Description

The `.dot` functions creates functions that allows relative-like specification of paths, but are safe from changing working directory.

Usage

```r
.dot(x, root = getwd(), mustExist = FALSE, relative = FALSE, create = TRUE)

.dot2(names, quiet = FALSE, ...)
```
annotate_y_axis

Arguments

x File path that is appended to BASEDIR.
root Root of your working directory, from which x is relative too.
mustExist Logical value; if TRUE and the resulting path does not exist, it raises an error.
relative For .dot, sets default for the returned function. For the returned function, when TRUE, the function returns a path relative to root.
create Logical values, creates the target directory when TRUE (default).
names Character vector of names
quiet Logical value, suppresses output to stdout() when TRUE.
... Arguments passed on to .dot.

Value

A function that returns file paths constructed from root, x, and ....

Side effect: It creates the directory.

Examples

## Not run:

## Not run:

# data <- .dot('data')
# .data('input.txt')
# .data(c('a.txt','b.txt'))

# .dot2(c('rawdata','results'))
# .rawdata('rawfile.csv')
# .results('myresults.txt')

# End(Not run)

__annotate_y_axis__ __Annotations in the axis__

Description

Annotations in the axis

Usage

annotate_y_axis(
  label,
  y,
  side = waiver(),
  print_label = TRUE,
  print_value = TRUE,
)
annotate_y_axis

```r
print_both = TRUE,
parsed = FALSE,
...
)
annotate_x_axis(
  label,
  x,
  side = waiver(),
  print_label = TRUE,
  print_value = TRUE,
  print_both = TRUE,
parsed = FALSE,
...
)
```

Arguments

- **label**: Text to print
- **y, x**: Position of the annotation.
- **side**: left or right, or top or bottom side to print annotation
- **print_label, print_value, print_both**
  - Logical; what to show on annotation. Label and/or value. print_both is shortcut for setting both print_label and print_value. When both is TRUE, uses argument sep to separate the label and value.
- **parsed**
  - Logical (default FALSE), when TRUE, uses mathplot for outputting expressions. See section "Showing values".
- **...**: Style settings for label and tick: colour, hjust, vjust, size, fontface, family, rot. When waiver() (default), the relevant theme element is used.

Showing values: See plotmath for using mathematical expressions. The function uses a simple replacement strategy where the literal strings .(y) and .(val) are replaced by the value after rounding to a number of digits, as given by argument digits.

Examples

```r
library(ggplot2)
p <- ggplot(mtcars, aes(mpg, hp, colour=disp)) + geom_point()

l <- p + annotate_y_axis("mark at", y=200, tick=TRUE)
l
(l + annotate_x_axis("good economy ->", x=25, print_value=FALSE, hjust=0, tick=TRUE))

l + annotate_y_axis("x^2 == .(y)", y=180, parsed=FALSE, tick=FALSE) +
annotate_y_axis("x^2 + bar(x) == .(y)", y=mean(mtcars$hp), parsed=TRUE, tick=TRUE)

l + annotate_y_axis("bar(x) == .(y)", y = mean(mtcars$hp), parsed=TRUE, tick=FALSE)
```

# use double equal signs, or the output becomes '!=(...)' for some reason.

l + annotate_y_axis('this is midway', y=sum(range(mtcars$hp))/2, print_value = FALSE, side='left')

# work around if an axis only contains parsed expressions
p + annotate_y_axis('some long string', y=100, tick=FALSE, print_value=FALSE, colour=NA) +
   annotate_y_axis("bar(x) == .(y)", y = mean(mtcars$hp), parsed=TRUE, tick=FALSE)

# Works together with other functions
p <- p + theme_light() + theme(panel.border=element_blank(),
   axis.line = element_line(),
   axis.ticks = element_line(colour='black'))
p + coord_capped_cart(bottom='right') +
   annotate_y_axis('More than I\ncan afford', y=125,
   print_value=FALSE, tick=TRUE)

---

**brackets_horizontal**  
Axis brackets instead of axis ticks and lines

**Description**
To be used with coord_flex_cart, coord_capped_cart, etc. for displaying brackets instead of the axis ticks and lines.

**Usage**

```r
brackets_horizontal(
  direction = c("up", "down"),
  length = unit(0.05, "npc"),
  tick.length = waiver()
)
```

```r
brackets_vertical(
  direction = c("left", "right"),
  length = unit(0.05, "npc"),
  tick.length = waiver()
)
```

**Arguments**

- **direction**  
  Which way should the opening side of the brackets point? up, down, left, or right?

- **length**  
  Length of the unit, parallel with axis line.

- **tick.length**  
  Height (width) of x-axis (y-axis) bracket. If waiver() (default), use axis.ticks.length from theme.
Details

The looks of the brackets are taken from theme(axis.ticks), or theme(axis.ticks.x) and theme(axis.ticks.y), respectively.

It does not re-calculate tick marks, but lets scale_x_* and scale_y_* calculate and draw ticks and labels, and then modifies the ticks with brackets.

Both length and tick.length accepts a numeric scalar instead of a unit object that is interpreted as an "npc" unit.

See Also

unit

Examples

library(ggplot2)

p <- ggplot(mpg, aes(as.factor(cyl), hwy, colour=class)) +
  geom_point(position=position_jitter(width=0.3)) +
  theme_bw() +
  theme(panel.border = element_blank(), axis.line = element_line())

p

p <- p + coord_flex_cart(bottom=brackets_horizontal(length=unit(0.08, 'npc')))

p

# However getting the correct width is a matter of tweaking either length or
# position_jitter...

# A further adjustment,
p + theme(panel.grid.major.x = element_blank())

---

coord_capped_cart  Cartesian coordinates with capped axis lines.

Description

Caps the axis lines to the outer ticks to e.g. indicate range of values. Methods correspond to coord_cartesian and coord_flip

Usage

coord_capped_cart(
  xlim = NULL,
  ylim = NULL,
  expand = TRUE,
  top = waiver(),
  left = waiver(),
  bottom = waiver(),
  right = waiver(),
)
coord_capped_cart

    gap = 0.01
  }

coord_capped_flip(
    xlim = NULL,
    ylim = NULL,
    expand = TRUE,
    top = waiver(),
    left = waiver(),
    bottom = waiver(),
    right = waiver(),
    gap = 0.01
  }

capped_horizontal(capped = c("both", "left", "right", "none"), gap = 0.01)

capped_vertical(capped = c("top", "bottom", "both", "none"), gap = 0.01)

Arguments

xlim, ylim Limits for the x and y axes.
expand If TRUE, the default, adds a small expansion factor to the limits to ensure that
data and axes don’t overlap. If FALSE, limits are taken exactly from the data or
xlim/ylim.
top, left, bottom, right
Either a function returned from capped_horizontal or brackets_horizontal. If string, it is assumed to be shorthand for capped_horizontal(capped) or similar for vertical.
gap Both ends are always capped by this proportion. Usually a value between 0 and
1.
capped Which end to cap the line. Can be one of (where relevant): both, none, left,
right, top, bottom.

Details

This function is a simple override of coord_flex_cart and coord_flex_flip, which allows shorthand specification of what to cap.

NB! A panel-border is typically drawn on top such that it covers tick marks, grid lines, and axis
lines. Many themes also do not draw axis lines. To ensure the modified axis lines are visible, use
theme(panel.border=element_blank(),axis.lines=element_line()).

Examples

library(ggplot2)
# Notice how the axis lines of the following plot meet in the lower-left corner.
p <- ggplot(mtcars, aes(x = mpg)) + geom_dotplot() +
    theme_bw() +
    theme(panel.border=element_blank(), axis.line=element_line())
# We can introduce a gap by capping the ends:
p + coord_capped_cart(bottom='none', left='none')

# The lower limit on the y-axis is 0. We can cap the line to this value.
# Notice how the x-axis line extends through the plot when we no long
# define its capping.
p + coord_capped_cart(left='both')

# It it also works on the flipped.
p + coord_capped_flip(bottom='both')

# And on secondary axis, in connuction with brackets:
p +
   scale_y_continuous(sec.axis = sec_axis(~.*100)) +
   scale_x_continuous(sec.axis = sec_axis(~1/., name='Madness scale')) +
   coord_capped_cart(bottom='none', left='none', right='both', top=brackets_horizontal())
# Although we cannot recommend the above madness.

---

**coord_flex_cart**  
*Cartesian coordinates with flexible options for drawing axes*

**Description**

Allows user to inject a function for drawing axes, such as `capped_horizontal` or `brackets_horizontal`.

**Usage**

```r
coord_flex_cart(
  xlim = NULL,
  ylim = NULL,
  expand = TRUE,
  top = waiver(),
  left = waiver(),
  bottom = waiver(),
  right = waiver()
)
```

```r
coord_flex_flip(
  xlim = NULL,
  ylim = NULL,
  expand = TRUE,
  top = waiver(),
  left = waiver(),
  bottom = waiver(),
  right = waiver()
)
```
coord_flex_cart

coord_flex_fixed(
    ratio = 1,
    xlim = NULL,
    ylim = NULL,
    expand = TRUE,
    top = waiver(),
    left = waiver(),
    bottom = waiver(),
    right = waiver()
)

Arguments

xlim, ylim Limits for the x and y axes.
expand If TRUE, the default, adds a small expansion factor to the limits to ensure that
data and axes don't overlap. If FALSE, limits are taken exactly from the data or
xlim/ylim.
top, left, bottom, right Function for drawing axis lines, ticks, and labels, use e.g. capped_horizontal
or brackets_horizontal.
ratio aspect ratio, expressed as y / x.

Details

NB! A panel-border is typically drawn on top such that it covers tick marks, grid lines, and axis
lines. Many themes also do not draw axis lines. To ensure the modified axis lines are visible, use
theme(panel.border=element_blank(), axis.line=element_line()).

User defined functions

The provided function in top, right, bottom, and left defaults to panel_guides_grob which is
defined in 'gR/coord-cartesian.r'.
The provided function is with the arguments scale_details, axis, scale, position, and theme,
and the function should return an absoluteGrob object.
For examples of modifying the drawn object, see e.g. capped_horizontal or brackets_horizontal.

Examples

library(ggplot2)
# A standard plot
p <- ggplot(mtcars, aes(disp, wt)) +
  geom_point() +
  geom_smooth() + theme(panel.border=element_blank(), axis.line=element_line())

# We desire that left axis does not extend beyond '6'
# and the x-axis is unaffected
p + coord_capped_cart(left='top')
# Specifying 'bottom' caps the axis with at most the length of 'gap'
+ coord_capped_cart(left='top', bottom='none')

# We can specify a ridiculous large 'gap', but the lines will always
# protrude to the outer most ticks.
+ coord_capped_cart(left='top', bottom='none', gap=2)

# We can use 'capped_horizontal' and 'capped_vertical' to specify for
# each axis individually.
+ coord_capped_cart(left='top', bottom=capped_horizontal('none', gap=2))

# At this point we might as well drop using the short-hand and go full on:
+ coord_flex_cart(left=brackets_vertical(), bottom=capped_horizontal('left'))

# Also works with secondary axes:
+ scale_y_continuous(sec.axis=sec_axis(~5*., name='wt times 5') +
  coord_flex_cart(left=brackets_vertical(), bottom=capped_horizontal('right'),
  right=capped_vertical('both', gap=0.02))

# Supports the usual 'coord_fixed':
+ coord_flex_fixed(ratio=1.2, bottom=capped_horizontal('right'))

# and coord_flip:
+ coord_flex_flip(ylim=c(2,5), bottom=capped_horizontal('right'))

---

**element_render**

.Render a ggplot2 grob or retrieve its gpar object.

**Description**

Helps add the ggplot2-theme's look-and-feel to grid's grob objects. render_gpar returns a gpar-object, element_render returns a grid.grob-object.

**Usage**

```
 element_render(theme, element, ..., name = NULL)
 render_gpar(theme, element, ...)
```

**Arguments**

- **theme**
  A ggplot2 theme
- **element**
  The name of an element in the theme, e.g. "axis.text".
- **...**
  Additional arguments sent to grobs (e.g. x or y).
- **name**
  Returned grob’s name.
facet_rep_grid

Value

A grid.grob or gpar object.

Author(s)

element_render is from ggplot2 source.

See Also

theme

---

facet_rep_grid  Repeat axis lines and labels across all facet panels

Description

facet_grid and facet_wrap, but with axis lines and labels preserved on all panels.

Usage

facet_rep_grid(..., repeat.tick.labels = FALSE)

facet_rep_wrap(..., scales = "fixed", repeat.tick.labels = FALSE)

Arguments

...  Arguments used for facet_grid or facet_wrap.

repeat.tick.labels
  When FALSE (default), axes on inner panels have their tick labels (i.e. the numbers) removed. Set this to TRUE to keep all labels, or any combination of top, bottom, left, right to keep only those specified. Also accepts 'x' and 'y'.

scales  As for facet_grid, but alters behaviour of repeat.tick.labels.

Details

These two functions are extensions to facet_grid and facet_wrap that keeps axis lines, ticks, and optionally tick labels across all panels.

Examples are given in the vignette "Repeat axis lines on facet panels" vignette.
Description

`geom_pointpath` combines `geom_point` and `geom_path`, such that a) when jittering is used, both lines and points stay connected, and b) provides a visual effect by adding a small gap between the point and the end of line. `geom_pointline` combines `geom_point` and `geom_path`.

Usage

```r
geom_pointpath(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  distance = unit(3, "pt"),
  shorten = 0.5,
  threshold = 0.1,
  lineend = "butt",
  linejoin = "round",
  linemitre = 1,
  linesize = 0.5,
  linecolour = waiver(),
  linecolor = waiver(),
  arrow = NULL,
  ...
)
```

```r
geom_pointline(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  distance = unit(3, "pt"),
  shorten = 0.5,
  threshold = 0.1,
  lineend = "butt",
  linejoin = "round",
  linemitre = 1,
  linesize = 0.5,
  linecolour = waiver(),
  linecolor = waiver(),
  arrow = NULL,
  ...
)
```
geom_pointpath

linecolour = waiver(),
linecolor = waiver(),
arrow = NULL,
...
)

geom_pointangeline(
    mapping = NULL,
data = NULL,
    stat = "identity",
position = "identity",
na.rm = FALSE,
show.legend = NA,
inherit.aes = TRUE,
distance = unit(3, "pt"),
lineend = "butt",
linejoin = "round",
linemitre = 1,
linesize = 0.5,
linecolour = waiver(),
linecolor = waiver(),
arow = NULL,
...
)

Arguments

mapping Set of aesthetic mappings created by aes or aes_

data The data to be displayed in this layer.
stat The statistical transformation to use on the data for this layer, as a string.
position Position adjustment, either as a string, or the result of a call to a position adjust-
ment function (e.g. position_jitter). Both lines and points gets the same ad-
justment (this is where the function excels over geom_point() + geom_line()).
	na.rm If FALSE (default), missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend Logical. Should this layer be included in the legends? NA (default), includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes If FALSE, overrides the default aesthetic, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification.
distance Gap size between point and end of lines; use unit. Is converted to 'pt' if given as simple numeric. When NULL or NA, gapping and shorten/threshold is disabled. To keep the latter, set to 0.
shorten, threshold When points are closer than threshold, shorten the line by the proportion in shorten instead of adding a gap by distance.
geom_pointpath

lineend       Line end style (round, butt, square).
linejoin      Line join style (round, mitre, bevel).
linemitre     Line mitre limit (number greater than 1).
linesize      Width of line.
linecolour, linecolor
              When not waiver(), the line is drawn with this colour instead of that set by aesthetic colour.
arrow         Arrow specification, as created by arrow.
...           other arguments passed on to layer.

Details

gem_pointpath connects the observations in the same order in which they appear in the data. gem_pointline connects them in order of the variable on the x-axis.

Both geom_pointpath and geom_pointline will only connect observations within the same group! However, if linecolour is not waiver(), connections will be made between groups, but possible in an incorrect order.

Aesthetics

gem_pointline and geom_pointpath understands the following aesthetics (required aesthetics are in bold):

• x
• y
• alpha
• colour – sets colour of point. Only affects line if linecolour=waiver().
• stroke
• shape
• stroke
• group
• linetype
• size – only affects point size. Width of line is set with linesize and cannot be linked to an aesthetic.

Examples

# geom_point examples
library(ggplot2)

p <- ggplot(mtcars, aes(wt, mpg))
p + geom_point() + geom_line()
p + geom_pointline()
p + geom_pointline(linecolour='brown')
p + geom_pointpath()

# Add aesthetic mappings
p + geom_pointline(aes(colour = factor(cyl)))
# Using linecolour preserved groups.
p + geom_pointline(aes(colour = factor(cyl)), linecolour='brown')

## If you want to combine the pretty lines of pointline that do *not* respect
## grouping (or order), combine several layers with geom_point on top:
p + geom_pointline() + geom_point(aes(colour=factor(cyl)))

# Change scales
p + geom_pointline(aes(colour = cyl)) + scale_colour_gradient(low = "blue")
p + geom_pointline(aes(colour = cyl), linecolour='black') + scale_colour_gradient(low = "blue")
p + geom_pointline(aes(shape = factor(cyl))) + scale_shape(solid = FALSE)

# For shapes that have a border (like 21), you can colour the inside and
# outside separately. Use the stroke aesthetic to modify the width of the
# border
ggplot(mtcars, aes(wt, mpg)) +
  geom_pointline(shape = 21, colour = "black", fill = "white",
                   size = 5, stroke = 5, distance = unit(10, 'pt'))

## Another example
df <- data.frame(x=rep(c('orange','apple','pear'), each=3),
                 b=rep(c('red','green','purple'), times=3), y=runif(9))
ggplot(df, aes(x=x, y=y, colour=b, group=b)) +
  geom_pointline(linesize=1, size=2, distance=6) + theme_bw()

# geom_pointline() is suitable for time series
ggplot(economics, aes(date, unemploy)) + geom_pointline()
ggplot(economics_long, aes(date, value01, colour = variable)) +
  geom_pointline()

---

**geom_siderange**

Display range of data in side of plot

**Description**
Projects data onto horizontal or vertical edge of panels.

**Usage**

```r
geom_siderange(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ..., 
  distance = 3,
)```

geom_siderange

arrow = NULL,
lineend = "butt",
sides = "bl",
start = NA,
end = NA,
na.rm = FALSE,
show.legend = NA,
inherit.aes = TRUE
}

Arguments

mapping    Set of aesthetic mappings created by aes or aes_.
data        The data to be displayed in this layer.
stat        The statistical transformation to use on the data for this layer, as a string.
position    Position adjustment, either as a string, or the result of a call to a position adjust-
            ment function (e.g. position_jitter). Both lines and points get the same ad-
            justment (this is where the function excels over geom_point() + geom_line()).
            ... other arguments passed on to layer.
distance    Distance between edge of panel and lines, and distance between lines, in multi-
            ples of line widths, see description.
arrow       Arrow specification, as created by arrow.
lineend     Line end style (round, butt, square).
sides       Character including top, right, bottom, and/or left, indicating which side to
            project data onto.
start, end   Adds a symbol to either end of the siderange. start corresponds to minimal
            value, end to maximal value.
na.rm        If FALSE (default), missing values are removed with a warning. If TRUE, missing
            values are silently removed.
show.legend  Logical. Should this layer be included in the legends? NA (default), includes if
            any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes  If FALSE, overrides the default aesthetic, rather than combining with them. This
            is most useful for helper functions that define both data and aesthetics and
            shouldn’t inherit behaviour from the default plot specification.

Details

The geom_siderange projects the data displayed in the panel onto the sides, using the same aes-
thetics. It has the added capability of potting a symbol at either end of the line, and lines are offset
from the edge and each other.

To display a symbol, specify an integer for either start or end. See the list for pch in points
for values to use. The arguments start and end also accepts a list object with named entries pch,
alpha, stroke, and fill, which correspond to the usual aesthetics, as well as a special named
entry, sizer (note the extra ‘r’). This last entry is a multiplier for enlarging the symbol relative to
the linewidth, as the aesthetic size affects both linewidth and symbol size.
The distance between the panel's edge and sideranges are specified by the argument `distance`. If a symbol is specified, the linewidth is further expanded to cover the width of the symbol (including `sizer`).

**Aesthetics**

The geom understands the following aesthetics (required are in bold):

- **x**
- **y**
- **alpha**
- **colour**
- **fill** (if a symbol is applied with `start` or `end`)
- **group**
- **linetype**
- **size**
- **stroke**

**See Also**

`geom_rug`

**Examples**

```r
library(ggplot2)

x <- rnorm(25)
df <- data.frame(x=x, y=x+rnorm(25, sd=0.2),
                 a=sample(c('horse','goat'), 25, replace=TRUE),
                 stringsAsFactors = FALSE)
df$y <- with(df, ifelse(y > 1 & a=='horse', 1, y))
(p <- ggplot(df, aes(x=x, y=y, colour=a)) + geom_point(shape=1))

p + geom_siderange(start=19)

# Capping the sideranges with different symbols:
p + geom_siderange(start=19, end=22, fill='black', sides='b') + geom_siderange(sides='tl')

# It also works with facets

p <- ggplot(mpg, aes(displ, hwy, colour=f1)) +
    geom_point() +
    facet_wrap(~class, nrow = 4)

p + geom_siderange()
```
get_panel_range  
Version safe(r) method to get the y- and x-range from trained scales.

Description

The names of the internal layout objects from `ggplot_build` changed slightly.

Usage

```r
get_panel_y_range(layout, index = 1)
get_panel_x_range(layout, index = 1)
get_panel_params(layout, index = 1)
```

Arguments

- `layout`: layout part from `ggplot_build`
- `index`: Could be panel number?

grid_arrange_shared_legend

*Share a legend between multiple plots*

Description

Extract legend, combines plots using `arrangeGrob` / `grid.arrange`, and places legend in a margin.

Usage

```r
grid_arrange_shared_legend(
    ..., 
    ncol = length(list(...)),
    nrow = 1,
    position = c("bottom", "right", "top", "left"),
    plot = TRUE
)
```

Arguments

- `...`: Objects to plot. First argument should be a `ggplot2` object, as the legend is extracted from this. Other arguments are passed on to `arrangeGrob`, including named arguments that are not defined for `grid_arrange_shared_legend`. `ggplot2` objects have their legends hidden.
- `ncol`: Integer, number of columns to arrange plots in.
gtake_show_grill

**nrow**  
Integer, number of rows to arrange plots in.

**position**  
'bottom' or 'right' for positioning legend.

**plot**  
Logical, when TRUE (default), draws combined plot on a new page.

**Value**

gtable of combined plot, invisibly. Draw gtable object using `grid.draw`.

**Author(s)**

Originally brought to you by Baptiste Auguié (https://github.com/tidyverse/ggplot2/wiki/Share-a-legend-between-two-ggplot2-graphs) and Shaun Jackman (original). Stefan McKinnon Edwards added left and top margins.

**See Also**

g_legend, reposition_legend

**Examples**

```r
library(ggplot2)
dsamp <- diamonds[sample(nrow(diamonds), 300), ]
p1 <- qplot(carat, price, data = dsamp, colour = clarity)
p2 <- qplot(cut, price, data = dsamp, colour = clarity)
p3 <- qplot(color, price, data = dsamp, colour = clarity)
p4 <- qplot(depth, price, data = dsamp, colour = clarity)
ggrid_arrange_shared_legend(p1, p2, p3, p4, ncol = 4, nrow = 1)
ggrid_arrange_shared_legend(p1, p2, p3, p4, ncol = 2, nrow = 2)

# Passing on plots in a grob are not touched
grid_arrange_shared_legend(p1, gridExtra::arrangeGrob(p2, p3, p4, ncol=3), ncol=1, nrow=2)

# We can also pass on named arguments to arrangeGrob:
title <- grid::textGrob('This is grob', gp=grid::gpar(fontsize=14, fontface='bold'))
nt <- theme(legend.position='none')
ggrid_arrange_shared_legend(p1,
  gridExtra::arrangeGrob(p2+nt, p3+nt, p4+nt, ncol=3), ncol=1, nrow=2,
  top=title)
```

**gtake_show_grill**

Visualise underlying gtable layout.

**Description**

Visualises the table structure or the names of the gtable’s components.
Usage

gtable_show_grill(x, plot = TRUE)

gtable_show_names(
  x,
  plot = TRUE,
  rect.gp = grid::gpar(col = "black", fill = "white", alpha = 1/4)
)

Arguments

x A gtable object. If given a ggplot object, it is converted to a gtable object with 
ggplotGrob.

plot Logical. When TRUE (default), draws resulting gtable object on a new page.#'

rect.gp Graphical parameters (gpar) for background drop.

Details

These functions are highly similar to gtable_show_layout. gtable_show_grill draws the grid
of the underlying table, and places row and column indicies in the margin. gtable_show_names
replaces the grobs with a semi-transparent rectangle and the component’s name.

Value

Modified gtable object, invisibly.

Examples

library(ggplot2)
library(gtable)
library(grid)

p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
gtable_show_grill(p)
library(ggplot2)
library(gtable)
library(grid)

p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
gtable_show_names(p)
Guidebox as a column

Description

Takes a plot or legend and returns a single guide-box in a single column, for embedding in e.g. tables.

Usage

guidebox_as_column(legend, which.legend = 1, add.title = FALSE)

Arguments

legend A ggplot2 plot or the legend extracted with g_legend. Do not provide a ggplotGrob as it is indistinguishable from a legend.

which.legend Integer, a legend can contain multiple guide-boxes (or vice versa?). Use this argument to select which to use.

add.title Does nothing yet.

Value

A gtable with keys and labels reordered into a single column and each pair of keys and labels in the same cell.

See Also

g_legend

Examples

library(ggplot2)
library(dplyr)

p <- ggplot(diamonds, aes(x=x, y=y, colour=cut)) + geom_point()
guidebox_as_column(p)
p <- p + guides(colour=guide_legend(ncol=2, byrow=TRUE))
guidebox_as_column(p)
**g_legend**  
*Extract ggplot legends*

**Description**

Extracts the legend (‘guide-box’) from a ggplot2 object.

**Usage**

```r
g_legend(a.gplot)
```

**Arguments**

- `a.gplot`: ggplot2 or gtable object.

**Details**

The extraction is applied after the plot is trained and themes are applied. Modifying the legend is easiest by applying themes etc. to the ggplot2 object, before calling `g_legend`.

An alternative method for extracting the legend is using `gtable::gtable_filter`:

```r
gtable_filter(ggplotGrob(a.ggplot.obj), 'guide-box')
```

This method however returns a gtable object which encapsulates the entire legend. The legend itself may be a collection of gtable. We have only noticed a problem with this extra layer when using the returned legend with `arrangeGrob` (see examples).

**Value**

gtable (grob) object. Draw with `grid.draw`.

**Author(s)**

Baptiste Auguié

**See Also**

`grid_arrange_shared_legend`, `reposition_legend`, `gtable_filter`

**Examples**

```r
library(ggplot2)
library(gtable)
library(grid)
library(gridExtra)
library(gtable)
dsamp <- diamonds[sample(nrow(diamonds), 1000), ]
(d <- ggplot(dsamp, aes(carat, price)) +
```
is.small

Is a given unit 'small'?

Description

Uses a holistic approach to determine whether a unit is 'small', i.e. less than 1 cm, 1 line, 10 pt, or 0.4 in.

Usage

is.small(x)

Arguments

x A unit.

Details

Based on arbitrarily chosen definitions of 'small', this function can return TRUE or FALSE if a unit is 'small'.

So far, less than 1 cm, 1 line, 10 pt, or 0.4 inches is defined as being 'small'. Unresolved sizes, such as 'grobheight', 'grobwidth', or 'null' are not small. Units based on arithmetic, such as sum of multiple units, are also not small. NAs are returned for undecided sizes.
Value

Logical or NA.

Description

Collection of misc. functions for changing subtle aspects of ggplots. Works mostly on gtables produced prior to printing.

Functions for axis

See `coord_capped_cart` and `coord_flex_cart`. The latter is a shorthand version of the former. It automatically uses `capped_horizontal` and `capped_vertical`, but both accepts these as well as `brackets_horizontal` and `brackets_vertical`.

Legends

Extract legend `g_legend`

Many plots, one legend `grid_arrange_shared_legend`

Place legend exactly on plot `reposition_legend`

Facets

`facet_rep_grid` and `facet_rep_wrap` are extensions to the wellknown `facet_grid` and `facet_wrap` where axis lines and labels are drawn on all panels.

Extending knitr

We automatically load knitr’s `knit_print` for data frames and dplyr tables to provide automatic pretty printing of data frame using `kable`.

See `lemon_print` or `vignette('lemon_print','lemon')`.

Relative paths safe from hanging directory: `.dot`.

Author(s)

Stefan McKinnon Edwards <sme@iysik.com>

Contributions from Baptiste Auguié on `g_legend` and `grid_arrange_shared_legend`.

Contributions from Shaun Jackman on `grid_arrange_shared_legend`.

Source

https://github.com/stefanedwards/lemon
lemon_print

See Also

Useful links:

- [https://github.com/stefanedwards/lemon](https://github.com/stefanedwards/lemon)
- Report bugs at [https://github.com/stefanedwards/lemon/issues](https://github.com/stefanedwards/lemon/issues)

**Description**

Convenience function for working with R Notebooks that ensures data frames (and dplyr tables) are printed with `kable` while allowing RStudio to render the data frame dynamically for inline display.

**Usage**

```r
lemon_print(x, options, ...)
```

## S3 method for class `data.frame`

```r
lemon_print(x, options, ...)
```

## S3 method for class `table`

```r
lemon_print(x, options, ...)
```

**Arguments**

- `x` an data frame or dplyr table object to be printed
- `options` Current chunk options are passed through this argument.
- `...` Ignored for now.

**Details**

These functions divert data frame and summary output to `kable` for nicely printing the output.

For `options` to `kable`, they can be given directly as chunk-options (see arguments to `kable`), or through as a list to a special chunk-option `kable.opts`.

For more examples, see vignette('lemon_print',package='lemon').

**Knitr usage**

To use for a single chunk, do

```r
```
Note: We are not calling the function, but instead refering to it.
An alternate route for specifying `kable` arguments is as:

```r
'\{r render=l lemon_print,kable.opts=list(align='l')}\n
data.frame
'```

The option `kable.opts` takes precedence over arguments given directly as chunk-options.
To enable as default printing method for all chunks, include

```r
knit_print.data.frame <- lemon_print
knit_print.table <- lemon_print
knit_print.grouped_df <- lemon_print # enableds dplyr results
knit_print.tibble <- lemon_print
knit_print.tbl <- lemon_print
```

Note: We are not calling the function, but instead assigning the `knit_print` functions for some classes.
To disable, temporarily, specify chunk option:

```r
'\{r render=normal_print}'
data.frame
'```

See Also

`knit_print`, `kable`

---

**remove_labels_from_axis**

*Removes labels from axis grobs.*

### Description

Called from `FacetGridRepeatLabels`.

### Usage

```r
remove_labels_from_axis(axisgrob)
```

### Arguments

- `axisgrob`: Grob with an axis.
reposition_legend  Reposition a legend onto a panel

Description

Repositions a legend onto a panel, by either taking it from the same ggplot, or by using another. Works on both ggplot2 and gtable objects, and can accept any grob as legend.

Usage

reposition_legend(
  aplot,  # a ggplot2 or gtable object.
  position = NULL,  
  legend = NULL,  
  panel = "panel",  
  x = NULL,  
  y = NULL,  
  just = NULL,  
  name = "guide-box",  
  clip = "on",  
  offset = c(0, 0),  
  z = Inf,  
  plot = TRUE  
)

Arguments

aplot  a ggplot2 or gtable object.
position Where to place the legend in the panel. Overrules just argument.
legend The legend to place, if NULL (default), it is extracted from aplot if this is a ggplot2 object.
panel Name of panel in gtable. See description.
x horizontal coordinate of legend, with 0 at left.
y vertical coordinate of legend, with 0 at bottom.
just ‘Anchor point’ of legend; it is this point of the legend that is placed at the x and y coordinates.
name, clip Parameters forwarded to gtable_add_grob.
offset Numeric vector, sets distance from edge of panel. First element for horizontal distance, second for vertical. Not used by arguments x and y.
z Z-level of legend. When Inf (default), legend is placed just under the axis-lines. If you want it on top of everything, use a sufficient large number, but not Inf.
plot Logical, when TRUE (default), draws plot with legend repositioned on a new page.
Details

To modify the look of the legend, use themes and the natural ggplot functions found in `guide_legend`

*Positioning* is done by argument `position` which places the panel relative in `panel` (see below). position resolves to three variables, x, y, and `just`. x and y is the coordinate in panel, where the anchorpoint of the legend (set via `just`) is placed. In other words, `just='bottom right'` places the bottom right corner of the legend at coordinates (x,y).

The positioning can be set by argument `position` alone, which can be further nudged by setting position, x, and y. Alternatively, manually positioning can be obtained by setting arguments. x, y, and `just`.

*Panel* name is by default `panel`, but when using facets it typically takes the form `panel-{col}-{row}`, but not for wrapped facets. Either print result from `ggplotGrob` or use `gtable_show_names` to display all the names of the gtable object.

panel takes multiple names, and will then use these components’ extremes for placing the legend.

If panel is an integer vector of length 2 or 4, these elements are used directly for top-left and bottom-right coordinates.

Value

gtable object, invisibly, with legend repositioned. Can be drawn with `grid.draw`.

Author(s)

Stefan McKinnon Edwards <sme@iysik.com>

See Also

g_legend, grid_arrange_shared_legend

Examples

library(ggplot2)

dsamp <- diamonds[sample(nrow(diamonds), 1000), ]

(d <- ggplot(dsamp, aes(carat, price)) +
  geom_point(aes(colour = clarity)))

reposition_legend(d + theme(legend.position='bottom'), 'bottom right')

# To change the orientation of the legend, use theme’s descriptors.
reposition_legend(d + theme(legend.position='bottom'), 'top left')

# Use odd specifications, here offset the legend with half its height from the bottom.
reposition_legend(d + theme(legend.position='bottom'), x=0.3, y=0, just=c(0, -0.5))

# For using with facets:
reposition_legend(d + facet_grid(~cut), 'top left', panel = 'panel-3-1')
scale_x_symmetric

Symmetrix position scale for continuous x and y

Description

scale_x_symmetric and scale_y_symmetric are like the default scales for continuous x and y, but ensures that the resulting scale is centered around mid. Does not work when setting limits on the scale.

Usage

scale_x_symmetric(mid = 0, ...)
scale_y_symmetric(mid = 0, ...)

Arguments

mid Value to center the scale around.
...

Values passed on to scale_continuous.

Examples

library(ggplot2)
df <- expand.grid(a = c(-1, 0, 1), b = c(-1, 0, 1))
rnorm2 <- function(x, y, n, sdx, sdy) {
  if (missing(sdy))
    sdy <- sdx
  data.frame(a = x, b = y, x = rnorm(n, x, sdx), y = rnorm(n, y, sdy))
}
df <- mapply(rnorm2, df$a, df$b, MoreArgs = list(n = 30, sdx = 1), SIMPLIFY = FALSE)
df <- do.call(rbind, df)
(p <- ggplot(df, aes(x = x, y = y)) + geom_point() +
  facet_grid(a ~ b, scales = 'free_x'))
}
p + scale_x_symmetric(mid = 0)
Index

*Topic interal

  remove_labels_from_axis, 26
  .dot, 2, 24
  .dot2 (.dot), 2

  absoluteGrob, 9
  aes, 13, 16
  aes_, 13, 16
  annotate_x_axis (annotate_y_axis), 3
  annotate_y_axis, 3
  arrangeGrob, 18, 22
  arrow, 14, 16

  brackets_horizontal
    (brackets_horizontal), 5
  brackets_horizontal, 5, 7–9, 24
  brackets_vertical, 24
  brackets_vertical
    (brackets_horizontal), 5

  capped_horizontal (coord_capped_cart), 6
  capped_horizontal, 7–9, 24
  capped_horizontal (coord_capped_cart), 6
  capped_vertical, 24
  capped_vertical (coord_capped_cart), 6
  coord_capped_cart, 5, 6, 24
  coord_capped_flip (coord_capped_cart), 6
  coord_cartesian, 6
  coord_flex_cart, 5, 7, 8, 24
  coord_flex_fixed (coord_flex_cart), 8
  coord_flex_flip, 7
  coord_flex_flip (coord_flex_cart), 8
  coord_flip, 6

  element_render, 10

  facet_grid, 11, 24
  facet_rep_grid, 11, 24
  facet_rep_wrap, 24
  facet_rep_wrap (facet_rep_grid), 11
  facet_wrap, 11, 24

  g_legend, 19, 21, 22, 24, 28
  geom_path, 12
  geom_point, 12
  geom_pointline (geom_pointpath), 12
  geom_pointpath, 12
  geom_pointrangeline (geom_pointpath), 12
  geom_rug, 17
  geom_siderange, 15
  get_panel_params (get_panel_range), 18
  get_panel_range, 18
  get_panel_x_range (get_panel_range), 18
  get_panel_y_range (get_panel_range), 18
  ggplotGrob, 20, 21, 28
  gpar, 10, 11, 20
  grid.draw, 19, 22, 28
  grid.grob, 10, 11
  grid Arrange_shared_legend, 18, 22, 24, 28
  gtable, 21
  gtable_add_grob, 27
  gtable_filter, 22
  gtable_show_grill, 19
  gtable_show_layout, 20
  gtable_show_names, 28
  gtable_show_names (gtable_show_grill), 19
  guide_legend, 28
  guidebox_as_column, 21

  is_small, 23

  kable, 24–26
  knit_print, 24, 26

  layer, 14, 16
  lemon, 24
  lemon-package (lemon), 24
  lemon_print, 24, 25

  plotmath, 4
points, 16
position_jitter, 13, 16

remove_labels_from_axis, 26
render_gpar(element_render), 10
reposition_legend, 19, 22, 24, 27

scale_continuous, 29
scale_x_symmetric, 29
scale_y_symmetric(scale_x_symmetric), 29

theme, 5, 10, 11

unit, 6, 13