” graphics device to use if <code>use_R4.1_clipping</code> is FALSE. If NULL (default) will use <code>ragg::agg_png()</code> if the suggested package <code>ragg</code> is available else <code>grDevices::png()</code> .
<code>res</code>	Resolution of desired rasterGrob in pixels per inch if <code>use_R4.1_clipping</code> is FALSE.
<code>default.units</code>	A string indicating the default units to use if <code>x</code> or <code>y</code> are only given as numeric vectors.
<code>name</code>	A character identifier.
<code>gp</code>	An object of class “gpar”, typically the output from a call to the function <code>gpar</code> . This is basically a list of graphical parameter settings.
<code>draw</code>	A logical value indicating whether graphics output should be produced.
<code>vp</code>	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\\_\(mathematics\)](https://en.wikipedia.org/wiki/Rose_(mathematics)) for more information.

**Examples**

```
if (require("grid") && 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))
  gp <- gpar(fill = c("blue", "red", "yellow", "green"), col = "black")

  grid.newpage()
  grid.pattern_rose(x_hex, y_hex,
                   spacing = 0.15, density = 0.5, angle = 0,
                   frequency = 1:4, gp = gp)

  grid.newpage()
  grid.pattern_rose(x_hex, y_hex,
                   spacing = 0.15, density = 0.5, angle = 0,
                   frequency = 1/1:4, gp = gp)

  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,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  size = gp$lwd %||% 1,
  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
fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('npc' units between 0 and 1).

xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype
size	Stroke 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.
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 <code>gpar</code> . 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 as well as the `ggpattern` documentation: <https://coolbutuseless.github.io/package/ggpattern/articles/pattern-stripe.html>

### Examples

```
if (require("grid")) {
  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.newpage()
  grid.pattern_stripe(x_hex, y_hex, density = 0.3,
                    gp = gpar(col = "blue", fill = "yellow"))
}
```

---

grid.pattern_text	<i>Text character patterned grobs</i>
-------------------	---------------------------------------

---

### 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$col %||% "grey20",
  angle = 30,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  scale = 0.5,
  shape = "X",
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  size = gp$fontsize %||% 12,
  fontfamily = gp$fontfamily %||% "sans",
  fontface = gp$fontface %||% "plain",
  use_R4.1_clipping = getOption("ggpattern_use_R4.1_clipping",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  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
angle	Rotation angle in degrees
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
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 <code>grid::textGrob()</code> 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.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in <code>names_hex</code> , <code>names_square</code> , or <code>names_weave</code> . See for <code>pattern_hex()</code> , <code>pattern_square()</code> , and <code>pattern_weave()</code> for more information about supported type arguments.
subtype	See for <code>pattern_hex()</code> , <code>pattern_square()</code> , and <code>pattern_weave()</code> 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	Stroke linewidth
fontfamily	The font family. See <code>grid::gpar()</code> for more details.
fontface	The font face. See <code>grid::gpar()</code> for more details.
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 use if <code>use_R4.1_clipping</code> is FALSE. If NULL (default) will use <code>ragg::agg_png()</code> if the suggested package <code>ragg</code> is available else <code>grDevices::png()</code> .
res	Resolution of desired rasterGrob in pixels per inch if <code>use_R4.1_clipping</code> 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 <code>gpar</code> . 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 (require("grid") && 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))

  playing_card_symbols <- c("\u2660", "\u2665", "\u2666", "\u2663")
  grid.newpage()
  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	<i>Wave patterned grobs</i>
-------------------	-----------------------------

---

**Description**

grid.pattern\_wave() draws a wave pattern onto the graphic device.

**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,
  amplitude = 0.5 * spacing,
  frequency = 1/spacing,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  size = gp$lwd %||% 1,
  grid = "square",
  type = "triangle",
  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
fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
amplitude	Wave amplitude ("snpc" units)
frequency	Linear frequency (inverse "snpc" units)
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype
size	Stroke 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.
type	Either "sine" or "triangle" (default).
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 <code>gpar</code> . 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 (require("grid")) {
  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.newpage()
  grid.pattern_wave(x_hex, y_hex, colour = "black", type = "sine",
                    fill = c("red", "blue"), density = 0.4,
                    spacing = 0.15, angle = 0,
                    amplitude = 0.05, frequency = 1 / 0.20)

  # zig-zag pattern is a wave of `type` "triangle"
  grid.newpage()
  grid.pattern_wave(x_hex, y_hex, colour = "black", type = "triangle",
                    fill = c("red", "blue"), density = 0.4,
                    spacing = 0.15, angle = 0, amplitude = 0.075)

}

```

---

grid.pattern_weave	<i>Weave patterned grobs</i>
--------------------	------------------------------

---

**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,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  size = gp$lwd %||% 1,
  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
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 ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype
size	Stroke 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.
type	The weave type. See <a href="#">pattern_weave()</a> for more details.
subtype	The weave subtype. See <a href="#">pattern_weave()</a> 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 <a href="#">gpar</a> . 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**

```

if (require("grid")) {
  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 <- 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.newpage()
  grid.pattern_weave(x_hex, y_hex, type = "matt_irregular",
                    fill2 = "yellow", gp = gp, spacing = 0.1, density = 0.3)

  # Twill weave
  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.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.newpage()
  gp$col <- NA
  grid.pattern_weave(x_hex, y_hex, type = "twill_herringbone",
                    fill2 = "yellow", gp = gp, spacing = 0.05, density = 1.0)
}

```

---

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.

**Examples**

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

---

pattern_hex	<i>Hex pattern matrix</i>
-------------	---------------------------

---

**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).

**Format**

An object of class character of length 5.

**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](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).

**Format**

An object of class character of length 6.

**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 from bottom left to top right.

**"square"** "square" attempts a uniform coloring using "square\_tiling" before 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).

**Format**

An object of class character of length 10.

**Details**

Here is a list of the various weave types 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.
- satint** 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)
```

---

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)

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



# Index

## \* datasets

- grid.pattern, 3
- grid.pattern\_magick, 17
- grid.pattern\_placeholder, 21
- grid.pattern\_polygon\_tiling, 25
- pattern\_hex, 42
- pattern\_square, 43
- pattern\_weave, 45

ambient::noise\_cubic(), 9

ambient::noise\_perlin(), 9

ambient::noise\_simplex(), 9

ambient::noise\_value(), 9

ambient::noise\_white(), 9

ambient::noise\_worley(), 9

clippingPathGrob, 2

gpar, 3, 5, 9, 10, 12, 14, 16, 18, 20, 22, 24, 26, 29, 32, 34, 36, 38, 40

graphics::points(), 20

grid.pattern, 3

grid.pattern\_ambient, 7

grid.pattern\_ambient(), 5

grid.pattern\_circle, 9

grid.pattern\_circle(), 5, 29

grid.pattern\_crosshatch, 11

grid.pattern\_crosshatch(), 5

grid.pattern\_gradient, 13

grid.pattern\_gradient(), 5

grid.pattern\_image, 15

grid.pattern\_image(), 5

grid.pattern\_magick, 17

grid.pattern\_magick(), 5

grid.pattern\_pch, 19

grid.pattern\_pch(), 5

grid.pattern\_placeholder, 21

grid.pattern\_placeholder(), 5

grid.pattern\_plasma, 23

grid.pattern\_plasma(), 5

grid.pattern\_polygon\_tiling, 25

grid.pattern\_polygon\_tiling(), 5

grid.pattern\_regular\_polygon, 27

grid.pattern\_regular\_polygon(), 5, 11, 21, 27, 43, 45, 48

grid.pattern\_rose, 30

grid.pattern\_stripe, 33

grid.pattern\_stripe(), 5

grid.pattern\_text, 35

grid.pattern\_text(), 5

grid.pattern\_wave, 37

grid.pattern\_wave(), 5

grid.pattern\_weave, 39

grid.pattern\_weave(), 5, 13, 47

grid::gpar(), 36

grid::grid.null(), 5

grid::textGrob(), 36

mean\_col, 41

names\_hex (pattern\_hex), 42

names\_magick (grid.pattern\_magick), 17

names\_magick\_intensity (grid.pattern\_magick), 17

names\_magick\_stripe (grid.pattern\_magick), 17

names\_pattern (grid.pattern), 3

names\_placeholder (grid.pattern\_placeholder), 21

names\_polygon\_tiling (grid.pattern\_polygon\_tiling), 25

names\_square (pattern\_square), 43

names\_weave (pattern\_weave), 45

pattern\_hex, 42

pattern\_hex(), 10, 20, 29, 31, 36

pattern\_square, 43

pattern\_square(), 10, 20, 29, 31, 36

pattern\_weave, 45

`pattern_weave()`, [10](#), [20](#), [29](#), [31](#), [36](#), [40](#), [44](#),  
[45](#)  
`patternGrob (grid.pattern)`, [3](#)  
`star_angle (star_scale)`, [47](#)  
`star_scale`, [47](#)