Package ‘patchwork’

January 8, 2024

Type Package

Title The Composer of Plots

Version 1.2.0

Maintainer Thomas Lin Pedersen <`thomasp85@gmail.com`>

Description The 'ggplot2' package provides a strong API for sequentially building up a plot, but does not concern itself with composition of multiple plots. 'patchwork' is a package that expands the API to allow for arbitrarily complex composition of plots by, among others, providing mathematical operators for combining multiple plots. Other packages that try to address this need (but with a different approach) are 'gridExtra' and 'cowplot'.

License MIT + file LICENSE

Encoding UTF-8

Imports ggplot2 (>= 3.0.0), gtable, grid, stats, grDevices, utils, graphics, rlang, cli

RoxygenNote 7.2.3

URL `https://patchwork.data-imaginist.com`,
`https://github.com/thomasp85/patchwork`

BugReports `https://github.com/thomasp85/patchwork/issues`

Suggests knitr, rmarkdown, gridGraphics, gridExtra, ragg, testthat (>= 2.1.0), vdiffr, covr, png

VignetteBuilder knitr

Config/Needs/website gifski

NeedsCompilation no

Author Thomas Lin Pedersen [cre, aut] (<`https://orcid.org/0000-0002-5147-4711`>)

Repository CRAN

Date/Publication 2024-01-08 14:40:02 UTC

R topics documented:

`area` *Specify a plotting area in a layout*

Description

This is a small helper used to specify a single area in a rectangular grid that should contain a plot. Objects constructed with `area()` can be concatenated together with `c()` in order to specify multiple areas.

Usage

```
area(t, l, b = t, r = l)
```

Arguments

`t, b`	The top and bottom bounds of the area in the grid
`l, r`	The left and right bounds of the area int the grid

Details

The grid that the areas are specified in reference to enumerate rows from top to bottom, and coloumns from left to right. This means that `t` and `l` should always be less or equal to `b` and `r` respectively. Instead of specifying area placement with a combination of `area()` calls, it is possible to instead pass in a single string

```
areas <- c(area(1, 1, 2, 1),
           area(2, 3, 3, 3))
```

is equivalent to

```
areas < -"A##
          A#B
          ##B"
```

For an example of this, see the `plot_layout()` examples.

Value

A `patch_area` object

Examples

```
library(ggplot2)

p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) + geom_bar(aes(gear)) + facet_wrap(~cyl)

layout <- c(
  area(1, 1),
  area(1, 3, 3),
  area(3, 1, 3, 2)
)

# Show the layout to make sure it looks as it should
plot(layout)

# Apply it to a patchwork
p1 + p2 + p3 + plot_layout(design = layout)
```

`free` *Free a plot from alignment*

Description

While the purpose of patchwork is often to align plots by their panels, sometimes this doesn't cut it and we want to compose plots without alignment. The `free()` function tells patchwork to treat the content (which can either be a ggplot or a patchwork) specially and not align it with the remaining panels in the composition. It works much like using `wrap_elements()` but has a few niceties. For starter, it is less verbose, both with a shorter name, but also without the need to use the `full` argument rather than the first. Second, A plot wrapped with `free()` retains all of it's behavior from before. You can still add stuff to it, change it's theme, etc., but more importantly you can still collect guides and recurse tags as usual. A further nicety is that margins of the plot behave as expected and is aligned with the other plots in the composition.

Usage

```
free(x)
```

Arguments

x	A ggplot or patchwork object

Value

A modified version of x with a `free_plot` class

Examples

```
# Sometimes you have a plot that defies good composition alginment, e.g. due
# to long axis labels
library(ggplot2)
p1 <- ggplot(mtcars) +
  geom_bar(aes(y = factor(gear), fill = factor(gear))) +
  scale_y_discrete(
    "",
    labels = c("3 gears are often enough",
               "But, you know, 4 is a nice number",
               "I would def go with 5 gears in a modern car")
  )

# When combined with other plots it ends up looking bad
p2 <- ggplot(mtcars) + geom_point(aes(mpg, disp))

p1 / p2

# We can fix this be using free
free(p1) / p2

# We can still collect guides like before
free(p1) / p2 + plot_layout(guides = "collect")
```

`guide_area` *Add an area to hold collected guides*

Description

Using the `guides` argument in `plot_layout()` you can collect and collapse guides from plots. By default these guides will be put on the side like with regular plots, but by adding a `guide_area()` to the plot you can tell patchwork to place the guides in that area instead. If guides are not collected or no guides exists to collect it behaves as a standard `plot_spacer()` instead.

Usage

```
guide_area()
```

Examples

```
library(ggplot2)
p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp, colour = factor(gear)))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) + geom_bar(aes(gear)) + facet_wrap(~cyl)

# Guides are by default kept beeside their plot
p1 + p2 + p3

# They can be collected and placed on the side (according to the patchwork
# theme)
p1 + p2 + p3 + plot_layout(guides = 'collect', ncol = 2)

# Using guide_area() you can also designate an empty area for this
p1 + p2 + p3 + guide_area() + plot_layout(guides = 'collect')
```

`inset_element` *Create an inset to be added on top of the previous plot*

Description

The standard approach of patchwork is to place plots next to each other based on the provided layout. However, it may sometimes be beneficial to place one or several plots or graphic elements freely on top or below another plot. The `inset_element()` function provides a way to create such insets and gives you full control over placement.

Usage

```
inset_element(
  p,
  left,
  bottom,
  right,
  top,
  align_to = "panel",
  on_top = TRUE,
  clip = TRUE,
  ignore_tag = FALSE
)
```

Arguments

`p`	A grob, ggplot, patchwork, formula, raster, or nativeRaster object to add as an inset
`left, bottom, right, top`	numerics or units giving the location of the outer bounds. If given as numerics they will be converted to `npc` units.

`align_to`	Specifies what `left`, `bottom`, etc should be relative to. Either `'panel'` (default), `'plot'`, or `'full'`.
`on_top`	Logical. Should the inset be placed on top of the other plot or below (but above the background)?
`clip`	Logical. Should clipping be performed on the inset?
`ignore_tag`	Logical. Should autotagging ignore the inset?

Value

A `inset_path` object

Examples

```
library(ggplot2)
p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))

# Basic use
p1 + inset_element(p2, 0.6, 0.6, 1, 1)

# Align to the full area instead
p1 + inset_element(p2, 0, 0.6, 0.4, 1, align_to = 'full')

# Grobs and other objects can be added as insets as well
p1 + inset_element(grid::circleGrob(), 0.4, 0.4, 0.6, 0.6)

if (requireNamespace('png', quietly = TRUE)) {
  logo <- system.file('help', 'figures', 'logo.png', package = 'patchwork')
  logo <- png::readPNG(logo, native = TRUE)
  p1 + inset_element(logo, 0.8, 0.8, 1, 1, align_to = 'full')
}

# Just as expected insets are still amenable to changes after the fact
p1 +
  inset_element(p2, 0.6, 0.6, 1, 1) +
  theme_classic()

# Tagging also continues to work as expected
p1 +
  inset_element(p2, 0.6, 0.6, 1, 1) +
  plot_annotation(tag_levels = '1')

# but can be turned off, like for wrapped plots
p1 +
  inset_element(p2, 0.6, 0.6, 1, 1, ignore_tag = TRUE) +
  plot_annotation(tag_levels = '1')
```

`multipage_align` *Align plots across multiple pages*

Description

Sometimes it is necessary to make sure that separate plots are aligned, with each other, but still exists as separate plots. That could e.g. be if they need to be part of a slideshow and you don't want titles and panels jumping around as you switch between slides. patchwork provides a range of utilities to achieve that. Currently it is only possible to align ggplots, but aligning patchworks will be supported in the future.

Usage

```
get_dim(plot)

set_dim(plot, dim)

get_max_dim(...)

align_patches(...)
```

Arguments

`plot`	A ggplot object
`dim`	A plot_dimension object as created by `get_dim()`
`...`	ggplot objects or a single list of them

Value

`get_dim()` and `get_max_dim()` return a plot_dimension object. `set_dim()` returns a modified ggplot object with fixed outer dimensions and `align_patches()` return a list of such. The modified ggplots still behaves like a standard ggplot and new layers, scales, etc can be added to them.

Examples

```
library(ggplot2)
p1 <- ggplot(mtcars) +
  geom_point(aes(mpg, disp)) +
  ggtitle('Plot 1')

p2 <- ggplot(mtcars) +
  geom_boxplot(aes(gear, disp, group = gear)) +
  ggtitle('Plot 2')

p3 <- ggplot(mtcars) +
  geom_point(aes(hp, wt, colour = mpg)) +
  ggtitle('Plot 3')
```

```
p4 <- ggplot(mtcars) +
  geom_bar(aes(gear)) +
  facet_wrap(~cyl) +
  ggtitle('Plot 4')

# Align a plot to p4
p4_dim <- get_dim(p4)
set_dim(p1, p4_dim)

# Align a plot to the maximum dimensions of a list of plots
max_dims <- get_max_dim(p1, p2, p3, p4)
set_dim(p2, max_dims)

# Align a list of plots with each other
aligned_plots <- align_patches(p1, p2, p3, p4)
aligned_plots[[3]]

# Aligned plots still behave like regular ggplots
aligned_plots[[3]] + theme_bw()
```

---

plot_annotation *Annotate the final patchwork*

Description

The result of this function can be added to a patchwork using + in the same way as `plot_layout()`, but unlike `plot_layout()` it will only have an effect on the top level plot. As the name suggests it controls different aspects of the annotation of the final plot, such as titles and tags.

Usage

```
plot_annotation(
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag_levels = NULL,
  tag_prefix = NULL,
  tag_suffix = NULL,
  tag_sep = NULL,
  theme = NULL
)
```

Arguments

`title, subtitle, caption`	Text strings to use for the various plot annotations.

`tag_levels`	A character vector defining the enumeration format to use at each level. Possible values are `'a'` for lowercase letters, `'A'` for uppercase letters, `'1'` for numbers, `'i'` for lowercase Roman numerals, and `'I'` for uppercase Roman numerals. It can also be a list containing character vectors defining arbitrary tag sequences. If any element in the list is a scalar and one of `'a'`, `'A'`, `'1'`, `'i`, or `'I'`, this level will be expanded to the expected sequence.
`tag_prefix, tag_suffix`	Strings that should appear before or after the tag.
`tag_sep`	A separator between different tag levels
`theme`	A ggplot theme specification to use for the plot. Only elements related to the titles as well as plot margin and background is used.

Details

Tagging of subplots is done automatically and following the order of the plots as they are added. When the plot contains nested layouts the `tag_level` argument in the nested plot_layout will define whether enumeration should continue as usual or add a new level. The format of the levels are defined with `tag_levels` argument in `plot_annotation`

Value

A `plot_annotation` object

Examples

```
library(ggplot2)

p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) + geom_bar(aes(gear)) + facet_wrap(~cyl)

# Add title, etc. to a patchwork
p1 + p2 + plot_annotation('This is a title', caption = 'made with patchwork')

# Change styling of patchwork elements
p1 + p2 +
  plot_annotation(
    title = 'This is a title',
    caption = 'made with patchwork',
    theme = theme(plot.title = element_text(size = 16))
  )

# Add tags to plots
p1 / (p2 | p3) +
  plot_annotation(tag_levels = 'A')

# Add multilevel tagging to nested layouts
p1 / ((p2 | p3) + plot_layout(tag_level = 'new')) +
  plot_annotation(tag_levels = c('A', '1'))
```

```
# Use a custom tag sequence (mixed with a standard one)
p1 / ((p2 | p3) + plot_layout(tag_level = 'new')) +
  plot_annotation(tag_levels = list(c('&', '%'), '1'))
```

---

plot_arithmetic *Plot arithmetic*

Description

In addition to the + operator known in ggplot2, patchwork defines logic for some of the other operators that aids in building up your plot composition and reduce code-reuse.

Usage

```
## S3 method for class 'ggplot'
e1 - e2

## S3 method for class 'ggplot'
e1 / e2

## S3 method for class 'ggplot'
e1 | e2

## S3 method for class 'gg'
e1 * e2

## S3 method for class 'gg'
e1 & e2
```

Arguments

e1	A ggplot or patchwork object
e2	A ggplot or patchwork object in case of /, or a gg object such as a geom or theme specification in case of * and &

Details

patchwork augment the + operator from ggplot2 and allows the user to add full ggplot objects together in order to compose them into the same view. The last added plot is always the active one where new geoms etc. are added to. Another operator that is much like it, but not quite, is -. It also adds plots together but instead of adding the right hand side to the patchwork defined in the left hand side, it puts the left hand side besides the right hand side in a patchwork. This might sound confusing, but in essence - ensures that the right and left side are put in the same nesting level (+ puts the right side *into* the left side). Using - might seem unintuitive if you think of the operator as "subtract", but look at it as a hyphen instead (the underlying reason is that - is the only operator in the same precedence group as +).

Often you are interested in creating single column or single row layouts. `patchwork` provides `|` (besides) and `/` (over) operators to support stacking and packing of plots. See the examples for their use.

In order to reduce code repetition `patchwork` provides two operators for adding ggplot elements (geoms, themes, facets, etc.) to multiple/all plots in a patchwork. `*` will add the element to all plots in the current nesting level, while `&` will recurse into nested patches.

Value

A `patchwork` object

Examples

```
library(ggplot2)

p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) + geom_bar(aes(gear)) + facet_wrap(~cyl)
p4 <- ggplot(mtcars) + geom_bar(aes(carb))

# Standard addition vs division
p1 + p2 + p3 + plot_layout(ncol = 1)
p1 + p2 - p3 + plot_layout(ncol = 1)

# Stacking and packing
(p1 | p2 | p3) /
      p4

# Add elements to the same nesting level
(p1 + (p2 + p3) + p4 + plot_layout(ncol = 1)) * theme_bw()

# Recurse into nested plots as well
(p1 + (p2 + p3) + p4 + plot_layout(ncol = 1)) & theme_bw()
```

`plot_layout` *Define the grid to compose plots in*

Description

To control how different plots are laid out, you need to add a layout specification. If you are nesting grids, the layout is scoped to the current nesting level.

Usage

```
plot_layout(
  ncol = NULL,
  nrow = NULL,
  byrow = NULL,
  widths = NULL,
  heights = NULL,
  guides = NULL,
  tag_level = NULL,
  design = NULL,
  axes = NULL,
  axis_titles = axes
)
```

Arguments

`ncol, nrow`	The dimensions of the grid to create - if both are `NULL` it will use the same logic as facet_wrap() to set the dimensions
`byrow`	Analogous to `byrow` in matrix(). If `FALSE` the plots will be filled in in column-major order
`widths, heights`	The relative widths and heights of each column and row in the grid. Will get repeated to match the dimensions of the grid. The special value of `NA/-1null` will behave as `1null` unless a fixed aspect plot is inserted in which case it will allow the dimension to expand or contract to match the aspect ratio of the content
`guides`	A string specifying how guides should be treated in the layout. `'collect'` will collect guides below to the given nesting level, removing duplicates. `'keep'` will stop collection at this level and let guides be placed alongside their plot. `auto` will allow guides to be collected if a upper level tries, but place them alongside the plot if not. If you modify default guide "position" with theme(legend.position=...) while also collecting guides you must apply that change to the overall patchwork (see example).
`tag_level`	A string (`'keep'` or `'new'`) to indicate how auto-tagging should behave. See `plot_annotation()`.
`design`	Specification of the location of areas in the layout. Can either be specified as a text string or by concatenating calls to `area()` together. See the examples for further information on use.
`axes`	A string specifying how axes should be treated. `'keep'` will retain all axes in individual plots. `'collect'` will remove duplicated axes when placed in the same run of rows or columns of the layout. `'collect_x'` and `'collect_y'` will remove duplicated x-axes in the columns or duplicated y-axes in the rows respectively.
`axis_titles`	A string specifying how axis titltes should be treated. `'keep'` will retain all axis titles in individual plots. `'collect'` will remove duplicated titles in one direction and merge titles in the opposite direction. `'collect_x'` and `'collect_y'` control this for x-axis titles and y-axis titles respectively.

Value

A `plot_layout` object to be added to a `ggassmble` object

Examples

```
library(ggplot2)

p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) + geom_bar(aes(gear)) + facet_wrap(~cyl)
p4 <- ggplot(mtcars) + geom_bar(aes(carb))
p5 <- ggplot(mtcars) + geom_violin(aes(cyl, mpg, group = cyl))

# The plots are layed out automatically by default
p1 + p2 + p3 + p4 + p5

# Use byrow to change how the grid is filled out
p1 + p2 + p3 + p4 + p5 + plot_layout(byrow = FALSE)

# Change the grid dimensions
p1 + p2 + p3 + p4 + p5 + plot_layout(ncol = 2, widths = c(1, 2))

# Define layout at different nesting levels
p1 +
  p2 +
  (p3 +
     p4 +
     plot_layout(ncol = 1)
  ) +
  p5 +
  plot_layout(widths = c(2, 1))

# Complex layouts can be created with the `design` argument
design <- c(
  area(1, 1, 2),
  area(1, 2, 1, 3),
  area(2, 3, 3),
  area(3, 1, 3, 2),
  area(2, 2)
)
p1 + p2 + p3 + p4 + p5 + plot_layout(design = design)


# The same can be specified as a character string:
design <- "
  122
  153
  443
"
p1 + p2 + p3 + p4 + p5 + plot_layout(design = design)

# When using strings to define the design `#` can be used to denote empty
# areas
design <- "
  1##
  123
  ##3
"
p1 + p2 + p3 + plot_layout(design = design)

# Use guides="collect" to remove duplicate guides
p6 <- ggplot(mtcars) + geom_point(aes(mpg, disp, color=cyl))
p7 <- ggplot(mtcars) + geom_point(aes(mpg, hp, color=cyl))
p6 + p7 + plot_layout(guides='collect')

# Guide position must be applied to entire patchwork
p6 + p7 + plot_layout(guides='collect') &
  theme(legend.position='bottom')
```

plot_spacer *Add a completely blank area*

Description

This simple wrapper creates an empty transparent patch that can be added to push your other plots apart. The patch responds to adding theme() specifications, but only `plot.background` will have an effect.

Usage

```
plot_spacer()
```

Value

A `ggplot` object containing an empty plot

Examples

```
library(ggplot2)

p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))

p1 + plot_spacer() + p2


# To have more control over spacing, you can use the `plot.margin`
# parameter for `theme()` on each individual plot.

(p1 + theme(plot.margin = unit(c(0,30,0,0), "pt"))) +
(p2 + theme(plot.margin = unit(c(0,0,0,30), "pt")))
```

`wrap_elements` *Wrap arbitrary graphics in a patchwork-compliant patch*

Description

In order to add non-ggplot2 element to a patchwork they can be converted to a compliant representation using the `wrap_elements()` function. This allows you to position either grobs, ggplot objects, patchwork objects, or even base graphics (if passed as a formula) in either the full area, the full plotting area (anything between and including the axis label), or the panel area (only the actual area where data is drawn). Further you can still add title, subtitle, tag, and caption using the same approach as with normal ggplots (using ggtitle() and labs()) as well as styling using theme(). For the latter, only the theme elements targeting plot margins and background as well as title, subtitle, etc styling will have an effect. If a patchwork or ggplot object is wrapped, it will be fixated in its state and will no longer respond to addition of styling, geoms, etc.. When grobs and formulas are added directly, they will implicitly be converted to `wrap_elements(full = x)`.

Usage

```
wrap_elements(
  panel = NULL,
  plot = NULL,
  full = NULL,
  clip = TRUE,
  ignore_tag = FALSE
)
```

Arguments

`panel, plot, full`	A grob, ggplot, patchwork, formula, raster, or nativeRaster object to add to the respective area.
`clip`	Should the grobs be clipped if expanding outside its area
`ignore_tag`	Should tags be ignored for this patch. This is relevant when using automatic tagging of plots and the content of the patch does not qualify for a tag.

Value

A wrapped_patch object

Examples

```
library(ggplot2)
library(grid)

# Combine grobs with each other
wrap_elements(panel = textGrob('Here are some text')) +
  wrap_elements(
    panel = rectGrob(gp = gpar(fill = 'steelblue')),
    full = rectGrob(gp = gpar(fill = 'goldenrod'))
  )

# wrapped elements can still get titles etc like ggplots
wrap_elements(panel = textGrob('Here are some text')) +
  wrap_elements(
    panel = rectGrob(gp = gpar(fill = 'steelblue')),
    full = rectGrob(gp = gpar(fill = 'goldenrod'))
  ) +
  ggtitle('Title for the amazing rectangles')

# You can also pass in ggplots or patchworks to e.g. have it fill out the
# panel area
p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p1 + wrap_elements(panel = p1 + ggtitle('Look at me shrink'))

# You can even add base graphics if you pass it as a formula (requires gridGraphics package)
if (requireNamespace("gridGraphics", quietly = TRUE)) {
  p1 + wrap_elements(full = ~ plot(mtcars$mpg, mtcars$disp))

  # Adding a grob or formula directly is equivalent to placing it in `full`
  p1 + ~ plot(mtcars$mpg, mtcars$disp)
}
```

wrap_ggplot_grob — *Make a gtable created from a ggplot object patchwork compliant*

Description

This function converts a gtable, as produced by `ggplot2::ggplotGrob()` and makes it ready to be added to a patchwork. In contrast to passing the gtable to `wrap_elements()`, `wrap_ggplot_grob()` ensures proper alignment as expected. On the other hand major restructuring of the gtable will result in an object that doesn't work properly with `wrap_ggplot_grob()`.

Usage

```
wrap_ggplot_grob(x)
```

Arguments

x	A gtable as produced by `ggplot2::ggplotGrob()`

Value

A `table_patch` object to be added to a patchwork

Examples

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

p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp)) + ggtitle('disp and mpg seems connected')
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))

# Convert p2 so we can add new stuff to it
p2_table <- ggplotGrob(p2)
stamp <- textGrob('TOP SECRET', rot = 35,
  gp = gpar(fontsize = 72, fontface = 'bold')
)
p2_table <- gtable_add_grob(p2_table, stamp,
  t = 1, l = 1, b = nrow(p2_table), r = ncol(p2_table)
)

# Adding it directly will loose alignment
p1 + p2_table

# Use wrap_ggplot_grob to keep alignment
p1 + wrap_ggplot_grob(p2_table)
```

`wrap_plots` *Wrap plots into a patchwork*

Description

While the use of + is a natural way to add plots together, it can be difficult to string together multiple plots programmatically if the number of plots is not known beforehand. `wrap_plots` makes it easy to take a list of plots and add them into one composition, along with layout specifications.

Usage

```
wrap_plots(
  ...,
  ncol = NULL,
  nrow = NULL,
  byrow = NULL,
  widths = NULL,
  heights = NULL,
  guides = NULL,
  tag_level = NULL,
  design = NULL
)
```

Arguments

`...`	multiple `ggplots` or a list containing `ggplot` objects
`ncol, nrow`	The dimensions of the grid to create - if both are `NULL` it will use the same logic as facet_wrap() to set the dimensions
`byrow`	Analogous to `byrow` in matrix(). If `FALSE` the plots will be filled in in column-major order
`widths, heights`	The relative widths and heights of each column and row in the grid. Will get repeated to match the dimensions of the grid. The special value of `NA`/`-1null` will behave as `1null` unless a fixed aspect plot is inserted in which case it will allow the dimension to expand or contract to match the aspect ratio of the content
`guides`	A string specifying how guides should be treated in the layout. `'collect'` will collect guides below to the given nesting level, removing duplicates. `'keep'` will stop collection at this level and let guides be placed alongside their plot. `auto` will allow guides to be collected if a upper level tries, but place them alongside the plot if not. If you modify default guide "position" with theme(legend.position=...) while also collecting guides you must apply that change to the overall patchwork (see example).
`tag_level`	A string (`'keep'` or `'new'`) to indicate how auto-tagging should behave. See `plot_annotation()`.
`design`	Specification of the location of areas in the layout. Can either be specified as a text string or by concatenating calls to `area()` together. See the examples for further information on use.

Details

If `design` is specified as a text string *and* the plots are named (e.g. `wrap_plots(A = p1, ...)`) *and* all plot names are single characters represented in the design layout string, the plots will be matched to their respective area by name. Otherwise the areas will be filled out sequentially in the same manner as using the + operator. See the examples for more.

Value

A `patchwork` object

Examples

```
library(ggplot2)

p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) + geom_bar(aes(gear)) + facet_wrap(~cyl)
p4 <- ggplot(mtcars) + geom_bar(aes(carb))
p5 <- ggplot(mtcars) + geom_violin(aes(cyl, mpg, group = cyl))

# Either add the plots as single arguments
wrap_plots(p1, p2, p3, p4, p5)
```

```
# Or add them as a list...
plots <- list(p1, p2, p3, p4, p5)
wrap_plots(plots)

# Match plots to areas by name
design <- "#BB
           AA#"
wrap_plots(B = p1, A = p2, design = design)

# Compare to not using named plot arguments
wrap_plots(p1, p2, design = design)
```

Index