Package 'ggforce'

July 7, 2018

Type Package

Title Accelerating 'ggplot2'

Version 0.1.3

Date 2018-07-07

Author Thomas Lin Pedersen

Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>

Description The aim of 'ggplot2' is to aid in visual data investigations. This focus has led to a lack of facilities for composing specialised plots. 'ggforce' aims to be a collection of mainly new stats and geoms that fills this gap. All additional functionality is aimed to come through the official extension system so using 'ggforce' should be a stable experience.

License GPL (>= 2)

Encoding UTF-8

LazyData TRUE

Depends ggplot2 (>= 2.2.0), R (>= 3.3.0)

Imports Rcpp (>= 0.12.2), grid, dplyr, scales, MASS, tweenr (>= 0.1.5), units (>= 0.4), gtable, lazyeval

LinkingTo Rcpp

RoxygenNote 6.0.1.9000

Suggests knitr, rmarkdown, devtools

VignetteBuilder knitr

Collate 'RcppExports.R' 'aaa.R' 'arc_bar.R' 'arc.R' 'bezier.R' 'bspline.R' 'circle.R' 'facet_grid_paginate.R' 'facet_wrap_paginate.R' 'facet_zoom.R' 'ggforce_package.R' 'ggproto-classes.R' 'interpolate.R' 'link.R' 'scale-unit.R' 'sina.R' 'themes.R' 'trans.R'

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-07-07 17:00:08 UTC

R topics documented:

`facet_grid_paginate` *Split facet_grid over multiple plots*

Description

This extension to `facet_grid` will allow you to split a facetted plot over multiple pages. You define a number of rows and columns per page as well as the page number to plot, and the function will automatically only plot the correct panels. Usually this will be put in a loop to render all pages one by one.

Usage

```
facet_grid_paginate(facets, margins = FALSE, scales = "fixed",
  space = "fixed", shrink = TRUE, labeller = "label_value",
  as.table = TRUE, switch = NULL, drop = TRUE, ncol = NULL,
  nrow = NULL, page = 1, byrow = TRUE)
```

Arguments

`facets`	This argument is soft-deprecated, please us `rows` and `cols` instead.
`margins`	Either a logical value or a character vector. Margins are additional facets which contain all the data for each of the possible values of the faceting variables. If `FALSE`, no additional facets are included (the default). If `TRUE`, margins are included for all faceting variables. If specified as a character vector, it is the names of variables for which margins are to be created.

scales	Are scales shared across all facets (the default, "fixed"), or do they vary across rows ("free_x"), columns ("free_y"), or both rows and columns ("free")?
space	If "fixed", the default, all panels have the same size. If "free_y" their height will be proportional to the length of the y scale; if "free_x" their width will be proportional to the length of the x scale; or if "free" both height and width will vary. This setting has no effect unless the appropriate scales also vary.
shrink	If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
labeller	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with formulae of the type ~cyl + am. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). See label_value() for more details and pointers to other options.
as.table	If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.
switch	By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".
drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
ncol	Number of columns per page
nrow	Number of rows per page
page	The page to draw
byrow	Should the pages be created row-wise or column wise

Note

If either ncol or nrow is NULL this function will fall back to the standard facet_grid functionality.

See Also

Other ggforce facets: facet_wrap_paginate, facet_zoom

Examples

```
# Draw a small section of the grid
ggplot(diamonds) +
  geom_point(aes(carat, price), alpha = 0.1) +
  facet_grid_paginate(color~cut:clarity, ncol = 3, nrow = 3, page = 4)
```

facet_wrap_paginate *Split facet_wrap over multiple plots*

Description

This extension to facet_wrap will allow you to split a facetted plot over multiple pages. You define a number of rows and columns per page as well as the page number to plot, and the function will automatically only plot the correct panels. Usually this will be put in a loop to render all pages one by one.

Usage

```
facet_wrap_paginate(facets, nrow = NULL, ncol = NULL, scales = "fixed",
  shrink = TRUE, labeller = "label_value", as.table = TRUE,
  switch = NULL, drop = TRUE, dir = "h", strip.position = "top",
  page = 1)
```

Arguments

facets	A set of variables or expressions quoted by vars() and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to labeller). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, '~a • b, or a character vector,c("a", "b")'.
nrow	Number of rows and columns.
ncol	Number of rows and columns
scales	Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y")?
shrink	If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
labeller	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with formulae of the type ~cyl + am. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). See label_value() for more details and pointers to other options.
as.table	If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.
switch	By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".

drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
dir	Direction: either "h" for horizontal, the default, or "v", for vertical.
strip.position	By default, the labels are displayed on the top of the plot. Using strip.position it is possible to place the labels on either of the four sides by setting strip.position = c("top", "bott
page	The page to draw

Note

If either ncol or nrow is NULL this function will fall back to the standard facet_wrap functionality.

See Also

Other ggforce facets: facet_grid_paginate, facet_zoom

Examples

```
# Calculate the number of pages with 9 panels per page
n_pages <- ceiling(
  length(levels(diamonds$cut)) * length(levels(diamonds$clarity)) / 9
)

# Draw each page
for (i in seq_len(n_pages)) {
  ggplot(diamonds) +
    geom_point(aes(carat, price), alpha = 0.1) +
    facet_wrap_paginate(~cut:clarity, ncol = 3, nrow = 3, page = i)
}
```

---

facet_zoom *Facet data for zoom with context*

Description

This facetting provides the means to zoom in on a subset of the data, while keeping the view of the full dataset as a separate panel. The zoomed-in area will be indicated on the full dataset panel for reference. It is possible to zoom in on both the x and y axis at the same time. If this is done it is possible to both get each zoom separately and combined or just combined.

Usage

```
facet_zoom(x, y, xy, split = FALSE, horizontal = TRUE, zoom.size = 2,
  show.area = TRUE, shrink = TRUE)
```

Arguments

x, y, xy	An expression evaluating to a logical vector that determines the subset of data to zoom in on
split	If both x and y is given, should each axis zoom be shown separately as well? Defaults to FALSE
horizontal	If both x and y is given and split = FALSE How should the zoom panel be positioned relative to the full data panel? Defaults to TRUE
zoom.size	Sets the relative size of the zoom panel to the full data panel. The default (2) makes the zoom panel twice the size of the full data panel.
show.area	Should the zoom area be drawn below the data points on the full data panel? Defaults to TRUE.
shrink	If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.

See Also

Other ggforce facets: facet_grid_paginate, facet_wrap_paginate

Examples

```
# Zoom in on the versicolor species on the x-axis
ggplot(iris, aes(Petal.Length, Petal.Width, colour = Species)) +
    geom_point() +
    facet_zoom(x = Species == "versicolor")

# Zoom in on versicolor on both axes
ggplot(iris, aes(Petal.Length, Petal.Width, colour = Species)) +
    geom_point() +
    facet_zoom(xy = Species == "versicolor")

# Use different zoom criteria on each axis
ggplot(iris, aes(Petal.Length, Petal.Width, colour = Species)) +
    geom_point() +
    facet_zoom(x = Species != 'setosa', y = Species == 'versicolor')

# Get each axis zoom separately as well
ggplot(iris, aes(Petal.Length, Petal.Width, colour = Species)) +
    geom_point() +
    facet_zoom(xy = Species == "versicolor", split = TRUE)
```

---

geom_arc *Arcs based on radius and radians*

Description

This set of stats and geoms makes it possible to draw circle segments based on a centre point, a radius and a start and end angle (in radians). These functions are intended for cartesian coordinate systems and makes it possible to create circular plot types without using the coord_polar coordinate system.

Usage

```
stat_arc(mapping = NULL, data = NULL, geom = "arc",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 360,
  inherit.aes = TRUE, ...)

geom_arc(mapping = NULL, data = NULL, stat = "arc",
  position = "identity", n = 360, arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

stat_arc2(mapping = NULL, data = NULL, geom = "path_interpolate",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 360,
  inherit.aes = TRUE, ...)

geom_arc2(mapping = NULL, data = NULL, stat = "arc2",
  position = "identity", n = 360, arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

stat_arc0(mapping = NULL, data = NULL, geom = "arc0",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, ...)

geom_arc0(mapping = NULL, data = NULL, stat = "arc0",
  position = "identity", ncp = 5, arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), is combined with the default mapping at the top level of the plot. You only need to supply mapping if there isn't a mapping defined for the plot.
data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
geom,	stat Override the default connection between geom_arc and stat_arc.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.

n	the smoothness of the arc. Sets the number of points to use if the arc would cover a full circle
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
...	other arguments passed on to layer. There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like color = "red" or size = 3. • Other arguments to the layer, for example you override the default stat associated with the layer. • Other arguments passed on to the stat.
stat	The statistical transformation to use on the data for this layer, as a string.
arrow	specification for arrow heads, as created by arrow()
lineend	Line end style (round, butt, square)
ncp	the number of control points used to draw the arc with curveGrob. Determines how well the arc approximates a circle section

Details

An arc is a segment of a line describing a circle. It is the fundamental visual element in donut charts where the length of the segment (and conversely the angular span of the segment) describes the proportion of an entety.

Aesthetics

geom_arc understand the following aesthetics (required aesthetics are in bold):

- **x0**
- **y0**
- **r**
- **start**
- **end**
- color
- size
- linetype
- alpha
- lineend

Computed variables

x, y The start coordinates for the segment

xend, yend The end coordinates for the segment

curvature The curvature of the curveGrob to match a circle

Author(s)

Thomas Lin Pedersen

See Also

geom_arc_bar for drawing arcs with fill

Examples

```
# Lets make some data
arcs <- data.frame(
  start = seq(0, 2*pi, length.out=11)[-11],
  end = seq(0, 2*pi, length.out=11)[-1],
  r = rep(1:2, 5)
)

# Behold the arcs
ggplot() + geom_arc(aes(x0=0, y0=0, r=r, start=start, end=end,
                        linetype=factor(r)),
                    data=arcs)
```

---

geom_arc_bar *Arcs and wedges as polygons*

Description

This set of stats and geoms makes it possible to draw arcs and wedges as known from pie and donut charts as well as more specialized plottypes such as sunburst plots.

Usage

```
stat_arc_bar(mapping = NULL, data = NULL, geom = "arc_bar",
  position = "identity", n = 360, na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, ...)

stat_pie(mapping = NULL, data = NULL, geom = "arc_bar",
  position = "identity", n = 360, sep = 0, na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, ...)

geom_arc_bar(mapping = NULL, data = NULL, stat = "arc_bar",
  position = "identity", n = 360, na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), is combined with the default mapping at the top level of the plot. You only need to supply mapping if there isn't a mapping defined for the plot.
data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
geom,	stat Override the default connection between geom_arc_bar and stat_arc_bar.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
n	The number of points used to draw a full circle. The number of points on each arc will then be calculated as n / span-of-arc
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
...	other arguments passed on to layer. There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like color = "red" or size = 3. • Other arguments to the layer, for example you override the default stat associated with the layer. • Other arguments passed on to the stat.
sep	The separation between arcs in pie/donut charts
stat	The statistical transformation to use on the data for this layer, as a string.

Details

An arc bar is the thick version of an arc; that is, a circle segment drawn as a polygon in the same way as a rectangle is a thick version of a line. A wedge is a special case of an arc where the inner radius is 0. As opposed to applying coord_polar to a stacked bar chart, these layers are drawn in cartesian space, which allows for transformations not possible with the native ggplot2 approach. Most notable of these are the option to explode arcs and wedgets away from their center point, thus detaching it from the main pie/donut.

Aesthetics

geom_arc_bar understand the following aesthetics (required aesthetics are in bold):

- **x0**
- **y0**
- **r0**

- **r**
- **start** - when using stat_arc_bar
- **end** - when using stat_arc_bar
- **amount** - when using stat_pie
- explode
- color
- fill
- size
- linetype
- alpha

Computed variables

x, y x and y coordinates for the polygon

x, y The start coordinates for the segment

Author(s)

Thomas Lin Pedersen

See Also

geom_arc for drawing arcs as lines

Examples

```
# If you know the angle spans to plot it is easy
arcs <- data.frame(
  start = seq(0, 2*pi, length.out=11)[-11],
  end = seq(0, 2*pi, length.out=11)[-1],
  r = rep(1:2, 5)
)

# Behold the arcs
ggplot() + geom_arc_bar(aes(x0=0, y0=0, r0=r-1, r=r, start=start, end=end,
                        fill = r),
                    data=arcs)

# If you got values for a pie chart, use stat_pie
states <- c('eaten', "eaten but said you didn't", 'cat took it', 'for tonight',
            'will decompose slowly')
pie <- data.frame(
  state = factor(rep(states, 2), levels = states),
  type = rep(c('Pie', 'Donut'), each = 5),
  r0 = rep(c(0, 0.8), each = 5),
  focus=rep(c(0.2, 0, 0, 0, 0), 2),
  amount = c(4,3, 1, 1.5, 6, 6, 1, 2, 3, 2),
  stringsAsFactors = FALSE
```

```
)

# Look at the cakes
ggplot() + geom_arc_bar(aes(x0=0, y0=0, r0=r0, r=1, amount=amount,
                            fill=state, explode=focus),
                        data=pie, stat='pie') +
  facet_wrap(~type, ncol=1) +
  coord_fixed() +
  theme_no_axes() +
  scale_fill_brewer('', type='qual')
```

---

geom_bezier *Create quadratic or cubic bezier curves*

Description

This set of geoms makes it possible to connect points creating either quadratic or cubic beziers. bezier and bezier2 both work by calculating points along the bezier and connecting these to draw the curve. bezier0 directly draws the bezier using bezierGrob and is thus probably more performant. In line with the geom_link and geom_link2 differences geom_bezier creates the points, assign an index to each interpolated point and repeat the aesthetics for the start point, while geom_bezier2 interpolates the aesthetics between the start and end points.

Usage

```
stat_bezier(mapping = NULL, data = NULL, geom = "path",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 100,
  inherit.aes = TRUE, ...)

geom_bezier(mapping = NULL, data = NULL, stat = "bezier",
  position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, n = 100, ...)

stat_bezier2(mapping = NULL, data = NULL, geom = "path_interpolate",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 100,
  inherit.aes = TRUE, ...)

geom_bezier2(mapping = NULL, data = NULL, stat = "bezier2",
  position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, n = 100, ...)

stat_bezier0(mapping = NULL, data = NULL, geom = "bezier0",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, ...)

geom_bezier0(mapping = NULL, data = NULL, stat = "bezier0",
  position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), is combined with the default mapping at the top level of the plot. You only need to supply mapping if there isn't a mapping defined for the plot.
data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
geom,	stat Override the default connection between geom_arc and stat_arc.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
n	The number of points to create for each segment
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
...	other arguments passed on to layer. There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like color = "red" or size = 3. • Other arguments to the layer, for example you override the default stat associated with the layer. • Other arguments passed on to the stat.
stat	The statistical transformation to use on the data for this layer, as a string.
arrow	specification for arrow heads, as created by arrow()
lineend	Line end style (round, butt, square)

Details

Input data is understood as a sequence of data points the first being the start point, then followed by one or two control points and then the end point. More than 4 and less than 3 points per group will throw an error. bezierGrob only takes cubic beziers so if three points are supplied the middle one as duplicated. This, along with the fact that bezierGrob estimates the curve using an x-spline means that the curves produced by geom_bezier and geom_bezier2 deviates from those produced by geom_bezier0. If you want true bezier paths use geom_bezier or geom_bezier2.

Aesthetics

geom_link, geom_link2 and geom_lin0 understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**

- color
- size
- linetype
- alpha
- lineend

Computed variables

x, y The interpolated point coordinates

index The progression along the interpolation mapped between 0 and 1

Author(s)

Thomas Lin Pedersen

Examples

```
beziers <- data.frame(
    x = c(1, 2, 3, 4, 4, 6, 6),
    y = c(0, 2, 0, 0, 2, 2, 0),
    type = rep(c('cubic', 'quadratic'), c(3, 4)),
    point = c('end', 'control', 'end', 'end', 'control', 'control', 'end')
)
help_lines <- data.frame(
    x = c(1, 3, 4, 6),
    xend = c(2, 2, 4, 6),
    y = 0,
    yend = 2
)
ggplot() + geom_segment(aes(x = x, xend = xend, y = y, yend = yend),
                        data = help_lines,
                        arrow = arrow(length = unit(c(0, 0, 0.5, 0.5), 'cm')),
                        colour = 'grey') +
    geom_bezier(aes(x= x, y = y, group = type, linetype = type),
                data = beziers) +
    geom_point(aes(x = x, y = y, colour = point), data = beziers)
```

---

geom_bspline *B-splines based on control points*

Description

This set of stats and geoms makes it possible to draw b-splines based on a set of control points. As with geom_bezier there exists several versions each having there own strengths. The base version calculates the b-spline as a number of points along the spline and connects these with a path. The *2 version does the same but in addition interpolates aesthetics between each control point. This makes the *2 version considerably slower so it shouldn't be used unless needed. The *0 version uses xsplineGrob with shape = 1 to approximate a b-spline for a high performant version.

Usage

```
stat_bspline(mapping = NULL, data = NULL, geom = "path",
  position = "identity", na.rm = FALSE, n = 100, show.legend = NA,
  inherit.aes = TRUE, ...)

geom_bspline(mapping = NULL, data = NULL, stat = "bspline",
  position = "identity", arrow = NULL, n = 100, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

stat_bspline2(mapping = NULL, data = NULL, geom = "path_interpolate",
  position = "identity", na.rm = FALSE, n = 100, show.legend = NA,
  inherit.aes = TRUE, ...)

geom_bspline2(mapping = NULL, data = NULL, stat = "bspline2",
  position = "identity", arrow = NULL, n = 100, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

stat_bspline0(mapping = NULL, data = NULL, geom = "bspline0",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, ...)

geom_bspline0(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), is combined with the default mapping at the top level of the plot. You only need to supply mapping if there isn't a mapping defined for the plot.
data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
geom,	stat Override the default connection between geom_arc and stat_arc.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
n	The number of points generated for each spline
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
...	other arguments passed on to layer. There are three types of arguments you can use here:

- Aesthetics: to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`.
- Other arguments to the layer, for example you override the default `stat` associated with the layer.
- Other arguments passed on to the stat.

`stat` The statistical transformation to use on the data for this layer, as a string.

`arrow` specification for arrow heads, as created by arrow()

`lineend` Line end style (round, butt, square)

Aesthetics

geom_edge_bundle understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- color
- size
- linetype
- alpha
- lineend

Computed variables

x, y The coordinates for the path describing the spline

index The progression along the interpolation mapped between 0 and 1

Author(s)

Thomas Lin Pedersen. The C++ code for De Boor's algorithm has been adapted from Jason Yu-Tseh Chi implementation

References

Holten, D. (2006). *Hierarchical edge bundles: visualization of adjacency relations in hierarchical data.* IEEE Transactions on Visualization and Computer Graphics, **12**(5), 741-748. http://doi.org/10.1109/TVCG.2006.147

Examples

```
# Define some control points
cp <- data.frame(
  x = c(0, -5, -5, 5, 5, 2.5, 5, 7.5, 5, 2.5, 5, 7.5, 5, -2.5, -5, -7.5, -5,
        -2.5, -5, -7.5, -5),
  y = c(0, -5, 5, -5, 5, 5, 7.5, 5, 2.5, -5, -7.5, -5, -2.5, 5, 7.5, 5, 2.5,
        -5, -7.5, -5, -2.5),
  class = sample(letters[1:3], 21, replace = TRUE)
)
```

```
# Now create some paths between them
paths <- data.frame(
  ind = c(7,5,8,8,5,9,9,5,6,6,5,7,7,5,1,3,15,8,5,1,3,17,9,5,1,2,19,6,5,1,4,
          12,7,5,1,4,10,6,5,1,2,20),
  group = c(1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,5,5,6,6,6,6,6,7,7,7,7,7,8,8,8,8,8,
            9,9,9,9,9,10,10,10,10,10)
)
paths$x <- cp$x[paths$ind]
paths$y <- cp$y[paths$ind]
paths$class <- cp$class[paths$ind]

ggplot() +
  geom_bspline(aes(x=x, y=y, group=group, colour = ..index..), data=paths) +
  geom_point(aes(x=x, y=y), data=cp, color='steelblue')

ggplot() +
  geom_bspline2(aes(x=x, y=y, group=group, colour = class), data=paths) +
  geom_point(aes(x=x, y=y), data=cp, color='steelblue')

ggplot() +
  geom_bspline0(aes(x=x, y=y, group=group), data=paths) +
  geom_point(aes(x=x, y=y), data=cp, color='steelblue')
```

`geom_circle` *Circles based on center and radius*

Description

This set of stats and geoms makes it possible to draw circles based on a center point and a radius. In contrast to using `geom_point`, the size of the circles are related to the coordinate system and not to a separate scale. These functions are intended for cartesian coordinate systems and will only produce a true circle if `coord_fixed` is used.

Usage

```
stat_circle(mapping = NULL, data = NULL, geom = "circle",
  position = "identity", n = 360, na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, ...)

geom_circle(mapping = NULL, data = NULL, stat = "circle",
  position = "identity", n = 360, na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, ...)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes` or `aes_`. If specified and `inherit.aes = TRUE` (the default), is combined with the default mapping at the top level of the plot. You only need to supply mapping if there isn't a mapping defined for the plot.

data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
geom,	stat Override the default connection between geom_arc and stat_arc.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
n	The number of points on the generated path per circle.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
...	other arguments passed on to layer. There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like color = "red" or size = 3. • Other arguments to the layer, for example you override the default stat associated with the layer. • Other arguments passed on to the stat.
stat	The statistical transformation to use on the data for this layer, as a string.

Aesthetics

geom_arc understand the following aesthetics (required aesthetics are in bold):

- **x0**
- **y0**
- **r**
- color
- fill
- size
- linetype
- alpha
- lineend

Computed variables

x, y The start coordinates for the segment

Note

If the intend is to draw a bubble chart then use geom_point and map a variable to the size scale

Author(s)

Thomas Lin Pedersen

See Also

geom_arc_bar for drawing arcs with fill

Examples

```
# Lets make some data
circles <- data.frame(
  x0 = rep(1:3, 3),
  y0 =  rep(1:3, each=3),
  r = seq(0.1, 1, length.out = 9)
)

# Behold the some circles
ggplot() + geom_circle(aes(x0=x0, y0=y0, r=r, fill=r), data=circles)

# Use coord_fixed to ensure true circularity
ggplot() + geom_circle(aes(x0=x0, y0=y0, r=r, fill=r), data=circles) +
  coord_fixed()
```

---

geom_link *Link points with paths*

Description

This set of geoms makes it possible to connect points using straight lines. Before you think geom_segment and geom_path, these functions have some additional tricks up their sleeves. geom_link connects two points in the same way as geom_segment but does so by interpolating multiple points between the two. An additional column called index is added to the data with a sequential progression of the interpolated points. This can be used to map color or size to the direction of the link. geom_link2 uses the same syntax as geom_path but interpolates between the aesthetics given by each row in the data.

Usage

```
stat_link(mapping = NULL, data = NULL, geom = "path",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 100,
  inherit.aes = TRUE, ...)

stat_link2(mapping = NULL, data = NULL, geom = "path_interpolate",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 100,
  inherit.aes = TRUE, ...)
```

```
geom_link(mapping = NULL, data = NULL, stat = "link",
  position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, n = 100, ...)

geom_link2(mapping = NULL, data = NULL, stat = "link2",
  position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, n = 100, ...)

geom_link0(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ..., arrow = NULL, arrow.fill = NULL,
  lineend = "butt", linejoin = "round", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), is combined with the default mapping at the top level of the plot. You only need to supply mapping if there isn't a mapping defined for the plot.
data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
geom,	stat Override the default connection between geom_arc and stat_arc.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
n	The number of points to create for each segment
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
...	other arguments passed on to layer. There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like color = "red" or size = 3. • Other arguments to the layer, for example you override the default stat associated with the layer. • Other arguments passed on to the stat.
stat	The statistical transformation to use on the data for this layer, as a string.
arrow	specification for arrow heads, as created by arrow()
lineend	Line end style (round, butt, square)
arrow.fill	fill color to use for the arrow head (if closed). 'NULL' means use colour aesthetic.
linejoin	Line join style (round, mitre, bevel).

Aesthetics

geom_link understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **xend**
- **yend**
- color
- size
- linetype
- alpha
- lineend

geom_link2 understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- color
- size
- linetype
- alpha
- lineend

Computed variables

x, y The interpolated point coordinates

index The progression along the interpolation mapped between 0 and 1

Author(s)

Thomas Lin Pedersen

Examples

```
# Lets make some data
lines <- data.frame(
  x = c(5, 12, 15, 9, 6),
  y = c(17, 20, 4, 15, 5),
  xend = c(19, 17, 2, 9, 5),
  yend = c(10, 18, 7, 12, 1),
  width = c(1, 10, 6, 2, 3),
  colour = letters[1:5]
)

ggplot() + geom_link(aes(x = x, y = y, xend = xend, yend = yend,
                         colour = colour, alpha = ..index..,
```

```
                        size = ..index..),
                    data = lines)

ggplot() + geom_link2(aes(x = x, y = y, colour = colour, size = width,
                          group = 1),
                      data = lines, lineend = 'round', n = 500)
```

---

geom_sina *Sina plot*

Description

The sina plot is a data visualization chart suitable for plotting any single variable in a multiclass dataset. It is an enhanced jitter strip chart, where the width of the jitter is controlled by the density distribution of the data within each class.

Usage

```
stat_sina(mapping = NULL, data = NULL, geom = "sina",
  position = "identity", ..., binwidth = NULL, bins = NULL,
  scale = TRUE, method = "density", maxwidth = NULL, adjust = 1,
  bin_limit = 1, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

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

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), is combined with the default mapping at the top level of the plot. You only need to supply mapping if there isn't a mapping defined for the plot.
data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
geom,	stat Override the default connection between geom_sina and stat_sina.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	other arguments passed on to layer. There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like color = "red" or size = 3. • Other arguments to the layer, for example you override the default stat associated with the layer. • Other arguments passed on to the stat.

binwidth	The width of the bins. The default is to use bins bins that cover the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data.
bins	Number of bins. Overridden by binwidth. Defaults to 50.
scale	Logical. When set to TRUE x-coordinate widths across all groups are scaled based on the densiest area in the plot. Default: TRUE
method	Choose the method to spread the samples within the same bin along the x-axis. Available methods: "density", "counts" (can be abbreviated, e.g. "d"). See Details.
maxwidth	Control the maximum width the points can spread into. Values between 0 and 1.
adjust	Adjusts the bandwidth of the density kernel when method == "density" (see density).
bin_limit	If the samples within the same y-axis bin are more than bin_limit, the samples's X coordinates will be adjusted.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
stat	The statistical transformation to use on the data for this layer, as a string.

Details

There are two available ways to define the x-axis borders for the samples to spread within:

- method == "density"
  A density kernel is estimated along the y-axis for every sample group. The borders are then defined by the density curve. Tuning parameter adjust can be used to control the density bandwidth in the same way it is used in density.
- method == "counts":
  The borders are defined by the number of samples that occupy the same bin.

Aesthetics

@section Aesthetics: geom_point understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- colour
- fill

- group
- shape
- size
- stroke

Learn more about setting these aesthetics in vignette("ggplot2-specs")

Computed variables

bin_counts sample counts per bin per group

scaled adjusted x-coordinates

Author(s)

Nikos Sidiropoulos

Examples

```
ggplot(midwest, aes(state, area)) + geom_point()

# Boxplot and Violin plots convey information on the distribution but not the
# number of samples, while Jitter does the opposite.
ggplot(midwest, aes(state, area)) + geom_violin()
ggplot(midwest, aes(state, area)) + geom_jitter()

# Sina does both!
ggplot(midwest, aes(state, area)) + geom_violin() + geom_sina()

p <- ggplot(midwest, aes(state, popdensity)) + scale_y_log10()
p + geom_sina()

# Colour the points based on the data set's columns
p + geom_sina(aes(colour = inmetro))

# Or any other way
cols <- midwest$popdensity > 10000
p + geom_sina(colour = cols + 1L)

# Sina plots with continuous x:
p <- ggplot(midwest, aes(cut_width(area, 0.02), popdensity)) + scale_y_log10()
p + geom_sina()


###Sample gaussian distributions
# Unimodal
a <- rnorm(500, 6, 1)
b <- rnorm(400, 5, 1.5)

# Bimodal
c <- c(rnorm(200, 3, .7), rnorm(50, 7, 0.4))
```

```
# Trimodal
d <- c(rnorm(200, 2, 0.7), rnorm(300, 5.5, 0.4), rnorm(100, 8, 0.4))

df <- data.frame(
  "Distribution" = c(rep("Unimodal 1", length(a)),
                     rep("Unimodal 2", length(b)),
                     rep("Bimodal", length(c)),
                     rep("Trimodal", length(d))),
  "Value" = c(a, b, c, d))

# Reorder levels
df$Distribution <- factor(df$Distribution,
                          levels(df$Distribution)[c(3, 4, 1, 2)])

p <- ggplot(df, aes(Distribution, Value))
p + geom_boxplot()
p + geom_violin() + geom_sina()

# By default, Sina plot scales the width of the class according to the width
# of the class with the highest density. Turn group-wise scaling off with:
p + geom_violin() + geom_sina(scale = FALSE)
```

---

ggforce *ggforce: Accelerating ggplot2*

Description

This package contains a range of useful stats, geoms, trans and utilities to get more out of ggplot2. Many of the additions are mainly aimed at developers wanting to develop new visualizations using ggplot2 rather than statisticians wanting to understand their data.

Author(s)

Thomas Lin Pedersen

Examples

```
rocketData <- data.frame(
  x = c(1,1,2,2),
  y = c(1,2,2,3)
)
rocketData <- do.call(rbind, lapply(seq_len(500)-1, function(i) {
  rocketData$y <- rocketData$y - c(0,i/500);
  rocketData$group <- i+1;
  rocketData
}))
rocketData2 <- data.frame(
  x = c(2, 2.25, 2),
  y = c(2, 2.5, 3)
)
```

```
rocketData2 <- do.call(rbind, lapply(seq_len(500)-1, function(i) {
  rocketData2$x[2] <- rocketData2$x[2] - i*0.25/500;
  rocketData2$group <- i+1 + 500;
  rocketData2
}))

ggplot() + geom_link(aes(x=2, y=2, xend=3, yend=3, alpha=..index..,
                         size = ..index..), colour='goldenrod', n=500) +
           geom_bezier(aes(x=x, y=y, group=group, colour=..index..),
                       data=rocketData) +
           geom_bezier(aes(x=y, y=x, group=group, colour=..index..),
                       data=rocketData) +
           geom_bezier(aes(x=x, y=y, group=group, colour=1),
                       data=rocketData2) +
           geom_bezier(aes(x=y, y=x, group=group, colour=1),
                       data=rocketData2) +
           geom_text(aes(x=1.65, y=1.65, label='ggplot2', angle=45),
                     colour='white', size=15) +
           coord_fixed() +
           scale_x_reverse() +
           scale_y_reverse() +
           scale_alpha(range=c(1, 0), guide='none') +
           scale_size_continuous(range=c(20, 0.1), trans='exp',
                                 guide='none') +
           scale_color_continuous(guide='none') +
           xlab('') + ylab('') +
           ggtitle('ggforce: Accelerating ggplot2') +
           theme(plot.title = element_text(size = 20))
```

`n_pages` *Determine the number of pages in a paginated facet plot*

Description

This is a simple helper that returns the number of pages it takes to plot all panels when using `facet_wrap_paginate` and `facet_grid_paginate`. It partially builds the plot so depending on the complexity of your plot it might take some time to calculate...

Usage

```
n_pages(plot)
```

Arguments

`plot`	A ggplot object using either facet_wrap_paginate or facet_grid_paginate

Value

If the plot uses using either facet_wrap_paginate or facet_grid_paginate it returns the total number of pages. Otherwise it returns NULL

Examples

```
p <- ggplot(diamonds) +
    geom_point(aes(carat, price), alpha = 0.1) +
    facet_wrap_paginate(~cut:clarity, ncol = 3, nrow = 3, page = 1)
n_pages(p)
```

---

power_trans *Create a power transformation object*

Description

This function can be used to create a proper trans object that encapsulates a power transformation (x^n).

Usage

```
power_trans(n)
```

Arguments

n	The degree of the power transformation

Value

A trans object

Examples

```
# Power of 5 transformations
trans <- power_trans(2)
trans$transform(1:10)

# Cubic root transformation
trans <- power_trans(1/3)
trans$transform(1:10)

# Use it in a plot
ggplot() + geom_line(aes(x=1:10, y=1:10)) +
  scale_x_continuous(trans = power_trans(2), expand=c(0,1))
```

radial_trans *Create radial data in a cartesian coordinate system*

Description

This function creates a trans object that converts radial data to their corresponding coordinates in cartesian space. The trans object is created for a specific radius and angle range that will be mapped to the unit circle so data doesn't have to be normalized to 0-1 and 0-2*pi in advance. While there exists a clear mapping from radial to cartesian, the inverse is not true as radial representation is periodic. It is impossible to know how many revolutions around the unit circle a point has taken from reading its coordinates. The inverse function will always assume that coordinates are in their first revolution i.e. map them back within the range of a.range.

Usage

```
radial_trans(r.range, a.range, offset = pi/2, pad = 0.5, clip = FALSE)
```

Arguments

r.range	The range in radius that correspond to 0 - 1 in the unit circle.
a.range	The range in angles that correspond to 2*pi - 0. As radians are normally measured counterclockwise while radial displays are read clockwise it's an inverse mapping
offset	The offset in angles to apply. Determines that start position on the circle. pi/2 (the default) corresponds to 12 o'clock.
pad	Adds to the end points of the angle range in order to separate the start and end point. Defaults to 0.5
clip	Should input data be clipped to r.range and a.range or be allowed to extend beyond. Defaults to FALSE (no clipping)

Value

A trans object. The transform method for the object takes an r (radius) and a (angle) argument and returns a data.frame with x and y columns with rows for each element in r/a. The inverse method takes an x and y argument and returns a data.frame with r and a columns and rows for each element in x/y.

Note

While trans objects are often used to modify scales in ggplot2, radial transformation is different as it is a coordinate transformation and takes two arguments. Consider it a trans version of coord_polar and use it to transform your data prior to plotting.

Examples

```
# Some data in radial form
rad <- data.frame(r = seq(1, 10, by = 0.1), a = seq(1, 10, by = 0.1))

# Create a transformation
radial <- radial_trans(c(0,1), c(0,5))

# Get data in x, y
cart <- radial$transform(rad$r, rad$a)

# Have a look
ggplot() + geom_path(aes(x=x, y=y), data = cart, color='forestgreen') +
  geom_path(aes(x=r, y=a), data = rad, color='firebrick')
```

---

scale_unit *Position scales for units data*

Description

These are the default scales for the units class. These will usually be added automatically. To override manually, use `scale_*_unit`.

Usage

```
scale_x_unit(name = waiver(), breaks = waiver(), unit = NULL,
  minor_breaks = waiver(), labels = waiver(), limits = NULL,
  expand = waiver(), oob = censor, na.value = NA_real_,
  trans = "identity", position = "bottom", sec.axis = waiver())

scale_y_unit(name = waiver(), breaks = waiver(), unit = NULL,
  minor_breaks = waiver(), labels = waiver(), limits = NULL,
  expand = waiver(), oob = censor, na.value = NA_real_,
  trans = "identity", position = "left", sec.axis = waiver())
```

Arguments

`name`	The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.
`breaks`	One of: • `NULL` for no breaks • `waiver()` for the default breaks computed by the transformation object • A numeric vector of positions • A function that takes the limits as input and returns breaks as output

`unit` A unit specification to use for the axis. If given, the values will be converted to this unit before plotting. An error will be thrown if the specified unit is incompatible with the unit of the data.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks.

`labels` One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

`limits` A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum.

`expand` Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`oob` Function that handles limits outside of the scale limits (out of bounds). The default replaces out of bounds values with NA.

`na.value` Missing values will be replaced with this value.

`trans` Either the name of a transformation object, or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "exp", "identity", "log", "log10", "log1p", "log2", "logit", "probability", "probit", "reciprocal", "reverse" and "sqrt".

A transformation object bundles together a transform, it's inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called `name_trans`, e.g. `scales::boxcox_trans()`. You can create your own transformation with `scales::trans_new()`.

`position` The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales

`sec.axis` specify a secondary axis

Examples

```
library(units)
gallon <- make_unit('gallon')
mtcars$consumption <- mtcars$mpg * with(ud_units, mi/gallon)
mtcars$power <- mtcars$hp * with(ud_units, hp)

# Use units encoded into the data
ggplot(mtcars) +
    geom_point(aes(power, consumption))
```

```
# Convert units on the fly during plotting
ggplot(mtcars) +
    geom_point(aes(power, consumption)) +
    scale_x_unit(unit = 'W') +
    scale_y_unit(unit = 'km/l')

# Resolve units when transforming data
ggplot(mtcars) +
    geom_point(aes(power, 1/consumption))
```

`StatArcBar` *ggforce extensions to ggplot2*

Description

ggforce makes heavy use of the ggproto class system to extend the functionality of ggplot2. In general the actual classes should be of little interest to users as the standard ggplot2 api of using geom_* and stat_* functions for building up the plot is encouraged.

`theme_no_axes` *Theme without axes and gridlines*

Description

This theme is a simple wrapper around any complete theme that removes the axis text, title and ticks as well as the grid lines for plots where these have little meaning.

Usage

```
theme_no_axes(base.theme = theme_bw())
```

Arguments

`base.theme`	The theme to use as a base for the new theme. Defaults to `theme_bw()`.

Value

A modified version of base.theme

Examples

```
p <- ggplot() + geom_point(aes(x = wt, y = qsec), data = mtcars)

p + theme_no_axes()
p + theme_no_axes(theme_grey())
```

`trans_reverser` *Reverse a transformation*

Description

While the scales package export a reverse_trans object it does not allow for reversing of already transformed ranged - e.g. a reverse exp transformation is not possible. trans_reverser takes a trans object or something coercible to one and creates a reverse version of it.

Usage

```
trans_reverser(trans)
```

Arguments

`trans`	A trans object or an object that can be converted to one using `as.trans`

Value

A trans object

Examples

```
# Lets make a plot
p <- ggplot() + geom_line(aes(x=1:10, y=1:10))

# scales already have a reverse trans
p + scale_x_continuous(trans='reverse')

# But what if you wanted to reverse an already log transformed scale?
p + scale_x_continuous(trans=trans_reverser('log'))
```

Index