

# Package ‘ggplot2’

February 19, 2015

**Type** Package

**Title** An implementation of the Grammar of Graphics

**Version** 1.0.0

**Author** Hadley Wickham <h.wickham@gmail.com>, Winston Chang <winston@stdout.org>

**Maintainer** Hadley Wickham <h.wickham@gmail.com>

**Description** An implementation of the grammar of graphics in R. It combines the advantages of both base and lattice graphics: conditioning and shared axes are handled automatically, and you can still build up a plot step by step from multiple data sources. It also implements a sophisticated multidimensional conditioning system and a consistent interface to map data to aesthetic attributes. See the ggplot2 website for more information, documentation and examples.

**Depends** R (>= 2.14), stats, methods

**Imports** plyr (>= 1.7.1), digest, grid, gtable (>= 0.1.1), reshape2, scales (>= 0.2.3), proto, MASS

**Suggests** quantreg, Hmisc, mapproj, maps, hexbin, maptools, multcomp, nlme, testthat, knitr, mgcv

**VignetteBuilder** knitr

**Enhances** sp

**License** GPL-2

**URL** <http://ggplot2.org>, <https://github.com/hadley/ggplot2>

**BugReports** <https://github.com/hadley/ggplot2/issues>

**LazyData** true

**Collate** 'aaa-.r' 'aaa-constants.r' 'aes-calculated.r'  
'aes-colour-fill-alpha.r' 'aes-group-order.r'  
'aes-linetype-size-shape.r' 'aes-position.r' 'aes.r' 'geom-.r'  
'annotation-custom.r' 'annotation-logticks.r' 'geom-polygon.r'  
'geom-map.r' 'annotation-map.r' 'geom-raster.r'  
'annotation-raster.r' 'annotation.r' 'autoplot.r' 'bench.r'

```

'coord-.r' 'coord-cartesian-.r' 'coord-fixed.r' 'coord-flip.r'
'coord-map.r' 'coord-munch.r' 'coord-polar.r'
'coord-quickmap.R' 'coord-transform.r' 'facet-.r'
'facet-grid-.r' 'facet-labels.r' 'facet-layout.r'
'facet-locate.r' 'facet-null.r' 'facet-viewports.r'
'facet-wrap.r' 'fortify-lm.r' 'fortify-map.r'
'fortify-multcomp.r' 'fortify-spatial.r' 'fortify.r'
'geom-abline.r' 'geom-bar-.r' 'geom-bar-histogram.r'
'geom-bin2d.r' 'geom-blank.r' 'geom-boxplot.r'
'geom-crossbar.r' 'geom-defaults.r' 'geom-dotplot.r'
'geom-error.r' 'geom-errorh.r' 'geom-freqpoly.r' 'geom-hex.r'
'geom-hline.r' 'geom-linrange.r' 'geom-path-.r'
'geom-path-contour.r' 'geom-path-density2d.r'
'geom-path-line.r' 'geom-path-step.r' 'geom-point-.r'
'geom-point-jitter.r' 'geom-pointrange.r' 'geom-quantile.r'
'geom-rect.r' 'geom-ribbon-.r' 'geom-ribbon-density.r'
'geom-rug.r' 'geom-segment.r' 'geom-smooth.r' 'geom-text.r'
'geom-tile.r' 'geom-violin.r' 'geom-vline.r' 'ggplot2.r'
'grob-absolute.r' 'grob-dotstack.r' 'grob-null.r'
'guide-colorbar.r' 'guide-legend.r' 'guides-.r' 'guides-axis.r'
'guides-grid.r' 'labels.r' 'layer.r' 'limits.r' 'panel.r'
'plot-build.r' 'plot-construction.r' 'plot-last.r'
'plot-render.r' 'plot.r' 'position-.r' 'position-collide.r'
'position-dodge.r' 'position-fill.r' 'position-identity.r'
'position-jitter.r' 'position-jitterdodge.R' 'position-stack.r'
'quick-plot.r' 'save.r' 'scale-.r' 'scale-alpha.r'
'scale-area.r' 'scale-brewer.r' 'scale-continuous.r'
'scale-date.r' 'scale-datetime.r' 'scale-discrete-.r'
'scale-gradient.r' 'scale-gradient2.r' 'scale-gradientn.r'
'scale-grey.r' 'scale-hue.r' 'scale-identity.r'
'scale-linetype.r' 'scale-manual.r' 'scale-shape.r'
'scale-size.r' 'scales-.r' 'stat-.r' 'stat-bin.r'
'stat-bin2d.r' 'stat-bindot.r' 'stat-binhex.r' 'stat-boxplot.r'
'stat-contour.r' 'stat-density-2d.r' 'stat-density.r'
'stat-ecdf.r' 'stat-ellipse.R' 'stat-function.r'
'stat-identity.r' 'stat-qq.r' 'stat-quantile.r'
'stat-smooth-methods.r' 'stat-smooth.r' 'stat-spoke.r'
'stat-sum.r' 'stat-summary-2d.r' 'stat-summary-hex.r'
'stat-summary.r' 'stat-unique.r' 'stat-vline.r'
'stat-ydensity.r' 'summary.r' 'templates.r' 'theme-defaults.r'
'theme-elements.r' 'theme.r' 'translate-qplot-base.r'
'translate-qplot-ggplot.r' 'translate-qplot-gpl.r'
'translate-qplot-lattice.r' 'utilities-break.r'
'utilities-grid.r' 'utilities-help.r' 'utilities-layer.r'
'utilities-matrix.r' 'utilities-resolution.r'
'utilities-table.r' 'utilities.r' 'xxx-digest.r' 'zxx.r'
'zzz.r'

```

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2014-05-21 15:36:28

## R topics documented:

<a href="#">+.gg</a>	<a href="#">6</a>
<a href="#">add_theme</a>	<a href="#">8</a>
<a href="#">aes</a>	<a href="#">8</a>
<a href="#">aes_all</a>	<a href="#">9</a>
<a href="#">aes_auto</a>	<a href="#">9</a>
<a href="#">aes_colour_fill_alpha</a>	<a href="#">10</a>
<a href="#">aes_group_order</a>	<a href="#">11</a>
<a href="#">aes_linetype_size_shape</a>	<a href="#">13</a>
<a href="#">aes_position</a>	<a href="#">14</a>
<a href="#">aes_string</a>	<a href="#">15</a>
<a href="#">annotate</a>	<a href="#">16</a>
<a href="#">annotation_custom</a>	<a href="#">17</a>
<a href="#">annotation_logticks</a>	<a href="#">18</a>
<a href="#">annotation_map</a>	<a href="#">20</a>
<a href="#">annotation_raster</a>	<a href="#">21</a>
<a href="#">autoplot</a>	<a href="#">22</a>
<a href="#">borders</a>	<a href="#">22</a>
<a href="#">calc_element</a>	<a href="#">23</a>
<a href="#">coord_cartesian</a>	<a href="#">24</a>
<a href="#">coord_fixed</a>	<a href="#">25</a>
<a href="#">coord_flip</a>	<a href="#">26</a>
<a href="#">coord_map</a>	<a href="#">26</a>
<a href="#">coord_polar</a>	<a href="#">28</a>
<a href="#">coord_quickmap</a>	<a href="#">29</a>
<a href="#">coord_trans</a>	<a href="#">30</a>
<a href="#">cut_interval</a>	<a href="#">32</a>
<a href="#">cut_number</a>	<a href="#">32</a>
<a href="#">diamonds</a>	<a href="#">33</a>
<a href="#">economics</a>	<a href="#">34</a>
<a href="#">element_blank</a>	<a href="#">34</a>
<a href="#">element_line</a>	<a href="#">35</a>
<a href="#">element_rect</a>	<a href="#">35</a>
<a href="#">element_text</a>	<a href="#">36</a>
<a href="#">expand_limits</a>	<a href="#">36</a>
<a href="#">facet_grid</a>	<a href="#">37</a>
<a href="#">facet_null</a>	<a href="#">40</a>
<a href="#">facet_wrap</a>	<a href="#">41</a>
<a href="#">fortify</a>	<a href="#">42</a>
<a href="#">fortify-multcomp</a>	<a href="#">43</a>
<a href="#">fortify.lm</a>	<a href="#">44</a>
<a href="#">fortify.map</a>	<a href="#">45</a>
<a href="#">fortify.sp</a>	<a href="#">46</a>

geom_abline . . . . .	47
geom_area . . . . .	48
geom_bar . . . . .	49
geom_bin2d . . . . .	52
geom_blank . . . . .	53
geom_boxplot . . . . .	54
geom_contour . . . . .	57
geom_crossbar . . . . .	58
geom_density . . . . .	59
geom_density2d . . . . .	60
geom_dotplot . . . . .	61
geom_errorbar . . . . .	64
geom_errorbarh . . . . .	65
geom_freqpoly . . . . .	67
geom_hex . . . . .	68
geom_histogram . . . . .	69
geom_hline . . . . .	71
geom_jitter . . . . .	73
geom_line . . . . .	74
geom_linerange . . . . .	76
geom_map . . . . .	77
geom_path . . . . .	79
geom_point . . . . .	82
geom_pointrange . . . . .	84
geom_polygon . . . . .	85
geom_quantile . . . . .	87
geom_raster . . . . .	88
geom_rect . . . . .	89
geom_ribbon . . . . .	91
geom_rug . . . . .	92
geom_segment . . . . .	93
geom_smooth . . . . .	95
geom_step . . . . .	96
geom_text . . . . .	98
geom_tile . . . . .	99
geom_violin . . . . .	101
geom_vline . . . . .	103
ggplot.data.frame . . . . .	105
ggplot2 . . . . .	105
ggsave . . . . .	106
ggtheme . . . . .	107
guides . . . . .	108
guide_colourbar . . . . .	109
guide_legend . . . . .	112
hmisc . . . . .	115
is.ggplot . . . . .	115
is.rel . . . . .	116
is.theme . . . . .	116

labeller	117
label_both	118
label_bquote	119
label_parsed	119
label_value	120
label_wrap_gen	120
labs	121
last_plot	122
map_data	122
mean_se	123
midwest	123
movies	125
mpg	126
msleep	126
opts	127
position_dodge	128
position_fill	129
position_identity	129
position_jitter	130
position_jitterdodge	131
position_stack	132
presidential	133
print.ggplot	133
qplot	134
rel	135
resolution	136
scale_alpha	137
scale_area	137
scale_colour_brewer	138
scale_colour_gradient	140
scale_colour_gradient2	142
scale_colour_gradientn	143
scale_colour_grey	145
scale_colour_hue	146
scale_identity	148
scale_linetype	149
scale_manual	149
scale_shape	151
scale_size	152
scale_size_area	153
scale_x_continuous	153
scale_x_date	155
scale_x_datetime	157
scale_x_discrete	158
seals	160
stat_bin	160
stat_bin2d	162
stat_bindot	163

stat_binhex . . . . .	165
stat_boxplot . . . . .	166
stat_contour . . . . .	167
stat_density . . . . .	169
stat_density2d . . . . .	171
stat_ecdf . . . . .	173
stat_ellipse . . . . .	174
stat_function . . . . .	175
stat_identity . . . . .	177
stat_qq . . . . .	178
stat_quantile . . . . .	179
stat_smooth . . . . .	181
stat_spoke . . . . .	183
stat_sum . . . . .	184
stat_summary . . . . .	186
stat_summary2d . . . . .	189
stat_summary_hex . . . . .	190
stat_unique . . . . .	191
stat_ydensity . . . . .	192
theme . . . . .	193
theme_blank . . . . .	197
theme_update . . . . .	198
translate_qplot_base . . . . .	199
translate_qplot_ggplot . . . . .	202
translate_qplot_gpl . . . . .	203
translate_qplot_lattice . . . . .	204
update_element . . . . .	206
update_geom_defaults . . . . .	207
update_labels . . . . .	208
xlim . . . . .	208

**Index****210**

+.gg

*Modify a ggplot or theme object by adding on new components.***Description**

This operator allows you to add objects to a ggplot or theme object.

**Usage**

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

e1 %+% e2

e1 %+replace% e2
```

## Arguments

e1	An object of class ggplot or theme
e2	A component to add to e1

## Details

If the first object is an object of class ggplot, you can add the following types of objects, and it will return a modified ggplot object.

- `data.frame`: replace current `data.frame` (must use `%>%`)
- `uneval`: replace current aesthetics
- `layer`: add new layer
- `theme`: update plot theme
- `scale`: replace current scale
- `coord`: override current coordinate system
- `facet`: override current coordinate faceting

If the first object is an object of class `theme`, you can add another theme object. This will return a modified theme object.

For theme objects, the `+` operator and the `%+replace%` can be used to modify elements in themes.

The `+` operator completely replaces elements with elements from `e2`.

In contrast, the `%+replace%` operator does not replace the entire element; it only updates element properties which are present (not `NULL`) in the second object.

## See Also

[theme](#)

## Examples

```
### Adding objects to a ggplot object
p <- qplot(wt, mpg, colour = hp, data = mtcars)

p + coord_cartesian(ylim = c(0, 40))
p + scale_colour_continuous(breaks = c(100, 300))
p + guides(colour = "colourbar")

# Use a different data frame
m <- mtcars[1:10, ]
p %>% m

### Adding objects to a theme object
# Compare these results of adding theme objects to other theme objects
add_el <- theme_grey() + theme(text = element_text(family = "Times"))
rep_el <- theme_grey() %+replace% theme(text = element_text(family = "Times"))

add_el$text
rep_el$text
```

---

add_theme	<i>Modify properties of an element in a theme object</i>
-----------	----------------------------------------------------------

---

### Description

Modify properties of an element in a theme object

### Usage

```
add_theme(t1, t2, t2name)
```

### Arguments

t1	A theme object
t2	A theme object that is to be added to t1
t2name	A name of the t2 object. This is used for printing informative error messages.

### See Also

`+.gg`

---

aes	<i>Generate aesthetic mappings that describe how variables in the data are mapped to visual properties (aesthetics) of geoms.</i>
-----	-----------------------------------------------------------------------------------------------------------------------------------

---

### Description

`aes` creates a list of unevaluated expressions. This function also performs partial name matching, converts color to colour, and old style R names to ggplot names (eg. `pch` to `shape`, `cex` to `size`)

### Usage

```
aes(x, y, ...)
```

### Arguments

x, y, ...	List of name value pairs giving aesthetics to map.
-----------	----------------------------------------------------

### See Also

See [aes\\_colour\\_fill\\_alpha](#), [aes\\_group\\_order](#), [aes\\_linetype\\_size\\_shape](#) and [aes\\_position](#) for more specific examples with different aesthetics.

Other aesthetic generators: [aes\\_q](#), [aes\\_string](#)

### Examples

```
aes(x = mpg, y = wt)
aes(x = mpg ^ 2, y = wt / cyl)
```



---

aes_all	<i>Given a character vector, create a set of identity mappings</i>
---------	--------------------------------------------------------------------

---

**Description**

Given a character vector, create a set of identity mappings

**Usage**

```
aes_all(vars)
```

**Arguments**

vars	vector of variable names
------	--------------------------

**Examples**

```
aes_all(names(mtcars))  
aes_all(c("x", "y", "col", "pch"))
```

---

aes_auto	<i>Automatic aesthetic mapping</i>
----------	------------------------------------

---

**Description**

Automatic aesthetic mapping

**Usage**

```
aes_auto(data = NULL, ...)
```

**Arguments**

data	data.frame or names of variables
...	aesthetics that need to be explicitly mapped.

**Examples**

```
df <- data.frame(x = 1, y = 1, colour = 1, label = 1, pch = 1)  
aes_auto(df)  
aes_auto(names(df))  
  
df <- data.frame(xp = 1, y = 1, colour = 1, txt = 1, foo = 1)  
aes_auto(df, x = xp, label = txt)  
aes_auto(names(df), x = xp, label = txt)  
  
df <- data.frame(foo = 1:3)  
aes_auto(df, x = xp, y = yp)  
aes_auto(df)
```

---

**aes\_colour\_fill\_alpha** *Colour related aesthetics: colour, fill and alpha*

---

**Description**

This page demonstrates the usage of a sub-group of aesthetics; colour, fill and alpha.

**Examples**

```
# Bar chart example
c <- ggplot(mtcars, aes(factor(cyl)))
# Default plotting
c + geom_bar()
# To change the interior colouring use fill aesthetic
c + geom_bar(fill = "red")
# Compare with the colour aesthetic which changes just the bar outline
c + geom_bar(colour = "red")
# Combining both, you can see the changes more clearly
c + geom_bar(fill = "white", colour = "red")

# The aesthetic fill also takes different colouring scales
# setting fill equal to a factor variable uses a discrete colour scale
k <- ggplot(mtcars, aes(factor(cyl), fill = factor(vs)))
k + geom_bar()

# Fill aesthetic can also be used with a continuous variable
m <- ggplot(movies, aes(x = rating))
m + geom_histogram()
m + geom_histogram(aes(fill = ..count..))

# Some geoms don't use both aesthetics (i.e. geom_point or geom_line)
b <- ggplot(economics, aes(x = date, y = unemploy))
b + geom_line()
b + geom_line(colour = "green")
b + geom_point()
b + geom_point(colour = "red")

# For large datasets with overplotting the alpha
# aesthetic will make the points more transparent
df <- data.frame(x = rnorm(5000), y = rnorm(5000))
h <- ggplot(df, aes(x,y))
h + geom_point()
h + geom_point(alpha = 0.5)
h + geom_point(alpha = 1/10)

#If a geom uses both fill and colour, alpha will only modify the fill colour
c + geom_bar(fill = "dark grey", colour = "black")
c + geom_bar(fill = "dark grey", colour = "black", alpha = 1/3)
```

```
# Alpha can also be used to add shading
j <- b + geom_line()
j
yrng <- range(economics$unemploy)
j <- j + geom_rect(aes(NULL, NULL, xmin = start, xmax = end, fill = party),
  ymin = yrng[1], ymax = yrng[2], data = presidential)
j
library(scales) # to access the alpha function
j + scale_fill_manual(values = alpha(c("blue", "red"), .3))
```

---

aes_group_order	<i>Aesthetics: group, order</i>
-----------------	---------------------------------

---

## Description

Aesthetics: group, order

## Examples

```
# By default, the group is set to the interaction of all discrete variables in the
# plot. This often partitions the data correctly, but when it does not, or when
# no discrete variable is used in the plot, you will need to explicitly define the
# grouping structure, by mapping group to a variable that has a different value
# for each group.
```

```
# For most applications you can simply specify the grouping with
# various aesthetics (colour, shape, fill, linetype) or with facets.
```

```
p <- ggplot(mtcars, aes(wt, mpg))
# A basic scatter plot
p + geom_point(size = 4)
# The colour aesthetic
p + geom_point(aes(colour = factor(cyl)), size = 4)
# Or you can use shape to distinguish the data
p + geom_point(aes(shape = factor(cyl)), size = 4)
```

```
# Using fill
a <- ggplot(mtcars, aes(factor(cyl)))
a + geom_bar()
a + geom_bar(aes(fill = factor(cyl)))
a + geom_bar(aes(fill = factor(vs)))
```

```
# Using linetypes
library(reshape2) # for melt
library(plyr) # for colwise
rescale01 <- function(x) (x - min(x)) / diff(range(x))
```

```

ec_scaled <- data.frame(
  date = economics$date,
  colwise(rescale01)(economics[, -(1:2)]))
ecm <- melt(ec_scaled, id = "date")
f <- ggplot(ecm, aes(date, value))
f + geom_line(aes(linetype = variable))

# Using facets
k <- ggplot(diamonds, aes(carat, ..density..)) + geom_histogram(binwidth = 0.2)
k + facet_grid(. ~ cut)

# There are three common cases where the default is not enough, and we
# will consider each one below. In the following examples, we will use a simple
# longitudinal dataset, Oxboys, from the nlme package. It records the heights
# (height) and centered ages (age) of 26 boys (Subject), measured on nine
# occasions (Occasion).

# Multiple groups with one aesthetic
library(nlme)
h <- ggplot(Oxboys, aes(age, height))
# A single line tries to connect all the observations
h + geom_line()
# The group aesthetic maps a different line for each subject
h + geom_line(aes(group = Subject))

# Different groups on different layers
h <- h + geom_line(aes(group = Subject))
# Using the group aesthetic with both geom_line() and geom_smooth()
# groups the data the same way for both layers
h + geom_smooth(aes(group = Subject), method = "lm", se = FALSE)
# Changing the group aesthetic for the smoother layer
# fits a single line of best fit across all boys
h + geom_smooth(aes(group = 1), size = 2, method = "lm", se = FALSE)

# Overriding the default grouping
# The plot has a discrete scale but you want to draw lines that connect across
# groups. This is the strategy used in interaction plots, profile plots, and parallel
# coordinate plots, among others. For example, we draw boxplots of height at
# each measurement occasion
boysbox <- ggplot(Oxboys, aes(Occasion, height))
boysbox + geom_boxplot()
# There is no need to specify the group aesthetic here; the default grouping
# works because occasion is a discrete variable. To overlay individual trajectories
# we again need to override the default grouping for that layer with aes(group = Subject)
boysbox <- boysbox + geom_boxplot()
boysbox + geom_line(aes(group = Subject), colour = "blue")

# Use the order aesthetic to change stacking order of bar charts
w <- ggplot(diamonds, aes(clarity, fill = cut))
w + geom_bar()
w + geom_bar(aes(order = desc(cut)))

# Can also be used to change plot order of scatter plots

```

```
d <- ggplot(diamonds, aes(carat, price, colour = cut))
d + geom_point()
d + geom_point(aes(order = sample(seq_along(carat)))))
```

---

aes\_linetype\_size\_shape

*Differentiation related aesthetics: linetype, size, shape*

---

## Description

This page demonstrates the usage of a sub-group of aesthetics; linetype, size and shape.

## Examples

```
# Line types should be specified with either an integer, a name, or with a string of
# an even number (up to eight) of hexadecimal digits which give the lengths in
# consecutive positions in the string.
# 0 = blank, 1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash

# Data
df <- data.frame(x = 1:10, y = 1:10)
f <- ggplot(df, aes(x = x, y = y))
f + geom_line(linetype = 2)
f + geom_line(linetype = "dotdash")
# An example with hex strings, the string "33" specifies three units on followed
# by three off and "3313" specifies three units on followed by three off followed
# by one on and finally three off.
f + geom_line(linetype = "3313")

# Mapping line type from a variable
library(plyr)
library(reshape2)
rescale01 <- function(x) (x - min(x)) / diff(range(x))
ec_scaled <- data.frame(
  date = economics$date,
  colwise(rescale01)(economics[, -(1:2)]))
ecm <- melt(ec_scaled, id = "date")
qplot(date, value, data = ecm, geom = "line", linetype = variable)

# Size examples
# Should be specified with a numerical value (in millimetres),
# or from a variable source
p <- ggplot(mtcars, aes(wt, mpg))
p + geom_point(size = 4)
p + geom_point(aes(size = qsec))
p + geom_point(size = 2.5) + geom_hline(yintercept = 25, size = 3.5)

# Shape examples
# Shape takes four types of values: an integer in [0, 25],
```

```
# a single character-- which uses that character as the plotting symbol,
# a . to draw the smallest rectangle that is visible (i.e., about one pixel)
# an NA to draw nothing
p + geom_point()
p + geom_point(shape = 5)
p + geom_point(shape = "k", size = 3)
p + geom_point(shape = ".")
p + geom_point(shape = NA)

# Shape can also be mapped from a variable
p + geom_point(aes(shape = factor(cyl)))

# A look at all 25 symbols
df2 <- data.frame(x = 1:5 , y = 1:25, z = 1:25)
s <- ggplot(df2, aes(x = x, y = y))
s + geom_point(aes(shape = z), size = 4) + scale_shape_identity()
# While all symbols have a foreground colour, symbols 19-25 also take a
# background colour (fill)
s + geom_point(aes(shape = z), size = 4, colour = "Red") +
  scale_shape_identity()
s + geom_point(aes(shape = z), size = 4, colour = "Red", fill = "Black") +
  scale_shape_identity()
```

---

aes\_position

*Position related aesthetics: x, y, xmin, xmax, ymin, ymax, xend, yend*

---

## Description

This page demonstrates the usage of a sub-group of aesthetics; x, y, xmin, xmax, ymin, ymax, xend, and yend.

## Examples

```
# Generate data: means and standard errors of means for prices
# for each type of cut
dmod <- lm(price ~ cut, data = diamonds)
cuts <- data.frame(cut = unique(diamonds$cut), predict(dmod, data.frame(cut =
unique(diamonds$cut)), se = TRUE)[c("fit", "se.fit")])
se <- ggplot(cuts, aes(x = cut, y = fit, ymin = fit - se.fit,
ymax = fit + se.fit, colour = cut))
se + geom_pointrange()

# Boxplot with precomputed statistics
# generate sample data
library(plyr)
abc <- adply(matrix(rnorm(100), ncol = 5), 2, quantile, c(0, .25, .5, .75, 1))
b <- ggplot(abc, aes(x = X1, ymin = "0%", lower = "25%",
middle = "50%", upper = "75%", ymax = "100%"))
b + geom_boxplot(stat = "identity")
```

```
# Using annotate
p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
p + annotate("rect", xmin = 2, xmax = 3.5, ymin = 2, ymax = 25,
  fill = "dark grey", alpha = .5)

# Geom_segment examples
library(grid)
p + geom_segment(aes(x = 2, y = 15, xend = 2, yend = 25),
  arrow = arrow(length = unit(0.5, "cm")))
p + geom_segment(aes(x = 2, y = 15, xend = 3, yend = 15),
  arrow = arrow(length = unit(0.5, "cm")))
p + geom_segment(aes(x = 5, y = 30, xend = 3.5, yend = 25),
  arrow = arrow(length = unit(0.5, "cm")))

# You can also use geom_segment to recreate plot(type = "h") :
counts <- as.data.frame(table(x = rpois(100, 5)))
counts$x <- as.numeric(as.character(counts$x))
with(counts, plot(x, Freq, type = "h", lwd = 10))

qplot(x, Freq, data = counts, geom = "segment", yend = 0, xend = x,
  size = I(10))
```

---

aes\_string

---

*Generate aesthetic mappings from a string/quoted objects*


---

## Description

Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of geoms. `aes` uses non-standard evaluation to capture the variable names. These two variants use regular evaluation, which is easier to use inside functions.

## Usage

```
aes_string(x = NULL, y = NULL, ...)
```

```
aes_q(x = NULL, y = NULL, ...)
```

## Arguments

`x, y, ...` List of name value pairs

## Details

`aes_string` and `aes_q` are particularly useful when writing functions that create plots because you can use strings or quoted names/calls to define the aesthetic mappings, rather than having to use `substitute` to generate a call to `aes()`.

**See Also**[aes](#)Other aesthetic generators: [aes](#)**Examples**

```
# Three ways of generating the same aesthetics
aes(mpg, wt, col = cyl, fill = NULL)
aes_string("mpg", "wt", col = "cyl", fill = NULL)
aes_q(quote(mpg), quote(wt), col = quote(cyl), fill = NULL)

aes(col = cyl, fill = NULL)
aes_string(col = "cyl", fill = NULL)
aes_q(col = quote(cyl), fill = NULL)
```

---

annotate	<i>Create an annotation layer.</i>
----------	------------------------------------

---

**Description**

This function adds geoms to a plot. Unlike typical a geom function, the properties of the geoms are not mapped from variables of a data frame, but are instead in as vectors. This is useful for adding small annotations (such as text labels) or if you have your data in vectors, and for some reason don't want to put them in a data frame.

**Usage**

```
annotate(geom, x = NULL, y = NULL, xmin = NULL, xmax = NULL,
         ymin = NULL, ymax = NULL, ...)
```

**Arguments**

geom	name of geom to use for annotation
x, y, xmin, ymin, xmax, ymax	positioning aesthetics - you must specify at least one of these.
...	other aesthetics. These are not scaled so you can do (e.g.) colour = "red" to get a red point.

**Details**

Note that all position aesthetics are scaled (i.e. they will expand the limits of the plot so they are visible), but all other aesthetics are set. This means that layers created with this function will never affect the legend.



## Examples

```
p <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()
p + annotate("text", x = 4, y = 25, label = "Some text")
p + annotate("text", x = 2:5, y = 25, label = "Some text")
p + annotate("rect", xmin = 3, xmax = 4.2, ymin = 12, ymax = 21,
  alpha = .2)
p + annotate("segment", x = 2.5, xend = 4, y = 15, yend = 25,
  colour = "blue")
p + annotate("pointrange", x = 3.5, y = 20, ymin = 12, ymax = 28,
  colour = "red", size = 1.5)

p + annotate("text", x = 2:3, y = 20:21, label = c("my label", "label 2"))
```

---

annotation_custom	<i>Annotation: Custom grob.</i>
-------------------	---------------------------------

---

## Description

This is a special geom intended for use as static annotations that are the same in every panel. These annotations will not affect scales (i.e. the x and y axes will not grow to cover the range of the grob, and the grob will not be modified by any ggplot settings or mappings).

## Usage

```
annotation_custom(grob, xmin = -Inf, xmax = Inf, ymin = -Inf,
  ymax = Inf)
```

## Arguments

<code>grob</code>	grob to display
<code>xmin, xmax</code>	x location (in data coordinates) giving horizontal location of raster
<code>ymin, ymax</code>	y location (in data coordinates) giving vertical location of raster

## Details

Most useful for adding tables, inset plots, and other grid-based decorations.

## Note

annotation\_custom expects the grob to fill the entire viewport defined by xmin, xmax, ymin, ymax. Grobs with a different (absolute) size will be center-justified in that region. Inf values can be used to fill the full plot panel (see examples).

## Examples

```
# Dummy plot
base <- qplot(1:10, 1:10, geom = "blank") + theme_bw()
# Adding a table
if (require(gridExtra)) {
  base + annotation_custom(grob = tableGrob(head(iris[,1:3])),
    xmin = 3, xmax = 6, ymin = 2, ymax = 8)
# full panel
base + annotation_custom(grob = roundrectGrob(),
  xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = Inf)
}
# Inset plot
g <- ggplotGrob(qplot(1, 1) +
  theme(plot.background = element_rect(colour = "black")))
base +
  annotation_custom(grob = g, xmin = 1, xmax = 10, ymin = 8, ymax = 10)
```

---

annotation\_logticks     *Annotation: log tick marks*

---

## Description

This annotation adds log tick marks with diminishing spacing. These tick marks probably make sense only for base 10.

## Usage

```
annotation_logticks(base = 10, sides = "bl", scaled = TRUE,
  short = unit(0.1, "cm"), mid = unit(0.2, "cm"), long = unit(0.3, "cm"),
  colour = "black", size = 0.5, linetype = 1, alpha = 1, color = NULL,
  ...)
```

## Arguments

base	the base of the log (default 10)
sides	a string that controls which sides of the plot the log ticks appear on. It can be set to a string containing any of "trbl", for top, right, bottom, and left.
short	a unit object specifying the length of the short tick marks
mid	a unit object specifying the length of the middle tick marks. In base 10, these are the "5" ticks.
long	a unit object specifying the length of the long tick marks. In base 10, these are the "1" (or "10") ticks.
scaled	is the data already log-scaled? This should be TRUE (default) when the data is already transformed with log10() or when using scale_y_log10. It should be FALSE when using coord_trans(y = "log10").
colour	Colour of the tick marks.

size	Thickness of tick marks, in mm.
linetype	Linetype of tick marks (solid, dashed, etc.)
alpha	The transparency of the tick marks.
color	An alias for colour.
...	Other parameters passed on to the layer

**See Also**

[scale\\_y\\_continuous](#), [scale\\_y\\_log10](#) for log scale transformations.

[coord\\_trans](#) for log coordinate transformations.

**Examples**

```
# Make a log-log plot (without log ticks)
library(MASS)
library(scales)
a <- ggplot(Animals, aes(x = body, y = brain)) + geom_point() +
  scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x),
               labels = trans_format("log10", math_format(10^.x))) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
               labels = trans_format("log10", math_format(10^.x))) +
  theme_bw()

a + annotation_logticks()           # Default: log ticks on bottom and left
a + annotation_logticks(sides = "lr") # Log ticks for y, on left and right
a + annotation_logticks(sides = "trbl") # All four sides

# Hide the minor grid lines because they don't align with the ticks
a + annotation_logticks(sides = "trbl") + theme(panel.grid.minor = element_blank())

# Another way to get the same results as 'a' above: log-transform the data before
b <- ggplot(Animals, aes(x = log10(body), y = log10(brain))) + geom_point() +
  scale_x_continuous(name = "body", labels = math_format(10^.x)) +
  scale_y_continuous(name = "brain", labels = math_format(10^.x)) +
  theme_bw() + theme(panel.grid.minor = element_blank())

b + annotation_logticks()

# This shows log(x) on the axes
d <- ggplot(Animals, aes(x = log10(body), y = log10(brain))) + geom_point() +
  theme_bw()

d + annotation_logticks()

# Using a coordinate transform requires scaled = FALSE
t <- ggplot(Animals, aes(x = body, y = brain)) + geom_point() +
  coord_trans(xtrans = "log10", ytrans = "log10") + theme_bw()
```

```
t + annotation_logticks(scaled = FALSE)

# Change the length of the ticks
library(grid)
a + annotation_logticks(short = unit(.5,"mm"), mid = unit(3,"mm"), long = unit(4,"mm"))
```

---

annotation_map	<i>Annotation: maps.</i>
----------------	--------------------------

---

## Description

Annotation: maps.

## Usage

```
annotation_map(map, ...)
```

## Arguments

map	data frame representing a map. Most map objects can be converted into the right format by using <a href="#">fortify</a>
...	other arguments used to modify aesthetics

## Examples

```
library(maps)
usamap <- map_data("state")

seal.sub <- subset(seals, long > -130 & lat < 45 & lat > 40)
ggplot(seal.sub, aes(x = long, y = lat)) +
  annotation_map(usamap, fill = "NA", colour = "grey50") +
  geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat))

seal2 <- transform(seal.sub,
  latr = cut(lat, 2),
  longr = cut(long, 2))

ggplot(seal2, aes(x = long, y = lat)) +
  annotation_map(usamap, fill = "NA", colour = "grey50") +
  geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat)) +
  facet_grid(latr ~ longr, scales = "free", space = "free")
```

---

annotation_raster	<i>Annotation: High-performance rectangular tiling.</i>
-------------------	---------------------------------------------------------

---

## Description

This is a special version of [geom\\_raster](#) optimised for static annotations that are the same in every panel. These annotations will not affect scales (i.e. the x and y axes will not grow to cover the range of the raster, and the raster must already have its own colours).

## Usage

```
annotation_raster(raster, xmin, xmax, ymin, ymax, interpolate = FALSE)
```

## Arguments

raster	raster object to display
xmin, xmax	x location (in data coordinates) giving horizontal location of raster
ymin, ymax	y location (in data coordinates) giving vertical location of raster
interpolate	If TRUE interpolate linearly, if FALSE (the default) don't interpolate.

## Details

Most useful for adding bitmap images.

## Examples

```
# Generate data
rainbow <- matrix(hcl(seq(0, 360, length = 50 * 50), 80, 70), nrow = 50)
qplot(mpg, wt, data = mtcars) +
  annotation_raster(rainbow, 15, 20, 3, 4)
# To fill up whole plot
qplot(mpg, wt, data = mtcars) +
  annotation_raster(rainbow, -Inf, Inf, -Inf, Inf) +
  geom_point()

rainbow2 <- matrix(hcl(seq(0, 360, length = 10), 80, 70), nrow = 1)
qplot(mpg, wt, data = mtcars) +
  annotation_raster(rainbow2, -Inf, Inf, -Inf, Inf) +
  geom_point()

rainbow2 <- matrix(hcl(seq(0, 360, length = 10), 80, 70), nrow = 1)
qplot(mpg, wt, data = mtcars) +
  annotation_raster(rainbow2, -Inf, Inf, -Inf, Inf, interpolate = TRUE) +
  geom_point()
```

---

`autoplot`*Create a complete ggplot appropriate to a particular data type*

---

**Description**

`autoplot` uses `ggplot2` to draw a particular plot for an object of a particular class in a single command. This defines the S3 generic that other classes and packages can extend.

**Usage**

```
autoplot(object, ...)
```

**Arguments**

<code>object</code>	an object, whose class will determine the behaviour of <code>autoplot</code>
<code>...</code>	other arguments passed to specific methods

**Value**

a `ggplot` object

**See Also**

[ggplot](#) and [fortify](#)

---

`borders`*Create a layer of map borders.*

---

**Description**

Create a layer of map borders.

**Usage**

```
borders(database = "world", regions = ".", fill = NA, colour = "grey50",  
...)
```

**Arguments**

<code>database</code>	map data, see <a href="#">map</a> for details
<code>regions</code>	map region
<code>fill</code>	fill colour
<code>colour</code>	border colour
<code>...</code>	other arguments passed onto <a href="#">geom_polygon</a>

**Examples**

```

if (require("maps")) {

  ia <- map_data("county", "iowa")
  mid_range <- function(x) mean(range(x))
  library(plyr)
  seats <- ddply(ia, .(subregion), colwise(mid_range, .(lat, long)))
  ggplot(ia, aes(long, lat)) +
    geom_polygon(aes(group = group), fill = NA, colour = "grey60") +
    geom_text(aes(label = subregion), data = seats, size = 2, angle = 45)

  data(us.cities)
  capitals <- subset(us.cities, capital == 2)
  ggplot(capitals, aes(long, lat)) +
    borders("state") +
    geom_point(aes(size = pop)) +
    scale_size_area()

}

```

---

calc_element	<i>Calculate the element properties, by inheriting properties from its parents</i>
--------------	------------------------------------------------------------------------------------

---

**Description**

Calculate the element properties, by inheriting properties from its parents

**Usage**

```
calc_element(element, theme, verbose = FALSE)
```

**Arguments**

element	The name of the theme element to calculate
theme	A theme object (like theme_grey())
verbose	If TRUE, print out which elements this one inherits from

**Examples**

```

t <- theme_grey()
calc_element('text', t)

# Compare the "raw" element definition to the element with calculated inheritance
t$axis.text.x
calc_element('axis.text.x', t, verbose = TRUE)

# This reports that axis.text.x inherits from axis.text,
# which inherits from text. You can view each of them with:

```

```
t$axis.text.x
t$axis.text
t$text
```

---

coord_cartesian	<i>Cartesian coordinates.</i>
-----------------	-------------------------------

---

## Description

The Cartesian coordinate system is the most familiar, and common, type of coordinate system. Setting limits on the coordinate system will zoom the plot (like you're looking at it with a magnifying glass), and will not change the underlying data like setting limits on a scale will.

## Usage

```
coord_cartesian(xlim = NULL, ylim = NULL, wise = NULL)
```

## Arguments

xlim	limits for the x axis
ylim	limits for the y axis
wise	deprecated in 0.9.1

## Examples

```
# There are two ways of zooming the plot display: with scales or
# with coordinate systems. They work in two rather different ways.

(p <- qplot(displ, wt, data=mtcars) + geom_smooth())

# Setting the limits on a scale will throw away all data that's not
# inside these limits. This is equivalent to plotting a subset of
# the original data
p + scale_x_continuous(limits = c(325, 500))

# Setting the limits on the coordinate system performs a visual zoom
# the data is unchanged, and we just view a small portion of the original
# plot. See how the axis labels are the same as the original data, and
# the smooth continue past the points visible on this plot.
p + coord_cartesian(xlim = c(325, 500))

# You can see the same thing with this 2d histogram
(d <- ggplot(diamonds, aes(carat, price)) +
  stat_bin2d(bins = 25, colour="grey50"))

# When zooming the scale, the we get 25 new bins that are the same
# size on the plot, but represent smaller regions of the data space
d + scale_x_continuous(limits = c(0, 2))
```



```
# When zooming the coordinate system, we see a subset of original 50 bins,
# displayed bigger
d + coord_cartesian(xlim = c(0, 2))
```

---

coord\_fixed

*Cartesian coordinates with fixed relationship between x and y scales.*


---

## Description

A fixed scale coordinate system forces a specified ratio between the physical representation of data units on the axes. The ratio represents the number of units on the y-axis equivalent to one unit on the x-axis. The default, `ratio = 1`, ensures that one unit on the x-axis is the same length as one unit on the y-axis. Ratios higher than one make units on the y axis longer than units on the x-axis, and vice versa. This is similar to [eqsplot](#), but it works for all types of graphics.

## Usage

```
coord_fixed(ratio = 1, xlim = NULL, ylim = NULL, wise = NULL)
```

## Arguments

ratio	aspect ratio, expressed as $y / x$
xlim	limits for the x axis
ylim	limits for the y axis
wise	deprecated in 0.9.1

## Examples

```
# ensures that the ranges of axes are equal to the specified ratio by
# adjusting the plot aspect ratio

qplot(mpg, wt, data = mtcars) + coord_fixed(ratio = 1)
qplot(mpg, wt, data = mtcars) + coord_fixed(ratio = 5)
qplot(mpg, wt, data = mtcars) + coord_fixed(ratio = 1/5)

# Resize the plot to see that the specified aspect ratio is maintained
```

---

coord_flip	<i>Flipped cartesian coordinates.</i>
------------	---------------------------------------

---

**Description**

Flipped cartesian coordinates so that horizontal becomes vertical, and vertical, horizontal. This is primarily useful for converting geoms and statistics which display y conditional on x, to x conditional on y.

**Usage**

```
coord_flip(...)
```

**Arguments**

... Other arguments passed onto [coord\\_cartesian](#)

**Examples**

```
# Very useful for creating boxplots, and other interval
# geoms in the horizontal instead of vertical position.
qplot(cut, price, data=diamonds, geom="boxplot")
last_plot() + coord_flip()

qplot(cut, data=diamonds, geom="bar")
last_plot() + coord_flip()

h <- qplot(carat, data=diamonds, geom="histogram")
h
h + coord_flip()
h + coord_flip() + scale_x_reverse()

# You can also use it to flip lines and area plots:
qplot(1:5, (1:5)^2, geom="area")
last_plot() + coord_flip()
```

---

coord_map	<i>Map projections.</i>
-----------	-------------------------

---

**Description**

This coordinate system provides the full range of map projections available in the mapproj package.

**Usage**

```
coord_map(projection = "mercator", ..., orientation = NULL, xlim = NULL,
          ylim = NULL)
```

**Arguments**

projection	projection to use, see <a href="#">mapproject</a> for list
...	other arguments passed on to <a href="#">mapproject</a>
orientation	projection orientation, which defaults to <code>c(90, 0, mean(range(x)))</code> . This is not optimal for many projections, so you will have to supply your own. See <a href="#">mapproject</a> for more information.
xlim	manually specific x limits (in degrees of longitude)
ylim	manually specific y limits (in degrees of latitude)

**Details**

This is still experimental, and if you have any advice to offer regarding a better (or more correct) way to do this, please let me know

**Examples**

```
if (require("maps")) {
  # Create a lat-long dataframe from the maps package
  nz <- map_data("nz")
  nzmap <- ggplot(nz, aes(x=long, y=lat, group=group)) +
    geom_polygon(fill="white", colour="black")

  # Use cartesian coordinates
  nzmap
  # With default mercator projection
  nzmap + coord_map()
  # Other projections
  nzmap + coord_map("cylindrical")
  nzmap + coord_map("azequalarea",orientation=c(-36.92,174.6,0))

  states <- map_data("state")
  usamap <- ggplot(states, aes(x=long, y=lat, group=group)) +
    geom_polygon(fill="white", colour="black")

  # Use cartesian coordinates
  usamap
  # With mercator projection
  usamap + coord_map()
  # See ?mapproject for coordinate systems and their parameters
  usamap + coord_map("gilbert")
  usamap + coord_map("lagrange")

  # For most projections, you'll need to set the orientation yourself
  # as the automatic selection done by mapproject is not available to
  # ggplot
```

```

usamap + coord_map("orthographic")
usamap + coord_map("stereographic")
usamap + coord_map("conic", lat0 = 30)
usamap + coord_map("bonne", lat0 = 50)

# World map, using geom_path instead of geom_polygon
world <- map_data("world")
worldmap <- ggplot(world, aes(x=long, y=lat, group=group)) +
  geom_path() +
  scale_y_continuous(breaks=(-2:2) * 30) +
  scale_x_continuous(breaks=(-4:4) * 45)

# Orthographic projection with default orientation (looking down at North pole)
worldmap + coord_map("ortho")
# Looking up up at South Pole
worldmap + coord_map("ortho", orientation=c(-90, 0, 0))
# Centered on New York (currently has issues with closing polygons)
worldmap + coord_map("ortho", orientation=c(41, -74, 0))
}

```

---

coord\_polar

*Polar coordinates.*


---

## Description

The polar coordinate system is most commonly used for pie charts, which are a stacked bar chart in polar coordinates.

## Usage

```
coord_polar(theta = "x", start = 0, direction = 1)
```

## Arguments

theta	variable to map angle to (x or y)
start	offset of starting point from 12 o'clock in radians
direction	1, clockwise; -1, anticlockwise

## Examples

```

# NOTE: Use these plots with caution - polar coordinates has
# major perceptual problems. The main point of these examples is
# to demonstrate how these common plots can be described in the
# grammar. Use with EXTREME caution.

# A coxcomb plot = bar chart + polar coordinates
cxc <- ggplot(mtcars, aes(x = factor(cyl))) +
  geom_bar(width = 1, colour = "black")

```

```

cxc + coord_polar()
# A new type of plot?
cxc + coord_polar(theta = "y")

# A pie chart = stacked bar chart + polar coordinates
pie <- ggplot(mtcars, aes(x = factor(1), fill = factor(cyl))) +
  geom_bar(width = 1)
pie + coord_polar(theta = "y")

# The bullseye chart
pie + coord_polar()

# Hadley's favourite pie chart
df <- data.frame(
  variable = c("resembles", "does not resemble"),
  value = c(80, 20)
)
ggplot(df, aes(x = "", y = value, fill = variable)) +
  geom_bar(width = 1, stat = "identity") +
  scale_fill_manual(values = c("red", "yellow")) +
  coord_polar("y", start = pi / 3) +
  labs(title = "Pac man")

# Windrose + doughnut plot
movies$rrating <- cut_interval(movies$rating, length = 1)
movies$budgetq <- cut_number(movies$budget, 4)

doh <- ggplot(movies, aes(x = rrating, fill = budgetq))

# Wind rose
doh + geom_bar(width = 1) + coord_polar()
# Race track plot
doh + geom_bar(width = 0.9, position = "fill") + coord_polar(theta = "y")

```

---

coord\_quickmap

*Cartesian coordinates with an aspect ratio approximating Mercator projection.*


---

## Description

The representation of a portion of the earth, which is approximately spherical, onto a flat 2D plane requires a projection. This is what `coord_map` does. These projections account for the fact that the actual length (in km) of one degree of longitude varies between the equator and the pole. Near the equator, the ratio between the lengths of one degree of latitude and one degree of longitude is approximately 1. Near the pole, it tends towards infinity because the length of one degree of longitude tends towards 0. For regions that span only a few degrees and are not too close to the poles, setting the aspect ratio of the plot to the appropriate lat/lon ratio approximates the usual mercator projection. This is what `coord_quickmap` does. With `coord_map` all elements of the graphic have to be projected which is not the case here. So `coord_quickmap` has the advantage

of being much faster, in particular for complex plots such as those using with `geom_tile`, at the expense of correctness in the projection.

### Usage

```
coord_quickmap(xlim = NULL, ylim = NULL)
```

### Arguments

<code>xlim</code>	limits for the x axis
<code>ylim</code>	limits for the y axis

### Examples

```
# ensures that the ranges of axes are equal to the specified ratio by
# adjusting the plot aspect ratio

if (require("maps")) {
  # Create a lat-long dataframe from the maps package
  nz <- map_data("nz")
  # Prepare a plot of the map
  nzmap <- ggplot(nz, aes(x = long, y = lat, group = group)) +
    geom_polygon(fill = "white", colour = "black")

  # Plot it in cartesian coordinates
  nzmap
  # With correct mercator projection
  nzmap + coord_map()
  # With the aspect ratio approximation
  nzmap + coord_quickmap()
}

# Resize the plot to see that the specified aspect ratio is maintained
```

---

coord\_trans

*Transformed cartesian coordinate system.*

---

### Description

`coord_trans` is different to scale transformations in that it occurs after statistical transformation and will affect the visual appearance of geoms - there is no guarantee that straight lines will continue to be straight.

### Usage

```
coord_trans(xtrans = "identity", ytrans = "identity", limx = NULL,
  limy = NULL)
```

## Arguments

xtrans,ytrans    transformers for x and y axes  
 limx,limy        limits for x and y axes. (Named so for backward compatability)

## Details

All current transformations only work with continuous values - see [trans\\_new](#) for list of transformations, and instructions on how to create your own.

## Examples

```
# See ?geom_boxplot for other examples

# Three ways of doing transforming in ggplot:
# * by transforming the data
qplot(log10(carat), log10(price), data=diamonds)
# * by transforming the scales
qplot(carat, price, data=diamonds, log="xy")
qplot(carat, price, data=diamonds) + scale_x_log10() + scale_y_log10()
# * by transforming the coordinate system:
qplot(carat, price, data=diamonds) + coord_trans(x = "log10", y = "log10")

# The difference between transforming the scales and
# transforming the coordinate system is that scale
# transformation occurs BEFORE statistics, and coordinate
# transformation afterwards. Coordinate transformation also
# changes the shape of geoms:

d <- subset(diamonds, carat > 0.5)
qplot(carat, price, data = d, log="xy") +
  geom_smooth(method="lm")
qplot(carat, price, data = d) +
  geom_smooth(method="lm") +
  coord_trans(x = "log10", y = "log10")

# Here I used a subset of diamonds so that the smoothed line didn't
# drop below zero, which obviously causes problems on the log-transformed
# scale

# With a combination of scale and coordinate transformation, it's
# possible to do back-transformations:
library(scales)
qplot(carat, price, data=diamonds, log="xy") +
  geom_smooth(method="lm") +
  coord_trans(x = exp_trans(10), y = exp_trans(10))
# cf.
qplot(carat, price, data=diamonds) + geom_smooth(method = "lm")

# Also works with discrete scales
df <- data.frame(a = abs(rnorm(26)), letters)
```

```
plot <- ggplot(df,aes(a,letters)) + geom_point()

plot + coord_trans(x = "log10")
plot + coord_trans(x = "sqrt")
```

---

cut_interval	<i>Cut numeric vector into intervals of equal length.</i>
--------------	-----------------------------------------------------------

---

### Description

Cut numeric vector into intervals of equal length.

### Usage

```
cut_interval(x, n = NULL, length = NULL, ...)
```

### Arguments

x	numeric vector
n	number of intervals to create, OR
length	length of each interval
...	other arguments passed on to <a href="#">cut</a>

### See Also

[cut\\_number](#)

### Examples

```
table(cut_interval(1:100, n = 10))
table(cut_interval(1:100, n = 11))
table(cut_interval(1:100, length = 10))
```

---

cut_number	<i>Cut numeric vector into intervals containing equal number of points.</i>
------------	-----------------------------------------------------------------------------

---

### Description

Cut numeric vector into intervals containing equal number of points.

### Usage

```
cut_number(x, n = NULL, ...)
```



**Arguments**

x	numeric vector
n	number of intervals to create
...	other arguments passed on to <a href="#">cut</a>

**See Also**

[cut\\_interval](#)

**Examples**

```
table(cut_number(runif(1000), n = 10))
```

---

diamonds	<i>Prices of 50,000 round cut diamonds</i>
----------	--------------------------------------------

---

**Description**

A dataset containing the prices and other attributes of almost 54,000 diamonds. The variables are as follows:

**Usage**

```
data(diamonds)
```

**Format**

A data frame with 53940 rows and 10 variables

**Details**

- price. price in US dollars (\\$326–\\$18,823)
- carat. weight of the diamond (0.2–5.01)
- cut. quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- colour. diamond colour, from J (worst) to D (best)
- clarity. a measurement of how clear the diamond is (I1 (worst), SI1, SI2, VS1, VS2, VVS1, VVS2, IF (best))
- x. length in mm (0–10.74)
- y. width in mm (0–58.9)
- z. depth in mm (0–31.8)
- depth. total depth percentage =  $z / \text{mean}(x, y) = 2 * z / (x + y)$  (43–79)
- table. width of top of diamond relative to widest point (43–95)

---

economics

*US economic time series.*


---

### Description

This dataset was produced from US economic time series data available from <http://research.stlouisfed.org/fred2>.

### Usage

```
data(economics)
```

### Format

A data frame with 478 rows and 6 variables

### Details

- date, Month of data collection
- psavert, personal savings rate, <http://research.stlouisfed.org/fred2/series/PSAVERT/>
- pce, personal consumption expenditures, in billions of dollars, <http://research.stlouisfed.org/fred2/series/PCE>
- unemploy, number of unemployed in thousands, <http://research.stlouisfed.org/fred2/series/UNEMPLOY>
- uempmed, median duration of unemployment, in week, <http://research.stlouisfed.org/fred2/series/UEMP MED>
- pop, total population, in thousands, <http://research.stlouisfed.org/fred2/series/POP>

---

element\_blank

*Theme element: blank. This theme element draws nothing, and assigns no space*


---

### Description

Theme element: blank. This theme element draws nothing, and assigns no space

### Usage

```
element_blank()
```

---

element_line	<i>Theme element: line.</i>
--------------	-----------------------------

---

**Description**

Theme element: line.

**Usage**

```
element_line(colour = NULL, size = NULL, linetype = NULL,  
             lineend = NULL, color = NULL)
```

**Arguments**

colour	line colour
size	line size
linetype	line type
lineend	line end
color	an alias for colour

---

element_rect	<i>Theme element: rectangle.</i>
--------------	----------------------------------

---

**Description**

Most often used for backgrounds and borders.

**Usage**

```
element_rect(fill = NULL, colour = NULL, size = NULL, linetype = NULL,  
             color = NULL)
```

**Arguments**

fill	fill colour
colour	border colour
size	border size
linetype	border linetype
color	an alias for colour

---

element_text	<i>Theme element: text.</i>
--------------	-----------------------------

---

### Description

Theme element: text.

### Usage

```
element_text(family = NULL, face = NULL, colour = NULL, size = NULL,
             hjust = NULL, vjust = NULL, angle = NULL, lineheight = NULL,
             color = NULL)
```

### Arguments

family	font family
face	font face ("plain", "italic", "bold", "bold.italic")
colour	text colour
size	text size (in pts)
hjust	horizontal justification (in [0, 1])
vjust	vertical justification (in [0, 1])
angle	angle (in [0, 360])
lineheight	line height
color	an alias for colour

---

expand_limits	<i>Expand the plot limits with data.</i>
---------------	------------------------------------------

---

### Description

panels or all plots. This function is a thin wrapper around [geom\\_blank](#) that makes it easy to add such values.

### Usage

```
expand_limits(...)
```

### Arguments

...	named list of aesthetics specifying the value (or values) that should be included in each scale.
-----	--------------------------------------------------------------------------------------------------

**Examples**

```
p <- qplot(mpg, wt, data = mtcars)
p + expand_limits(x = 0)
p + expand_limits(y = c(1, 9))
p + expand_limits(x = 0, y = 0)

qplot(mpg, wt, data = mtcars, colour = cyl) +
  expand_limits(colour = seq(2, 10, by = 2))
qplot(mpg, wt, data = mtcars, colour = factor(cyl)) +
  expand_limits(colour = factor(seq(2, 10, by = 2)))
```

---

facet_grid	<i>Lay out panels in a grid.</i>
------------	----------------------------------

---

**Description**

Lay out panels in a grid.

**Usage**

```
facet_grid(facets, margins = FALSE, scales = "fixed", space = "fixed",
  shrink = TRUE, labeller = "label_value", as.table = TRUE, drop = TRUE)
```

**Arguments**

facets	a formula with the rows (of the tabular display) on the LHS and the columns (of the tabular display) on the RHS; the dot in the formula is used to indicate there should be no faceting on this dimension (either row or column). The formula can also be provided as a string instead of a classical formula object
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.
labeller	A function that takes two arguments (variable and value) and returns a string suitable for display in the facet strip. See <a href="#">label_value</a> 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.

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.
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.

## Examples

```
p <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
# With one variable
p + facet_grid(. ~ cyl)
p + facet_grid(cyl ~ .)

# With two variables
p + facet_grid(vs ~ am)
p + facet_grid(am ~ vs)
p + facet_grid(vs ~ am, margins=TRUE)

# To change plot order of facet grid,
# change the order of variable levels with factor()

set.seed(6809)
diamonds <- diamonds[sample(nrow(diamonds), 1000), ]
diamonds$cut <- factor(diamonds$cut,
  levels = c("Ideal", "Very Good", "Fair", "Good", "Premium"))

# Repeat first example with new order
p <- ggplot(diamonds, aes(carat, ..density..)) +
  geom_histogram(binwidth = 1)
p + facet_grid(. ~ cut)

qplot(mpg, wt, data=mtcars, facets = . ~ vs + am)
qplot(mpg, wt, data=mtcars, facets = vs + am ~ .)

# You can also use strings, which makes it a little easier
# when writing functions that generate faceting specifications
# p + facet_grid("cut ~ .")

# see also ?plotmatrix for the scatterplot matrix

# If there isn't any data for a given combination, that panel
# will be empty
qplot(mpg, wt, data=mtcars) + facet_grid(cyl ~ vs)

p <- qplot(mpg, wt, data=mtcars, facets = vs ~ cyl)

df <- data.frame(mpg = 22, wt = 3)
p + geom_point(data = df, colour="red", size = 2)

df2 <- data.frame(mpg = c(19, 22), wt = c(2,4), vs = c(0, 1))
p + geom_point(data = df2, colour="red", size = 2)
```

```

df3 <- data.frame(mpg = c(19, 22), wt = c(2,4), vs = c(1, 1))
p + geom_point(data = df3, colour="red", size = 2)

# You can also choose whether the scales should be constant
# across all panels (the default), or whether they should be allowed
# to vary
mt <- ggplot(mtcars, aes(mpg, wt, colour = factor(cyl))) + geom_point()

mt + facet_grid(. ~ cyl, scales = "free")
# If scales and space are free, then the mapping between position
# and values in the data will be the same across all panels
mt + facet_grid(. ~ cyl, scales = "free", space = "free")

mt + facet_grid(vs ~ am, scales = "free")
mt + facet_grid(vs ~ am, scales = "free_x")
mt + facet_grid(vs ~ am, scales = "free_y")
mt + facet_grid(vs ~ am, scales = "free", space="free")
mt + facet_grid(vs ~ am, scales = "free", space="free_x")
mt + facet_grid(vs ~ am, scales = "free", space="free_y")

# You may need to set your own breaks for consistent display:
mt + facet_grid(. ~ cyl, scales = "free_x", space="free") +
  scale_x_continuous(breaks = seq(10, 36, by = 2))
# Adding scale limits override free scales:
last_plot() + xlim(10, 15)

# Free scales are particularly useful for categorical variables
qplot(cty, model, data=mpg) +
  facet_grid(manufacturer ~ ., scales = "free", space = "free")
# particularly when you reorder factor levels
mpg <- within(mpg, {
  model <- reorder(model, cty)
  manufacturer <- reorder(manufacturer, cty)
})
last_plot() %+% mpg + theme(strip.text.y = element_text())

# Use as.table to control direction of horizontal facets, TRUE by default
h <- ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point()
h + facet_grid(cyl ~ vs)
h + facet_grid(cyl ~ vs, as.table = FALSE)

# Use labeller to control facet labels, label_value is default
h + facet_grid(cyl ~ vs, labeller = label_both)
# Using label_parsed, see ?plotmath for more options
mtcars$cyl2 <- factor(mtcars$cyl, labels = c("alpha", "beta", "sqrt(x, y)"))
k <- qplot(wt, mpg, data = mtcars)
k + facet_grid(. ~ cyl2)
k + facet_grid(. ~ cyl2, labeller = label_parsed)
# For label_bquote the label value is x.
p <- qplot(wt, mpg, data = mtcars)
p + facet_grid(. ~ vs, labeller = label_bquote(alpha ^ .(x)))

```

```

p + facet_grid(. ~ vs, labeller = label_bquote.(x) ^ .(x)))

# Margins can be specified by logically (all yes or all no) or by specific
# variables as (character) variable names
mg <- ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point()
mg + facet_grid(vs + am ~ gear)
mg + facet_grid(vs + am ~ gear, margins = TRUE)
mg + facet_grid(vs + am ~ gear, margins = "am")
# when margins are made over "vs", since the facets for "am" vary
# within the values of "vs", the marginal facet for "vs" is also
# a margin over "am".
mg + facet_grid(vs + am ~ gear, margins = "vs")
mg + facet_grid(vs + am ~ gear, margins = "gear")
mg + facet_grid(vs + am ~ gear, margins = c("gear", "am"))

```

---

facet\_null

*Facet specification: a single panel.*


---

## Description

Facet specification: a single panel.

## Usage

```
facet_null(shrink = TRUE)
```

## Arguments

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.
--------	----------------------------------------------------------------------------------------------------------------------------------------

## Examples

```

# facet_null is the default facetting specification if you
# don't override it with facet_grid or facet_wrap
ggplot(mtcars, aes(mpg, wt)) + geom_point()
qplot(mpg, wt, data = mtcars)

```



facet\_wrap

*Wrap a 1d ribbon of panels into 2d.***Description**

Wrap a 1d ribbon of panels into 2d.

**Usage**

```
facet_wrap(facets, nrow = NULL, ncol = NULL, scales = "fixed",
  shrink = TRUE, as.table = TRUE, drop = TRUE)
```

**Arguments**

nrow	number of rows
ncol	number of columns
facets	formula specifying variables to facet by
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.
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.
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.

**Examples**

```
d <- ggplot(diamonds, aes(carat, price, fill = ..density..)) +
  xlim(0, 2) + stat_binhex(na.rm = TRUE) + theme(aspect.ratio = 1)
d + facet_wrap(~ color)
d + facet_wrap(~ color, ncol = 1)
d + facet_wrap(~ color, ncol = 4)
d + facet_wrap(~ color, nrow = 1)
d + facet_wrap(~ color, nrow = 3)

# Using multiple variables continues to wrap the long ribbon of
# plots into 2d - the ribbon just gets longer
# d + facet_wrap(~ color + cut)

# To change plot order of facet wrap,
# change the order of variable levels with factor()
diamonds$color <- factor(diamonds$color, levels = c("G", "J", "D", "E", "I", "F", "H"))
# Repeat first example with new order
```

```

d <- ggplot(diamonds, aes(carat, price, fill = ..density..)) +
  xlim(0, 2) + stat_binhex(na.rm = TRUE) + theme(aspect.ratio = 1)
d + facet_wrap(~ color)

# You can choose to keep the scales constant across all panels
# or vary the x scale, the y scale or both:
p <- qplot(price, data = diamonds, geom = "histogram", binwidth = 1000)
p + facet_wrap(~ color)
p + facet_wrap(~ color, scales = "free_y")

p <- qplot(displ, hwy, data = mpg)
p + facet_wrap(~ cyl)
p + facet_wrap(~ cyl, scales = "free")

# Use as.table to control direction of horizontal facets, TRUE by default
p + facet_wrap(~ cyl, as.table = FALSE)

# Add data that does not contain all levels of the faceting variables
cyl6 <- subset(mpg, cyl == 6)
p + geom_point(data = cyl6, colour = "red", size = 1) +
  facet_wrap(~ cyl)
p + geom_point(data = transform(cyl6, cyl = 7), colour = "red") +
  facet_wrap(~ cyl)
p + geom_point(data = transform(cyl6, cyl = NULL), colour = "red") +
  facet_wrap(~ cyl)

```

---

fortify

*Fortify a model with data.*


---

## Description

Method to convert a generic R object into a data frame useful for plotting. Takes its name from the idea of fortifying the original data with model fit statistics, and vice versa.

## Usage

```
fortify(model, data, ...)
```

## Arguments

model	model or other R object to convert to data frame
data	original dataset, if needed
...	other arguments passed to methods

## See Also

[fortify.lm](#)

## Description

Fortify methods for objects produced by **multcomp**

## Usage

```
## S3 method for class 'glht'  
fortify(model, data, ...)  
  
## S3 method for class 'confint.glht'  
fortify(model, data, ...)  
  
## S3 method for class 'summary.glht'  
fortify(model, data, ...)  
  
## S3 method for class 'cld'  
fortify(model, data, ...)
```

## Arguments

model	an object of class <code>glht</code> , <code>confint.glht</code> , <code>summary.glht</code> or <code>cld</code>
data, ...	other arguments to the generic ignored in this method.

## Examples

```
if (require("multcomp")) {  
  amod <- aov(breaks ~ wool + tension, data = warpbreaks)  
  wht <- glht(amod, linfct = mcp(tension = "Tukey"))  
  
  fortify(wht)  
  ggplot(wht, aes(lhs, estimate)) + geom_point()  
  
  CI <- confint(wht)  
  fortify(CI)  
  ggplot(CI, aes(lhs, estimate, ymin = lwr, ymax = upr)) +  
    geom_pointrange()  
  
  fortify(summary(wht))  
  ggplot(mapping = aes(lhs, estimate)) +  
    geom_linerange(aes(ymin = lwr, ymax = upr), data = CI) +  
    geom_point(aes(size = p), data = summary(wht)) +  
    scale_size(trans = "reverse")  
  
  cld <- cld(wht)  
  fortify(cld)  
}
```

---

fortify.lm

---

*Supplement the data fitted to a linear model with model fit statistics.*


---

## Description

If you have missing values in your model data, you may need to refit the model with `na.action = na.exclude`.

## Usage

```
## S3 method for class 'lm'
fortify(model, data = model$model, ...)
```

## Arguments

<code>model</code>	linear model
<code>data</code>	data set, defaults to data used to fit model
<code>...</code>	not used by this method

## Value

The original data with extra columns:

<code>.hat</code>	Diagonal of the hat matrix
<code>.sigma</code>	Estimate of residual standard deviation when corresponding observation is dropped from model
<code>.cooks</code>	Cooks distance, <a href="#">cooks.distance</a>
<code>.fitted</code>	Fitted values of model
<code>.resid</code>	Residuals
<code>.stdresid</code>	Standardised residuals

## Examples

```
mod <- lm(mpg ~ wt, data = mtcars)
head(fortify(mod))
head(fortify(mod, mtcars))

plot(mod, which = 1)
qplot(.fitted, .resid, data = mod) +
  geom_hline(yintercept = 0) +
  geom_smooth(se = FALSE)
qplot(.fitted, .stdresid, data = mod) +
  geom_hline(yintercept = 0) +
  geom_smooth(se = FALSE)
qplot(.fitted, .stdresid, data = fortify(mod, mtcars),
  colour = factor(cyl))
qplot(mpg, .stdresid, data = fortify(mod, mtcars), colour = factor(cyl))
```

```

plot(mod, which = 2)
# qqplot(sample = .stdresid, data = mod, stat = "qq") + geom_abline()

plot(mod, which = 3)
qqplot(.fitted, sqrt(abs(.stdresid)), data = mod) + geom_smooth(se = FALSE)

plot(mod, which = 4)
qqplot(seq_along(.cooksds), .cooksds, data = mod, geom = "bar",
  stat="identity")

plot(mod, which = 5)
qqplot(.hat, .stdresid, data = mod) + geom_smooth(se = FALSE)
ggplot(mod, aes(.hat, .stdresid)) +
  geom_vline(size = 2, colour = "white", xintercept = 0) +
  geom_hline(size = 2, colour = "white", yintercept = 0) +
  geom_point() + geom_smooth(se = FALSE)

qqplot(.hat, .stdresid, data = mod, size = .cooksds) +
  geom_smooth(se = FALSE, size = 0.5)

plot(mod, which = 6)
ggplot(mod, aes(.hat, .cooksds)) +
  geom_vline(xintercept = 0, colour = NA) +
  geom_abline(slope = seq(0, 3, by = 0.5), colour = "white") +
  geom_smooth(se = FALSE) +
  geom_point()
qqplot(.hat, .cooksds, size = .cooksds / .hat, data = mod) + scale_size_area()

```

---

fortify.map

*Fortify method for map objects.*


---

## Description

This function turns a map into a data frame that can more easily be plotted with ggplot2.

## Usage

```
## S3 method for class 'map'
fortify(model, data, ...)
```

## Arguments

model	map object
data	not used by this method
...	not used by this method

## See Also

[map\\_data](#) and [borders](#)

## Examples

```
if (require("maps")) {
  ca <- map("county", "ca", plot = FALSE, fill = TRUE)
  head(fortify(ca))
  qplot(long, lat, data = ca, geom = "polygon", group = group)

  tx <- map("county", "texas", plot = FALSE, fill = TRUE)
  head(fortify(tx))
  qplot(long, lat, data = tx, geom = "polygon", group = group,
        colour = I("white"))
}
```

---

fortify.sp

*Fortify method for classes from the sp package.*


---

## Description

To figure out the correct variable name for region, inspect `as.data.frame(model)`.

## Usage

```
## S3 method for class 'SpatialPolygonsDataFrame'
fortify(model, data, region = NULL, ...)

## S3 method for class 'SpatialPolygons'
fortify(model, data, ...)

## S3 method for class 'Polygons'
fortify(model, data, ...)

## S3 method for class 'Polygon'
fortify(model, data, ...)

## S3 method for class 'SpatialLinesDataFrame'
fortify(model, data, ...)

## S3 method for class 'Lines'
fortify(model, data, ...)

## S3 method for class 'Line'
fortify(model, data, ...)
```

## Arguments

model	SpatialPolygonsDataFrame to convert into a dataframe.
data	not used by this method
region	name of variable used to split up regions
...	not used by this method

## Examples

```
if (require("maptools")) {
  sids <- system.file("shapes/sids.shp", package="maptools")
  nc1 <- readShapePoly(sids,
    proj4string = CRS("+proj=longlat +datum=NAD27"))
  nc1_df <- fortify(nc1)
}
```

---

geom\_abline

*Line specified by slope and intercept.*

---

## Description

The abline geom adds a line with specified slope and intercept to the plot.

## Usage

```
geom_abline(mapping = NULL, data = NULL, stat = "abline",
  position = "identity", show_guide = FALSE, ...)
```

## Arguments

show_guide	should a legend be drawn? (defaults to FALSE)
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Details

With its siblings `geom_hline` and `geom_vline`, it's useful for annotating plots. You can supply the parameters for `geom_abline`, `intercept` and `slope`, in two ways: either explicitly as fixed values, or in a data frame. If you specify the fixed values (`geom_abline(intercept=0, slope=1)`) then the line will be the same in all panels. If the `intercept` and `slope` are stored in the data, then they can vary from panel to panel. See the examples for more ideas.

## Aesthetics

`geom_abline` understands the following aesthetics (required aesthetics are in bold):

- alpha
- colour
- linetype
- size

**See Also**

[stat\\_smooth](#) to add lines derived from the data, [geom\\_hline](#) for horizontal lines, [geom\\_vline](#) for vertical lines [geom\\_segment](#)

**Examples**

```
p <- qplot(wt, mpg, data = mtcars)

# Fixed slopes and intercepts
p + geom_abline() # Can't see it - outside the range of the data
p + geom_abline(intercept = 20)

# Calculate slope and intercept of line of best fit
coef(lm(mpg ~ wt, data = mtcars))
p + geom_abline(intercept = 37, slope = -5)
p + geom_abline(intercept = 10, colour = "red", size = 2)

# See ?stat_smooth for fitting smooth models to data
p + stat_smooth(method="lm", se=FALSE)

# Slopes and intercepts as data
p <- ggplot(mtcars, aes(x = wt, y=mpg), . ~ cyl) + geom_point()
df <- data.frame(a=rnorm(10, 25), b=rnorm(10, 0))
p + geom_abline(aes(intercept=a, slope=b), data=df)

# Slopes and intercepts from linear model
library(plyr)
coefs <- ddply(mtcars, .(cyl), function(df) {
  m <- lm(mpg ~ wt, data=df)
  data.frame(a = coef(m)[1], b = coef(m)[2])
})
str(coefs)
p + geom_abline(data=coefs, aes(intercept=a, slope=b))

# It's actually a bit easier to do this with stat_smooth
p + geom_smooth(aes(group=cyl), method="lm")
p + geom_smooth(aes(group=cyl), method="lm", fullrange=TRUE)

# With coordinate transforms
p + geom_abline(intercept = 37, slope = -5) + coord_flip()
p + geom_abline(intercept = 37, slope = -5) + coord_polar()
```

---

geom\_area

Area plot.

---

**Description**

An area plot is the continuous analog of a stacked bar chart (see [geom\\_bar](#)), and can be used to show how composition of the whole varies over the range of x. Choosing the order in which different



components is stacked is very important, as it becomes increasingly hard to see the individual pattern as you move up the stack.

### Usage

```
geom_area(mapping = NULL, data = NULL, stat = "identity",
  position = "stack", na.rm = FALSE, ...)
```

### Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

### Details

An area plot is a special case of [geom\\_ribbon](#), where the minimum of the range is fixed to 0, and the position adjustment defaults to `position_stacked`.

### Examples

```
# see geom_ribbon
```

---

geom\_bar

*Bars, rectangles with bases on x-axis*

---

### Description

The bar geom is used to produce 1d area plots: bar charts for categorical x, and histograms for continuous y. `stat_bin` explains the details of these summaries in more detail. In particular, you can use the `weight` aesthetic to create weighted histograms and barcharts where the height of the bar no longer represent a count of observations, but a sum over some other variable. See the examples for a practical example.

### Usage

```
geom_bar(mapping = NULL, data = NULL, stat = "bin", position = "stack",
  ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Details

The heights of the bars commonly represent one of two things: either a count of cases in each group, or the values in a column of the data frame. By default, `geom_bar` uses `stat="bin"`. This makes the height of each bar equal to the number of cases in each group, and it is incompatible with mapping values to the y aesthetic. If you want the heights of the bars to represent values in the data, use `stat="identity"` and map a value to the y aesthetic.

By default, multiple x's occurring in the same place will be stacked a top one another by `position_stack`. If you want them to be dodged from side-to-side, see [position\\_dodge](#). Finally, [position\\_fill](#) shows relative proportions at each x by stacking the bars and then stretching or squashing to the same height.

Sometimes, bar charts are used not as a distributional summary, but instead of a dotplot. Generally, it's preferable to use a dotplot (see `geom_point`) as it has a better data-ink ratio. However, if you do want to create this type of plot, you can set y to the value you have calculated, and use `stat='identity'`

A bar chart maps the height of the bar to a variable, and so the base of the bar must always been shown to produce a valid visual comparison. Naomi Robbins has a nice [article on this topic](#). This is the reason it doesn't make sense to use a log-scaled y axis with a bar chart

## Aesthetics

`geom_bar` understands the following aesthetics (required aesthetics are in bold):

- **x**
- alpha
- colour
- fill
- linetype
- size
- weight

## See Also

[stat\\_bin](#) for more details of the binning algorithm, [position\\_dodge](#) for creating side-by-side barcharts, [position\\_stack](#) for more info on stacking,

**Examples**

```

# Generate data
c <- ggplot(mtcars, aes(factor(cyl)))

# By default, uses stat="bin", which gives the count in each category
c + geom_bar()
c + geom_bar(width=.5)
c + geom_bar() + coord_flip()
c + geom_bar(fill="white", colour="darkgreen")

# Use qplot
qplot(factor(cyl), data=mtcars, geom="bar")
qplot(factor(cyl), data=mtcars, geom="bar", fill=factor(cyl))

# When the data contains y values in a column, use stat="identity"
library(plyr)
# Calculate the mean mpg for each level of cyl
mm <- ddply(mtcars, "cyl", summarise, mmpg = mean(mpg))
ggplot(mm, aes(x = factor(cyl), y = mmpg)) + geom_bar(stat = "identity")

# Stacked bar charts
qplot(factor(cyl), data=mtcars, geom="bar", fill=factor(vs))
qplot(factor(cyl), data=mtcars, geom="bar", fill=factor(gear))

# Stacked bar charts are easy in ggplot2, but not effective visually,
# particularly when there are many different things being stacked
ggplot(diamonds, aes(clarity, fill=cut)) + geom_bar()
ggplot(diamonds, aes(color, fill=cut)) + geom_bar() + coord_flip()

# Faceting is a good alternative:
ggplot(diamonds, aes(clarity)) + geom_bar() +
  facet_wrap(~ cut)
# If the x axis is ordered, using a line instead of bars is another
# possibility:
ggplot(diamonds, aes(clarity)) +
  geom_freqpoly(aes(group = cut, colour = cut))

# Dodged bar charts
ggplot(diamonds, aes(clarity, fill=cut)) + geom_bar(position="dodge")
# compare with
ggplot(diamonds, aes(cut, fill=cut)) + geom_bar() +
  facet_grid(. ~ clarity)

# But again, probably better to use frequency polygons instead:
ggplot(diamonds, aes(clarity, colour=cut)) +
  geom_freqpoly(aes(group = cut))

# Often we don't want the height of the bar to represent the
# count of observations, but the sum of some other variable.
# For example, the following plot shows the number of diamonds
# of each colour

```

```

qplot(color, data=diamonds, geom="bar")
# If, however, we want to see the total number of carats in each colour
# we need to weight by the carat variable
qplot(color, data=diamonds, geom="bar", weight=carat, ylab="carat")

# A bar chart used to display means
meanprice <- tapply(diamonds$price, diamonds$cut, mean)
cut <- factor(levels(diamonds$cut), levels = levels(diamonds$cut))
qplot(cut, meanprice)
qplot(cut, meanprice, geom="bar", stat="identity")
qplot(cut, meanprice, geom="bar", stat="identity", fill = I("grey50"))

# Another stacked bar chart example
k <- ggplot(mpg, aes(manufacturer, fill=class))
k + geom_bar()
# Use scales to change aesthetics defaults
k + geom_bar() + scale_fill_brewer()
k + geom_bar() + scale_fill_grey()

# To change plot order of class variable
# use factor() to change order of levels
mpg$class <- factor(mpg$class, levels = c("midsize", "minivan",
"suv", "compact", "2seater", "subcompact", "pickup"))
m <- ggplot(mpg, aes(manufacturer, fill=class))
m + geom_bar()

```

---

geom\_bin2d

Add heatmap of 2d bin counts.

---

## Description

Add heatmap of 2d bin counts.

## Usage

```

geom_bin2d(mapping = NULL, data = NULL, stat = "bin2d",
  position = "identity", ...)

```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

geom\_bin2d understands the following aesthetics (required aesthetics are in bold):

- **xmax**
- **xmin**
- **ymax**
- **ymin**
- alpha
- colour
- fill
- linetype
- size
- weight

## Examples

```
d <- ggplot(diamonds, aes(x = x, y = y)) + xlim(4,10) + ylim(4,10)
d + geom_bin2d()
d + geom_bin2d(binwidth = c(0.1, 0.1))

# See ?stat_bin2d for more examples
```

---

geom_blank	<i>Blank, draws nothing.</i>
------------	------------------------------

---

## Description

The blank geom draws nothing, but can be a useful way of ensuring common scales between different plots.

## Usage

```
geom_blank(mapping = NULL, data = NULL, stat = "identity",
           position = "identity", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Examples

```
qplot(length, rating, data = movies, geom = "blank")
# Nothing to see here!

# Take the following scatter plot
a <- ggplot(mtcars, aes(x = wt, y = mpg), . ~ cyl) + geom_point()
# Add to that some lines with geom_abline()
df <- data.frame(a = rnorm(10, 25), b = rnorm(10, 0))
a + geom_abline(aes(intercept = a, slope = b), data = df)
# Suppose you then wanted to remove the geom_point layer
# If you just remove geom_point, you will get an error
b <- ggplot(mtcars, aes(x = wt, y = mpg))
## Not run: b + geom_abline(aes(intercept = a, slope = b), data = df)
# Switching to geom_blank() gets the desired plot
c <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_blank()
c + geom_abline(aes(intercept = a, slope = b), data = df)
```

---

geom\_boxplot

*Box and whiskers plot.*

---

## Description

The upper and lower "hinges" correspond to the first and third quartiles (the 25th and 75th percentiles). This differs slightly from the method used by the `boxplot` function, and may be apparent with small samples. See [boxplot.stats](#) for more information on how hinge positions are calculated for `boxplot`.

## Usage

```
geom_boxplot(mapping = NULL, data = NULL, stat = "boxplot",
  position = "dodge", outlier.colour = NULL, outlier.shape = NULL,
  outlier.size = NULL, notch = FALSE, notchwidth = 0.5,
  varwidth = FALSE, ...)
```

## Arguments

<code>outlier.colour</code>	colour for outlying points. Uses the default from <code>geom_point()</code> .
<code>outlier.shape</code>	shape of outlying points. Uses the default from <code>geom_point()</code> .
<code>outlier.size</code>	size of outlying points. Uses the default from <code>geom_point()</code> .
<code>notch</code>	if <code>FALSE</code> (default) make a standard box plot. If <code>TRUE</code> , make a notched box plot. Notches are used to compare groups; if the notches of two boxes do not overlap, this is strong evidence that the medians differ.
<code>notchwidth</code>	for a notched box plot, width of the notch relative to the body (default 0.5)
<code>varwidth</code>	if <code>FALSE</code> (default) make a standard box plot. If <code>TRUE</code> , boxes are drawn with widths proportional to the square-roots of the number of observations in the groups (possibly weighted, using the <code>weight</code> aesthetic).

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

### Details

The upper whisker extends from the hinge to the highest value that is within 1.5 \* IQR of the hinge, where IQR is the inter-quartile range, or distance between the first and third quartiles. The lower whisker extends from the hinge to the lowest value within 1.5 \* IQR of the hinge. Data beyond the end of the whiskers are outliers and plotted as points (as specified by Tukey).

In a notched box plot, the notches extend  $1.58 * \text{IQR} / \sqrt{n}$ . This gives a roughly 95% See McGill et al. (1978) for more details.

### Aesthetics

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

- **lower**
- **middle**
- **upper**
- **x**
- **ymax**
- **ymin**
- alpha
- colour
- fill
- linetype
- shape
- size
- weight

### References

McGill, R., Tukey, J. W. and Larsen, W. A. (1978) Variations of box plots. The American Statistician 32, 12-16.

### See Also

[stat\\_quantile](#) to view quantiles conditioned on a continuous variable, [geom\\_jitter](#) for another way to look at conditional distributions"

**Examples**

```

p <- ggplot(mtcars, aes(factor(cyl), mpg))

p + geom_boxplot()
qplot(factor(cyl), mpg, data = mtcars, geom = "boxplot")

p + geom_boxplot() + geom_jitter()
p + geom_boxplot() + coord_flip()
qplot(factor(cyl), mpg, data = mtcars, geom = "boxplot") +
  coord_flip()

p + geom_boxplot(notch = TRUE)
p + geom_boxplot(notch = TRUE, notchwidth = .3)

p + geom_boxplot(outlier.colour = "green", outlier.size = 3)

# Add aesthetic mappings
# Note that boxplots are automatically dodged when any aesthetic is
# a factor
p + geom_boxplot(aes(fill = cyl))
p + geom_boxplot(aes(fill = factor(cyl)))
p + geom_boxplot(aes(fill = factor(vs)))
p + geom_boxplot(aes(fill = factor(am)))

# Set aesthetics to fixed value
p + geom_boxplot(fill = "grey80", colour = "#3366FF")
qplot(factor(cyl), mpg, data = mtcars, geom = "boxplot",
  colour = I("#3366FF"))

# Scales vs. coordinate transforms -----
# Scale transformations occur before the boxplot statistics are computed.
# Coordinate transformations occur afterwards. Observe the effect on the
# number of outliers.
library(plyr) # to access round_any
m <- ggplot(movies, aes(y = votes, x = rating,
  group = round_any(rating, 0.5)))
m + geom_boxplot()
m + geom_boxplot() + scale_y_log10()
m + geom_boxplot() + coord_trans(y = "log10")
m + geom_boxplot() + scale_y_log10() + coord_trans(y = "log10")

# Boxplots with continuous x:
# Use the group aesthetic to group observations in boxplots
qplot(year, budget, data = movies, geom = "boxplot")
qplot(year, budget, data = movies, geom = "boxplot",
  group = round_any(year, 10, floor))

# Using precomputed statistics
# generate sample data
abc <- adply(matrix(rnorm(100), ncol = 5), 2, quantile, c(0, .25, .5, .75, 1))
b <- ggplot(abc, aes(x = X1, ymin = `0%`, lower = `25%`,

```



```

    middle = `50%`, upper = `75%`, ymax = `100%`))
b + geom_boxplot(stat = "identity")
b + geom_boxplot(stat = "identity") + coord_flip()
b + geom_boxplot(aes(fill = X1), stat = "identity")

# Using varwidth
p + geom_boxplot(varwidth = TRUE)
qplot(factor(cyl), mpg, data = mtcars, geom = "boxplot", varwidth = TRUE)

# Update the defaults for the outliers by changing the defaults for geom_point

p <- ggplot(mtcars, aes(factor(cyl), mpg))
p + geom_boxplot()

update_geom_defaults("point", list(shape = 1, colour = "red", size = 5))
p + geom_boxplot()

```

geom\_contour

*Display contours of a 3d surface in 2d.***Description**

Display contours of a 3d surface in 2d.

**Usage**

```

geom_contour(mapping = NULL, data = NULL, stat = "contour",
  position = "identity", lineend = "butt", linejoin = "round",
  linemitre = 1, na.rm = FALSE, ...)

```

**Arguments**

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.
lineend	Line end style (round, butt, square)
linejoin	Line join style (round, mitre, bevel)
linemitre	Line mitre limit (number greater than 1)

## Aesthetics

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

- **x**
- **y**
- alpha
- colour
- linetype
- size
- weight

## See Also

[geom\\_density2d](#): 2d density contours

## Examples

```
# See stat_contour for examples
```

---

geom\_crossbar

*Hollow bar with middle indicated by horizontal line.*

---

## Description

Hollow bar with middle indicated by horizontal line.

## Usage

```
geom_crossbar(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", fatten = 2, ...)
```

## Arguments

fatten	a multiplicate factor to fatten middle bar by
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Aesthetics**

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

- **x**
- **y**
- **ymax**
- **ymin**
- alpha
- colour
- fill
- linetype
- size

**See Also**

[geom\\_errorbar](#) for error bars, [geom\\_pointrange](#) and [geom\\_linerange](#) for other ways of showing mean + error, [stat\\_summary](#) to compute errors from the data, [geom\\_smooth](#) for the continuous analog.

**Examples**

```
# See geom_linerange for examples
```

---

geom_density	<i>Display a smooth density estimate.</i>
--------------	-------------------------------------------

---

**Description**

A smooth density estimate calculated by [stat\\_density](#).

**Usage**

```
geom_density(mapping = NULL, data = NULL, stat = "density",
  position = "identity", na.rm = FALSE, ...)
```

**Arguments**

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Aesthetics**

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

- **x**
- **y**
- alpha
- colour
- fill
- linetype
- size
- weight

**See Also**

[geom\\_histogram](#) for the histogram and [stat\\_density](#) for examples.

**Examples**

```
# See stat_density for examples
```

---

geom_density2d	<i>Contours from a 2d density estimate.</i>
----------------	---------------------------------------------

---

**Description**

Perform a 2D kernel density estimation using kde2d and display the results with contours.

**Usage**

```
geom_density2d(mapping = NULL, data = NULL, stat = "density2d",
  position = "identity", lineend = "butt", linejoin = "round",
  linemitre = 1, na.rm = FALSE, ...)
```

**Arguments**

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

lineend	Line end style (round, butt, square)
linejoin	Line join style (round, mitre, bevel)
linemitre	Line mitre limit (number greater than 1)

## Details

This can be useful for dealing with overplotting.

## Aesthetics

geom\_density2d understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour
- linetype
- size

## See Also

[geom\\_contour](#) for contour drawing geom, [stat\\_sum](#) for another way of dealing with overplotting

## Examples

```
# See stat_density2d for examples
```

---

geom_dotplot	<i>Dot plot</i>
--------------	-----------------

---

## Description

In a dot plot, the width of a dot corresponds to the bin width (or maximum width, depending on the binning algorithm), and dots are stacked, with each dot representing one observation.

## Usage

```
geom_dotplot(mapping = NULL, data = NULL, stat = "bindot",
  position = "identity", na.rm = FALSE, binwidth = NULL, binaxis = "x",
  method = "dotdensity", binpositions = "bygroup", stackdir = "up",
  stackratio = 1, dotsize = 1, stackgroups = FALSE, ...)
```

## Arguments

<code>binaxis</code>	which axis to bin along "x" (default) or "y"
<code>method</code>	"dotdensity" (default) for dot-density binning, or "histodot" for fixed bin widths (like <code>stat_bin</code> )
<code>binwidth</code>	When method is "dotdensity", this specifies maximum bin width. When method is "histodot", this specifies bin width. Defaults to 1/30 of the range of the data
<code>binpositions</code>	When method is "dotdensity", "bygroup" (default) determines positions of the bins for each group separately. "all" determines positions of the bins with all the data taken together; this is used for aligning dot stacks across multiple groups.
<code>stackdir</code>	which direction to stack the dots. "up" (default), "down", "center", "centerw-hole" (centered, but with dots aligned)
<code>stackratio</code>	how close to stack the dots. Default is 1, where dots just touch. Use smaller values for closer, overlapping dots.
<code>dotsize</code>	The diameter of the dots relative to binwidth, default 1.
<code>stackgroups</code>	should dots be stacked across groups? This has the effect that <code>position = "stack"</code> should have, but can't (because this geom has some odd properties).
<code>mapping</code>	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
<code>data</code>	A layer specific dataset - only needed if you want to override the plot defaults.
<code>stat</code>	The statistical transformation to use on the data for this layer.
<code>position</code>	The position adjustment to use for overlapping points on this layer
<code>na.rm</code>	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
<code>...</code>	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Details

With dot-density binning, the bin positions are determined by the data and `binwidth`, which is the maximum width of each bin. See Wilkinson (1999) for details on the dot-density binning algorithm.

With `histodot` binning, the bins have fixed positions and fixed widths, much like a histogram.

When binning along the x axis and stacking along the y axis, the numbers on y axis are not meaningful, due to technical limitations of `ggplot2`. You can hide the y axis, as in one of the examples, or manually scale it to match the number of dots.

## Aesthetics

`geom_dotplot` understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour
- fill

## References

Wilkinson, L. (1999) Dot plots. *The American Statistician*, 53(3), 276-281.

## Examples

```
ggplot(mtcars, aes(x = mpg)) + geom_dotplot()
ggplot(mtcars, aes(x = mpg)) + geom_dotplot(binwidth = 1.5)

# Use fixed-width bins
ggplot(mtcars, aes(x = mpg)) +
  geom_dotplot(method="histodot", binwidth = 1.5)

# Some other stacking methods
ggplot(mtcars, aes(x = mpg)) +
  geom_dotplot(binwidth = 1.5, stackdir = "center")
ggplot(mtcars, aes(x = mpg)) +
  geom_dotplot(binwidth = 1.5, stackdir = "centerwhole")

# y axis isn't really meaningful, so hide it
ggplot(mtcars, aes(x = mpg)) + geom_dotplot(binwidth = 1.5) +
  scale_y_continuous(name = "", breaks = NULL)

# Overlap dots vertically
ggplot(mtcars, aes(x = mpg)) + geom_dotplot(binwidth = 1.5, stackratio = .7)

# Expand dot diameter
ggplot(mtcars, aes(x = mpg)) + geom_dotplot(binwidth = 1.5, dotsize = 1.25)

# Examples with stacking along y axis instead of x
ggplot(mtcars, aes(x = 1, y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "center")

ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "center")

ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "centerwhole")

ggplot(mtcars, aes(x = factor(vs), fill = factor(cyl), y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "center", position = "dodge")

# binpositions="all" ensures that the bins are aligned between groups
ggplot(mtcars, aes(x = factor(am), y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "center", binpositions="all")

# Stacking multiple groups, with different fill
ggplot(mtcars, aes(x = mpg, fill = factor(cyl))) +
  geom_dotplot(stackgroups = TRUE, binwidth = 1, binpositions = "all")

ggplot(mtcars, aes(x = mpg, fill = factor(cyl))) +
  geom_dotplot(stackgroups = TRUE, binwidth = 1, method = "histodot")
```

```
ggplot(mtcars, aes(x = 1, y = mpg, fill = factor(cyl))) +
  geom_dotplot(binaxis = "y", stackgroups = TRUE, binwidth = 1, method = "histodot")

# Use qplot instead
qplot(mpg, data = mtcars, geom = "dotplot")
```

---

geom\_errorbar

Error bars.

---

## Description

Error bars.

## Usage

```
geom_errorbar(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

- **x**
- **y**max
- **y**min
- alpha
- colour
- linetype
- size
- width



**See Also**

[geom\\_pointrange](#): range indicated by straight line, with point in the middle; [geom\\_linerange](#): range indicated by straight line; [geom\\_crossbar](#): hollow bar with middle indicated by horizontal line; [stat\\_summary](#): examples of these guys in use, [geom\\_smooth](#) for continuous analog

**Examples**

```
# Create a simple example dataset
df <- data.frame(
  trt = factor(c(1, 1, 2, 2)),
  resp = c(1, 5, 3, 4),
  group = factor(c(1, 2, 1, 2)),
  se = c(0.1, 0.3, 0.3, 0.2)
)
df2 <- df[c(1,3),]

# Define the top and bottom of the errorbars
limits <- aes(ymax = resp + se, ymin=resp - se)

p <- ggplot(df, aes(fill=group, y=resp, x=trt))
p + geom_bar(position="dodge", stat="identity")

# Because the bars and errorbars have different widths
# we need to specify how wide the objects we are dodging are
dodge <- position_dodge(width=0.9)
p + geom_bar(position=dodge) + geom_errorbar(limits, position=dodge, width=0.25)

p <- ggplot(df2, aes(fill=group, y=resp, x=trt))
p + geom_bar(position=dodge)
p + geom_bar(position=dodge) + geom_errorbar(limits, position=dodge, width=0.25)

p <- ggplot(df, aes(colour=group, y=resp, x=trt))
p + geom_point() + geom_errorbar(limits, width=0.2)
p + geom_pointrange(limits)
p + geom_crossbar(limits, width=0.2)

# If we want to draw lines, we need to manually set the
# groups which define the lines - here the groups in the
# original dataframe
p + geom_line(aes(group=group)) + geom_errorbar(limits, width=0.2)
```

---

geom\_errorbarh

*Horizontal error bars*


---

**Description**

Horizontal error bars

**Usage**

```
geom_errorbarh(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ...)
```

**Arguments**

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Aesthetics**

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

- **x**
- **xmax**
- **xmin**
- **y**
- alpha
- colour
- height
- linetype
- size

**See Also**

[geom\\_errorbar](#): vertical error bars

**Examples**

```
df <- data.frame(
  trt = factor(c(1, 1, 2, 2)),
  resp = c(1, 5, 3, 4),
  group = factor(c(1, 2, 1, 2)),
  se = c(0.1, 0.3, 0.3, 0.2)
)

# Define the top and bottom of the errorbars

p <- ggplot(df, aes(resp, trt, colour = group))
p + geom_point() +
  geom_errorbarh(aes(xmax = resp + se, xmin = resp - se))
p + geom_point() +
  geom_errorbarh(aes(xmax = resp + se, xmin = resp - se, height = .2))
```

---

geom_freqpoly	<i>Frequency polygon.</i>
---------------	---------------------------

---

## Description

Frequency polygon.

## Usage

```
geom_freqpoly(mapping = NULL, data = NULL, stat = "bin",  
              position = "identity", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

- alpha
- colour
- linetype
- size

## See Also

[geom\\_histogram](#): histograms

## Examples

```
qplot(carat, data = diamonds, geom = "freqpoly")  
qplot(carat, data = diamonds, geom = "freqpoly", binwidth = 0.1)  
qplot(carat, data = diamonds, geom = "freqpoly", binwidth = 0.01)  
  
qplot(price, data = diamonds, geom = "freqpoly", binwidth = 1000)  
qplot(price, data = diamonds, geom = "freqpoly", binwidth = 1000,  
      colour = color)  
qplot(price, ..density.., data = diamonds, geom = "freqpoly",  
      binwidth = 1000, colour = color)
```

---

`geom_hex`*Hexagon binning.*

---

## Description

Hexagon binning.

## Usage

```
geom_hex(mapping = NULL, data = NULL, stat = "binhex",  
         position = "identity", ...)
```

## Arguments

<code>mapping</code>	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
<code>data</code>	A layer specific dataset - only needed if you want to override the plot defaults.
<code>stat</code>	The statistical transformation to use on the data for this layer.
<code>position</code>	The position adjustment to use for overlapping points on this layer
<code>...</code>	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

`geom_hex` understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour
- fill
- size

## Examples

```
# See ?stat_binhex for examples
```

---

geom_histogram	<i>Histogram</i>
----------------	------------------

---

## Description

geom\_histogram is an alias for [geom\\_bar](#) plus [stat\\_bin](#) so you will need to look at the documentation for those objects to get more information about the parameters.

## Usage

```
geom_histogram(mapping = NULL, data = NULL, stat = "bin",  
  position = "stack", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Details

By default, stat\_bin uses 30 bins - this is not a good default, but the idea is to get you experimenting with different binwidths. You may need to look at a few to uncover the full story behind your data.

## Aesthetics

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

- **x**
- alpha
- colour
- fill
- linetype
- size
- weight

## Examples

```

set.seed(5689)
movies <- movies[sample(nrow(movies), 1000), ]
# Simple examples
qplot(rating, data=movies, geom="histogram")
qplot(rating, data=movies, weight=votes, geom="histogram")
qplot(rating, data=movies, weight=votes, geom="histogram", binwidth=1)
qplot(rating, data=movies, weight=votes, geom="histogram", binwidth=0.1)

# More complex
m <- ggplot(movies, aes(x=rating))
m + geom_histogram()
m + geom_histogram(aes(y = ..density..)) + geom_density()

m + geom_histogram(binwidth = 1)
m + geom_histogram(binwidth = 0.5)
m + geom_histogram(binwidth = 0.1)

# Add aesthetic mappings
m + geom_histogram(aes(weight = votes))
m + geom_histogram(aes(y = ..count..))
m + geom_histogram(aes(fill = ..count..))

# Change scales
m + geom_histogram(aes(fill = ..count..)) +
  scale_fill_gradient("Count", low = "green", high = "red")

# Often we don't want the height of the bar to represent the
# count of observations, but the sum of some other variable.
# For example, the following plot shows the number of movies
# in each rating.
qplot(rating, data=movies, geom="bar", binwidth = 0.1)
# If, however, we want to see the number of votes cast in each
# category, we need to weight by the votes variable
qplot(rating, data=movies, geom="bar", binwidth = 0.1,
  weight=votes, ylab = "votes")

m <- ggplot(movies, aes(x = votes))
# For transformed scales, binwidth applies to the transformed data.
# The bins have constant width on the transformed scale.
m + geom_histogram() + scale_x_log10()
m + geom_histogram(binwidth = 1) + scale_x_log10()
m + geom_histogram() + scale_x_sqrt()
m + geom_histogram(binwidth = 10) + scale_x_sqrt()

# For transformed coordinate systems, the binwidth applies to the
# raw data. The bins have constant width on the original scale.

# Using log scales does not work here, because the first
# bar is anchored at zero, and so when transformed becomes negative
# infinity. This is not a problem when transforming the scales, because

```

```

# no observations have 0 ratings.
m + geom_histogram(origin = 0) + coord_trans(x = "log10")
# Use origin = 0, to make sure we don't take sqrt of negative values
m + geom_histogram(origin = 0) + coord_trans(x = "sqrt")
m + geom_histogram(origin = 0, binwidth = 1000) + coord_trans(x = "sqrt")

# You can also transform the y axis. Remember that the base of the bars
# has value 0, so log transformations are not appropriate
m <- ggplot(movies, aes(x = rating))
m + geom_histogram(binwidth = 0.5) + scale_y_sqrt()
m + geom_histogram(binwidth = 0.5) + scale_y_reverse()

# Set aesthetics to fixed value
m + geom_histogram(colour = "darkgreen", fill = "white", binwidth = 0.5)

# Use facets
m <- m + geom_histogram(binwidth = 0.5)
m + facet_grid(Action ~ Comedy)

# Often more useful to use density on the y axis when facetting
m <- m + aes(y = ..density..)
m + facet_grid(Action ~ Comedy)
m + facet_wrap(~ mpaa)

# Multiple histograms on the same graph
# see ?position, ?position_fill, etc for more details.
set.seed(6298)
diamonds_small <- diamonds[sample(nrow(diamonds), 1000), ]
ggplot(diamonds_small, aes(x=price)) + geom_bar()
hist_cut <- ggplot(diamonds_small, aes(x=price, fill=cut))
hist_cut + geom_bar() # defaults to stacking
hist_cut + geom_bar(position="fill")
hist_cut + geom_bar(position="dodge")

# This is easy in ggplot2, but not visually effective. It's better
# to use a frequency polygon or density plot. Like this:
ggplot(diamonds_small, aes(price, ..density.., colour = cut)) +
  geom_freqpoly(binwidth = 1000)
# Or this:
ggplot(diamonds_small, aes(price, colour = cut)) +
  geom_density()
# Or if you want to be fancy, maybe even this:
ggplot(diamonds_small, aes(price, fill = cut)) +
  geom_density(alpha = 0.2)
# Which looks better when the distributions are more distinct
ggplot(diamonds_small, aes(depth, fill = cut)) +
  geom_density(alpha = 0.2) + xlim(55, 70)

rm(movies)

```

## Description

This geom allows you to annotate the plot with horizontal lines (see [geom\\_vline](#) and [geom\\_abline](#) for other types of lines).

## Usage

```
geom_hline(mapping = NULL, data = NULL, stat = "hline",
  position = "identity", show_guide = FALSE, ...)
```

## Arguments

show_guide	should a legend be drawn? (defaults to FALSE)
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Details

There are two ways to use it. You can either specify the intercept of the line in the call to the geom, in which case the line will be in the same position in every panel. Alternatively, you can supply a different intercept for each panel using a data.frame. See the examples for the differences

## Aesthetics

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

- alpha
- colour
- linetype
- size

## See Also

[geom\\_vline](#) for vertical lines, [geom\\_abline](#) for lines defined by a slope and intercept, [geom\\_segment](#) for a more general approach

## Examples

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

p + geom_hline(aes(yintercept=mpg))
p + geom_hline(yintercept=20)
p + geom_hline(yintercept=seq(10, 30, by=5))
```



```
# With coordinate transforms
p + geom_hline(aes(yintercept=mpg)) + coord_equal()
p + geom_hline(aes(yintercept=mpg)) + coord_flip()
p + geom_hline(aes(yintercept=mpg)) + coord_polar()

# To display different lines in different facets, you need to
# create a data frame.
p <- qplot(mpg, wt, data=mtcars, facets = vs ~ am)

hline.data <- data.frame(z = 1:4, vs = c(0,0,1,1), am = c(0,1,0,1))
p + geom_hline(aes(yintercept = z), hline.data)
```

geom\_jitter

*Points, jittered to reduce overplotting.***Description**

The jitter geom is a convenient default for `geom_point` with `position = 'jitter'`. See [position\\_jitter](#) to see how to adjust amount of jittering.

**Usage**

```
geom_jitter(mapping = NULL, data = NULL, stat = "identity",
  position = "jitter", na.rm = FALSE, ...)
```

**Arguments**

<code>mapping</code>	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
<code>data</code>	A layer specific dataset - only needed if you want to override the plot defaults.
<code>stat</code>	The statistical transformation to use on the data for this layer.
<code>position</code>	The position adjustment to use for overlapping points on this layer
<code>na.rm</code>	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
<code>...</code>	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Aesthetics**

`geom_jitter` understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour

- fill
- shape
- size

### See Also

[geom\\_point](#) for regular, unjittered points, [geom\\_boxplot](#) for another way of looking at the conditional distribution of a variable, [position\\_jitter](#) for examples of using jittering with other geoms

### Examples

```
p <- ggplot(mpg, aes(displ, hwy))
p + geom_point()
p + geom_point(position = "jitter")

# Add aesthetic mappings
p + geom_jitter(aes(colour = cyl))

# Vary parameters
p + geom_jitter(position = position_jitter(width = .5))
p + geom_jitter(position = position_jitter(height = .5))

# Use qplot instead
qplot(displ, hwy, data = mpg, geom = "jitter")
qplot(class, hwy, data = mpg, geom = "jitter")
qplot(class, hwy, data = mpg, geom = c("boxplot", "jitter"))
qplot(class, hwy, data = mpg, geom = c("jitter", "boxplot"))
```

---

geom\_line

*Connect observations, ordered by x value.*

---

### Description

Connect observations, ordered by x value.

### Usage

```
geom_line(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ...)
```

### Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Aesthetics**

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

- **x**
- **y**
- alpha
- colour
- linetype
- size

**See Also**

[geom\\_path](#): connect observations in data order, [geom\\_segment](#): draw line segments, [geom\\_ribbon](#): fill between line and x-axis

**Examples**

```
# Summarise number of movie ratings by year of movie
mry <- do.call(rbind, by(movies, round(movies$rating), function(df) {
  nums <- tapply(df$length, df$year, length)
  data.frame(rating=round(df$rating[1]), year = as.numeric(names(nums)), number=as.vector(nums))
}))

p <- ggplot(mry, aes(x=year, y=number, group=rating))
p + geom_line()

# Add aesthetic mappings
p + geom_line(aes(size = rating))
p + geom_line(aes(colour = rating))

# Change scale
p + geom_line(aes(colour = rating)) + scale_colour_gradient(low="red")
p + geom_line(aes(size = rating)) + scale_size(range = c(0.1, 3))

# Set aesthetics to fixed value
p + geom_line(colour = "red", size = 1)

# Use qplot instead
qplot(year, number, data=mry, group=rating, geom="line")

# Using a time series
qplot(date, pop, data=economics, geom="line")
qplot(date, pop, data=economics, geom="line", log="y")
qplot(date, pop, data=subset(economics, date > as.Date("2006-1-1")), geom="line")
qplot(date, pop, data=economics, size=unemploy/pop, geom="line")

# Use the arrow parameter to add an arrow to the line
# See ?grid::arrow for more details
c <- ggplot(economics, aes(x = date, y = pop))
# Arrow defaults to "last"
```

```
library(grid)
c + geom_line(arrow = arrow())
c + geom_line(arrow = arrow(angle = 15, ends = "both", type = "closed"))

# See scale_date for examples of plotting multiple times series on
# a single graph

# A simple pcpx example

y2005 <- runif(300, 20, 120)
y2010 <- y2005 * runif(300, -1.05, 1.5)
group <- rep(LETTERS[1:3], each = 100)

df <- data.frame(id = seq_along(group), group, y2005, y2010)
library(reshape2) # for melt
dfm <- melt(df, id.var = c("id", "group"))
ggplot(dfm, aes(variable, value, group = id, colour = group)) +
  geom_path(alpha = 0.5)
```

---

geom_linerange	<i>An interval represented by a vertical line.</i>
----------------	----------------------------------------------------

---

## Description

An interval represented by a vertical line.

## Usage

```
geom_linerange(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

- **x**
- **ymax**

- `ymin`
- `alpha`
- `colour`
- `linetype`
- `size`

### See Also

[geom\\_errorbar](#): error bars; [geom\\_pointrange](#): range indicated by straight line, with point in the middle; [geom\\_crossbar](#): hollow bar with middle indicated by horizontal line; [stat\\_summary](#): examples of these guys in use; [geom\\_smooth](#): for continuous analog

### Examples

```
# Generate data: means and standard errors of means for prices
# for each type of cut
dmod <- lm(price ~ cut, data=diamonds)
cuts <- data.frame(cut = unique(diamonds$cut),
  predict(dmod, data.frame(cut = unique(diamonds$cut)), se=TRUE)[c("fit","se.fit")])

qplot(cut, fit, data=cuts)
# With a bar chart, we are comparing lengths, so the y-axis is
# automatically extended to include 0
qplot(cut, fit, data=cuts, geom="bar")

# Display estimates and standard errors in various ways
se <- ggplot(cuts, aes(cut, fit,
  ymin = fit - se.fit, ymax=fit + se.fit, colour = cut))
se + geom_linerange()
se + geom_pointrange()
se + geom_errorbar(width = 0.5)
se + geom_crossbar(width = 0.5)

# Use coord_flip to flip the x and y axes
se + geom_linerange() + coord_flip()
```

---

geom\_map

*Polygons from a reference map.*

---

### Description

Does not affect position scales.

### Usage

```
geom_map(mapping = NULL, data = NULL, map, stat = "identity", ...)
```

## Arguments

map	Data frame that contains the map coordinates. This will typically be created using <code>fortify</code> on a spatial object. It must contain columns x or long, y or lat, and region or id.
mapping	The aesthetic mapping, usually constructed with <code>aes</code> or <code>aes_string</code> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
...	other arguments passed on to <code>layer</code> . This can include aesthetics whose values you want to set, not map. See <code>layer</code> for more details.

## Aesthetics

`geom_map` understands the following aesthetics (required aesthetics are in bold):

- **map\_id**
- alpha
- colour
- fill
- linetype
- size

## Examples

```
# When using geom_polygon, you will typically need two data frames:
# one contains the coordinates of each polygon (positions), and the
# other the values associated with each polygon (values). An id
# variable links the two together
```

```
ids <- factor(c("1.1", "2.1", "1.2", "2.2", "1.3", "2.3"))
```

```
values <- data.frame(
  id = ids,
  value = c(3, 3.1, 3.1, 3.2, 3.15, 3.5)
)
```

```
positions <- data.frame(
  id = rep(ids, each = 4),
  x = c(2, 1, 1.1, 2.2, 1, 0, 0.3, 1.1, 2.2, 1.1, 1.2, 2.5, 1.1, 0.3,
        0.5, 1.2, 2.5, 1.2, 1.3, 2.7, 1.2, 0.5, 0.6, 1.3),
  y = c(-0.5, 0, 1, 0.5, 0, 0.5, 1.5, 1, 0.5, 1, 2.1, 1.7, 1, 1.5,
        2.2, 2.1, 1.7, 2.1, 3.2, 2.8, 2.1, 2.2, 3.3, 3.2)
)
```

```
ggplot(values) + geom_map(aes(map_id = id), map = positions) +
  expand_limits(positions)
ggplot(values, aes(fill = value)) +
  geom_map(aes(map_id = id), map = positions) +
```

```

    expand_limits(positions)
ggplot(values, aes(fill = value)) +
  geom_map(aes(map_id = id), map = positions) +
  expand_limits(positions) + ylim(0, 3)

# Better example
crimes <- data.frame(state = tolower(rownames(USArrests)), USArrests)
library(reshape2) # for melt
crimesm <- melt(crimes, id = 1)
if (require(maps)) {
  states_map <- map_data("state")
  ggplot(crimes, aes(map_id = state)) +
    geom_map(aes(fill = Murder), map = states_map) +
    expand_limits(x = states_map$long, y = states_map$lat)

  last_plot() + coord_map()
  ggplot(crimesm, aes(map_id = state)) +
    geom_map(aes(fill = value), map = states_map) +
    expand_limits(x = states_map$long, y = states_map$lat) +
    facet_wrap(~ variable)
}

```

---

geom\_path

---

*Connect observations in original order*


---

## Description

Connect observations in original order

## Usage

```

geom_path(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", lineend = "butt", linejoin = "round",
  linemitre = 1, na.rm = FALSE, arrow = NULL, ...)

```

## Arguments

lineend	Line end style (round, butt, square)
linejoin	Line join style (round, mitre, bevel)
linemitre	Line mitre limit (number greater than 1)
arrow	Arrow specification, as created by <code>?grid::arrow</code>
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer

na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

- **x**
- **y**
- alpha
- colour
- linetype
- size

## See Also

[geom\\_line](#): Functional (ordered) lines; [geom\\_polygon](#): Filled paths (polygons); [geom\\_segment](#): Line segments

## Examples

```
# Generate data
library(plyr)
myear <- ddply(movies, .(year), colwise(mean, .(length, rating)))
p <- ggplot(myear, aes(length, rating))
p + geom_path()

# Add aesthetic mappings
p + geom_path(aes(size = year))
p + geom_path(aes(colour = year))

# Change scale
p + geom_path(aes(size = year)) + scale_size(range = c(1, 3))

# Set aesthetics to fixed value
p + geom_path(colour = "green")

# Control line join parameters
df <- data.frame(x = 1:3, y = c(4, 1, 9))
base <- ggplot(df, aes(x, y))
base + geom_path(size = 10)
base + geom_path(size = 10, lineend = "round")
base + geom_path(size = 10, linejoin = "mitre", lineend = "butt")

# Use qplot instead
qplot(length, rating, data=myear, geom="path")
```



```

# Using economic data:
# How is unemployment and personal savings rate related?
qplot(unemploy/pop, psavert, data=economics)
qplot(unemploy/pop, psavert, data=economics, geom="path")
qplot(unemploy/pop, psavert, data=economics, geom="path", size=as.numeric(date))

# How is rate of unemployment and length of unemployment?
qplot(unemploy/pop, uempmed, data=economics)
qplot(unemploy/pop, uempmed, data=economics, geom="path")
qplot(unemploy/pop, uempmed, data=economics, geom="path") +
  geom_point(data=head(economics, 1), colour="red") +
  geom_point(data=tail(economics, 1), colour="blue")
qplot(unemploy/pop, uempmed, data=economics, geom="path") +
  geom_text(data=head(economics, 1), label="1967", colour="blue") +
  geom_text(data=tail(economics, 1), label="2007", colour="blue")

# geom_path removes missing values on the ends of a line.
# use na.rm = T to suppress the warning message
df <- data.frame(
  x = 1:5,
  y1 = c(1, 2, 3, 4, NA),
  y2 = c(NA, 2, 3, 4, 5),
  y3 = c(1, 2, NA, 4, 5),
  y4 = c(1, 2, 3, 4, 5))
qplot(x, y1, data = df, geom = c("point", "line"))
qplot(x, y2, data = df, geom = c("point", "line"))
qplot(x, y3, data = df, geom = c("point", "line"))
qplot(x, y4, data = df, geom = c("point", "line"))

# Setting line type vs colour/size
# Line type needs to be applied to a line as a whole, so it can
# not be used with colour or size that vary across a line

x <- seq(0.01, .99, length=100)
df <- data.frame(x = rep(x, 2), y = c(qlogis(x), 2 * qlogis(x)), group = rep(c("a", "b"), each=100))
p <- ggplot(df, aes(x=x, y=y, group=group))

# Should work
p + geom_line(linetype = 2)
p + geom_line(aes(colour = group), linetype = 2)
p + geom_line(aes(colour = x))

# Should fail
should_stop(p + geom_line(aes(colour = x), linetype=2))

# Use the arrow parameter to add an arrow to the line
# See ?grid::arrow for more details
library(grid)
c <- ggplot(economics, aes(x = date, y = pop))
# Arrow defaults to "last"
c + geom_path(arrow = arrow())
c + geom_path(arrow = arrow(angle = 15, ends = "both", length = unit(0.6, "inches"))))

```

---

geom_point	<i>Points, as for a scatterplot</i>
------------	-------------------------------------

---

## Description

The point geom is used to create scatterplots.

## Usage

```
geom_point(mapping = NULL, data = NULL, stat = "identity",
           position = "identity", na.rm = FALSE, ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Details

The scatterplot is useful for displaying the relationship between two continuous variables, although it can also be used with one continuous and one categorical variable, or two categorical variables. See [geom\\_jitter](#) for possibilities.

The *bubblechart* is a scatterplot with a third variable mapped to the size of points. There are no special names for scatterplots where another variable is mapped to point shape or colour, however.

The biggest potential problem with a scatterplot is overplotting: whenever you have more than a few points, points may be plotted on top of one another. This can severely distort the visual appearance of the plot. There is no one solution to this problem, but there are some techniques that can help. You can add additional information with [stat\\_smooth](#), [stat\\_quantile](#) or [stat\\_density2d](#). If you have few unique x values, [geom\\_boxplot](#) may also be useful. Alternatively, you can summarise the number of points at each location and display that in some way, using [stat\\_sum](#). Another technique is to use transparent points, `geom_point(alpha = 0.05)`.

## Aesthetics

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

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

- alpha
- colour
- fill
- shape
- size

### See Also

[scale\\_size](#) to see scale area of points, instead of radius, [geom\\_jitter](#) to jitter points to reduce (mild) overplotting

### Examples

```
p <- ggplot(mtcars, aes(wt, mpg))
p + geom_point()

# Add aesthetic mappings
p + geom_point(aes(colour = qsec))
p + geom_point(aes(alpha = qsec))
p + geom_point(aes(colour = factor(cyl)))
p + geom_point(aes(shape = factor(cyl)))
p + geom_point(aes(size = qsec))

# Change scales
p + geom_point(aes(colour = cyl)) + scale_colour_gradient(low = "blue")
p + geom_point(aes(size = qsec)) + scale_size_area()
p + geom_point(aes(shape = factor(cyl))) + scale_shape(solid = FALSE)

# Set aesthetics to fixed value
p + geom_point(colour = "red", size = 3)
qplot(wt, mpg, data = mtcars, colour = I("red"), size = I(3))

# Varying alpha is useful for large datasets
d <- ggplot(diamonds, aes(carat, price))
d + geom_point(alpha = 1/10)
d + geom_point(alpha = 1/20)
d + geom_point(alpha = 1/100)

# You can create interesting shapes by layering multiple points of
# different sizes
p <- ggplot(mtcars, aes(mpg, wt))
p + geom_point(colour="grey50", size = 4) + geom_point(aes(colour = cyl))
p + aes(shape = factor(cyl)) +
  geom_point(aes(colour = factor(cyl)), size = 4) +
  geom_point(colour="grey90", size = 1.5)
p + geom_point(colour="black", size = 4.5) +
  geom_point(colour="pink", size = 4) +
  geom_point(aes(shape = factor(cyl)))

# These extra layers don't usually appear in the legend, but we can
```

```
# force their inclusion
p + geom_point(colour="black", size = 4.5, show_guide = TRUE) +
  geom_point(colour="pink", size = 4, show_guide = TRUE) +
  geom_point(aes(shape = factor(cyl)))

# Transparent points:
qplot(mpg, wt, data = mtcars, size = I(5), alpha = I(0.2))

# geom_point warns when missing values have been dropped from the data set
# and not plotted, you can turn this off by setting na.rm = TRUE
mtcars2 <- transform(mtcars, mpg = ifelse(runif(32) < 0.2, NA, mpg))
qplot(wt, mpg, data = mtcars2)
qplot(wt, mpg, data = mtcars2, na.rm = TRUE)

# Use qplot instead
qplot(wt, mpg, data = mtcars)
qplot(wt, mpg, data = mtcars, colour = factor(cyl))
qplot(wt, mpg, data = mtcars, colour = I("red"))
```

---

geom\_pointrange

*An interval represented by a vertical line, with a point in the middle.*


---

## Description

An interval represented by a vertical line, with a point in the middle.

## Usage

```
geom_pointrange(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

- **x**

- **y**
- **ymin**
- **ymax**
- **alpha**
- **colour**
- **fill**
- **linetype**
- **shape**
- **size**

### See Also

[geom\\_errorbar](#) for error bars, [geom\\_linerange](#) for range indicated by straight line, + examples, [geom\\_crossbar](#) for hollow bar with middle indicated by horizontal line, [stat\\_summary](#) for examples of these guys in use, [geom\\_smooth](#) for continuous analog"

### Examples

```
# See geom_linerange for examples
```

---

geom_polygon	<i>Polygon, a filled path.</i>
--------------	--------------------------------

---

### Description

Polygon, a filled path.

### Usage

```
geom_polygon(mapping = NULL, data = NULL, stat = "identity",
             position = "identity", ...)
```

### Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

- **x**
- **y**
- alpha
- colour
- fill
- linetype
- size

## See Also

[geom\\_path](#) for an unfilled polygon, [geom\\_ribbon](#) for a polygon anchored on the x-axis

## Examples

```
# When using geom_polygon, you will typically need two data frames:
# one contains the coordinates of each polygon (positions), and the
# other the values associated with each polygon (values). An id
# variable links the two together

ids <- factor(c("1.1", "2.1", "1.2", "2.2", "1.3", "2.3"))

values <- data.frame(
  id = ids,
  value = c(3, 3.1, 3.1, 3.2, 3.15, 3.5)
)

positions <- data.frame(
  id = rep(ids, each = 4),
  x = c(2, 1, 1.1, 2.2, 1, 0, 0.3, 1.1, 2.2, 1.1, 1.2, 2.5, 1.1, 0.3,
0.5, 1.2, 2.5, 1.2, 1.3, 2.7, 1.2, 0.5, 0.6, 1.3),
  y = c(-0.5, 0, 1, 0.5, 0, 0.5, 1.5, 1, 0.5, 1, 2.1, 1.7, 1, 1.5,
2.2, 2.1, 1.7, 2.1, 3.2, 2.8, 2.1, 2.2, 3.3, 3.2)
)

# Currently we need to manually merge the two together
datapoly <- merge(values, positions, by=c("id"))

(p <- ggplot(datapoly, aes(x=x, y=y)) + geom_polygon(aes(fill=value, group=id)))

# Which seems like a lot of work, but then it's easy to add on
# other features in this coordinate system, e.g.:

stream <- data.frame(
  x = cumsum(runif(50, max = 0.1)),
  y = cumsum(runif(50, max = 0.1))
)
```

```
p + geom_line(data = stream, colour="grey30", size = 5)

# And if the positions are in longitude and latitude, you can use
# coord_map to produce different map projections.
```

---

geom\_quantile

---

Add quantile lines from a quantile regression.

---

## Description

This can be used as a continuous analogue of a `geom_boxplot`.

## Usage

```
geom_quantile(mapping = NULL, data = NULL, stat = "quantile",
  position = "identity", lineend = "butt", linejoin = "round",
  linemitre = 1, na.rm = FALSE, ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.
lineend	Line end style (round, butt, square)
linejoin	Line join style (round, mitre, bevel)
linemitre	Line mitre limit (number greater than 1)

## Aesthetics

`geom_quantile` understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour
- linetype
- size
- weight

See Also

See [stat\\_quantile](#) for examples.

Examples

```
# See stat_quantile for examples
```

---

geom_raster	<i>High-performance rectangular tiling.</i>
-------------	---------------------------------------------

---

Description

This is a special case of [geom\\_tile](#) where all tiles are the same size. It is implemented highly efficiently using the internal rasterGrob function.

Usage

```
geom_raster(mapping = NULL, data = NULL, stat = "identity",  
  position = "identity", hjust = 0.5, vjust = 0.5, interpolate = FALSE,  
  ...)
```

Arguments

hjust, vjust	horizontal and vertical justification of the grob. Each justification value should be a number between 0 and 1. Defaults to 0.5 for both, centering each pixel over its data location.
interpolate	If TRUE interpolate linearly, if FALSE (the default) don't interpolate.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

Details

By default, `geom_raster` add a vertical and horizontal padding. The size of padding depends on the resolution of data. If you want to manually set the padding (e.g. want zero-padding), you can change the behavior by setting `hpad` and `vpad`.



**Aesthetics**

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

- **x**
- **y**
- alpha
- fill

**Examples**

```
# Generate data
pp <- function (n,r=4) {
  x <- seq(-r*pi, r*pi, len=n)
  df <- expand.grid(x=x, y=x)
  df$r <- sqrt(df$x^2 + df$y^2)
  df$z <- cos(df$r^2)*exp(-df$r/6)
  df
}
qplot(x, y, data = pp(20), fill = z, geom = "raster")
# Interpolation worsens the appearance of this plot, but can help when
# rendering images.
qplot(x, y, data = pp(20), fill = z, geom = "raster", interpolate = TRUE)

# For the special cases where it is applicable, geom_raster is much
# faster than geom_tile:
pp200 <- pp(200)
base <- ggplot(pp200, aes(x, y, fill = z))
benchplot(base + geom_raster())
benchplot(base + geom_tile())

# justification
df <- expand.grid(x = 0:5, y = 0:5)
df$z <- runif(nrow(df))
# default is compatible with geom_tile()
ggplot(df, aes(x, y, fill = z)) + geom_raster()
# zero padding
ggplot(df, aes(x, y, fill = z)) + geom_raster(hjust = 0, vjust = 0)
```

---

geom\_rect

2d rectangles.

---

**Description**

2d rectangles.

## Usage

```
geom_rect(mapping = NULL, data = NULL, stat = "identity",  
  position = "identity", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

- **xmax**
- **xmin**
- **ymax**
- **ymin**
- alpha
- colour
- fill
- linetype
- size

## Examples

```
df <- data.frame(  
  x = sample(10, 20, replace = TRUE),  
  y = sample(10, 20, replace = TRUE)  
)  
ggplot(df, aes(xmin = x, xmax = x + 1, ymin = y, ymax = y + 2)) +  
  geom_rect()
```

---

geom\_ribbon*Ribbons, y range with continuous x values.*

---

## Description

Ribbons, y range with continuous x values.

## Usage

```
geom_ribbon(mapping = NULL, data = NULL, stat = "identity",  
            position = "identity", na.rm = FALSE, ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

- **x**
- **ymin**
- **ymax**
- alpha
- colour
- fill
- linetype
- size

## See Also

[geom\\_bar](#) for discrete intervals (bars), [geom\\_linerange](#) for discrete intervals (lines), [geom\\_polygon](#) for general polygons"

## Examples

```
# Generate data
huron <- data.frame(year = 1875:1972, level = as.vector(LakeHuron))
library(plyr) # to access round_any
huron$decade <- round_any(huron$year, 10, floor)

h <- ggplot(huron, aes(x=year))

h + geom_ribbon(aes(ymin=0, ymax=level))
h + geom_area(aes(y = level))

# Add aesthetic mappings
h + geom_ribbon(aes(ymin=level-1, ymax=level+1))
h + geom_ribbon(aes(ymin=level-1, ymax=level+1)) + geom_line(aes(y=level))

# Take out some values in the middle for an example of NA handling
huron[huron$year > 1900 & huron$year < 1910, "level"] <- NA
h <- ggplot(huron, aes(x=year))
h + geom_ribbon(aes(ymin=level-1, ymax=level+1)) + geom_line(aes(y=level))

# Another data set, with multiple y's for each x
m <- ggplot(movies, aes(y=votes, x=year))
(m <- m + geom_point())

# The default summary isn't that useful
m + stat_summary(geom="ribbon", fun.ymin="min", fun.ymax="max")
m + stat_summary(geom="ribbon", fun.data="median_hilow")

# Use qplot instead
qplot(year, level, data=huron, geom=c("area", "line"))
```

---

geom\_rug

*Marginal rug plots.*


---

## Description

Marginal rug plots.

## Usage

```
geom_rug(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", sides = "bl", ...)
```

## Arguments

**sides** A string that controls which sides of the plot the rugs appear on. It can be set to a string containing any of "trbl", for top, right, bottom, and left.

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

- alpha
- colour
- linetype
- size

## Examples

```
p <- ggplot(mtcars, aes(x=wt, y=mpg))
p + geom_point()
p + geom_point() + geom_rug()
p + geom_point() + geom_rug(sides="b")    # Rug on bottom only
p + geom_point() + geom_rug(sides="trbl") # All four sides
p + geom_point() + geom_rug(position='jitter')
```

---

geom_segment	<i>Single line segments.</i>
--------------	------------------------------

---

## Description

Single line segments.

## Usage

```
geom_segment(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE,
  ...)
```

**Arguments**

arrow	specification for arrow heads, as created by <code>arrow()</code>
lineend	Line end style (round, butt, square)
mapping	The aesthetic mapping, usually constructed with <code>aes</code> or <code>aes_string</code> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments passed on to <code>layer</code> . This can include aesthetics whose values you want to set, not map. See <code>layer</code> for more details.

**Aesthetics**

`geom_segment` understands the following aesthetics (required aesthetics are in bold):

- **x**
- **xend**
- **y**
- **yend**
- alpha
- colour
- linetype
- size

**See Also**

`geom_path` and `geom_line` for multi- segment lines and paths.

**Examples**

```
library(grid) # needed for arrow function
p <- ggplot(seals, aes(x = long, y = lat))
(p <- p + geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat),
  arrow = arrow(length = unit(0.1, "cm"))))

if (require("maps")) {

  xlim <- range(seals$long)
  ylim <- range(seals$lat)
  usamap <- data.frame(map("world", xlim = xlim, ylim = ylim, plot =
    FALSE)[c("x", "y")])
  usamap <- rbind(usamap, NA, data.frame(map('state', xlim = xlim, ylim
    = ylim, plot = FALSE)[c("x", "y")]))
  names(usamap) <- c("long", "lat")
}
```

```

p + geom_path(data = usamap) + scale_x_continuous(limits = xlim)
}

# You can also use geom_segment to recreate plot(type = "h") :
counts <- as.data.frame(table(x = rpois(100,5)))
counts$x <- as.numeric(as.character(counts$x))
with(counts, plot(x, Freq, type = "h", lwd = 10))

qplot(x, Freq, data = counts, geom = "segment",
      yend = 0, xend = x, size = I(10))

# Adding line segments
library(grid) # needed for arrow function
b <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
b + geom_segment(aes(x = 2, y = 15, xend = 2, yend = 25))
b + geom_segment(aes(x = 2, y = 15, xend = 3, yend = 15))
b + geom_segment(aes(x = 5, y = 30, xend = 3.5, yend = 25),
  arrow = arrow(length = unit(0.5, "cm")))

```

geom\_smooth

*Add a smoothed conditional mean.*

## Description

Add a smoothed conditional mean.

## Usage

```
geom_smooth(mapping = NULL, data = NULL, stat = "smooth",
            position = "identity", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

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

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

- alpha
- colour
- fill
- linetype
- size
- weight

### See Also

The default stat for this geom is [stat\\_smooth](#) see that documentation for more options to control the underlying statistical transformation.

### Examples

```
# See stat_smooth for examples of using built in model fitting
# if you need some more flexible, this example shows you how to
# plot the fits from any model of your choosing
qplot(wt, mpg, data=mtcars, colour=factor(cyl))

model <- lm(mpg ~ wt + factor(cyl), data=mtcars)
grid <- with(mtcars, expand.grid(
  wt = seq(min(wt), max(wt), length = 20),
  cyl = levels(factor(cyl))
))

grid$mpg <- stats::predict(model, newdata=grid)

qplot(wt, mpg, data=mtcars, colour=factor(cyl)) + geom_line(data=grid)

# or with standard errors

err <- stats::predict(model, newdata=grid, se = TRUE)
grid$ucl <- err$fit + 1.96 * err$se.fit
grid$lcl <- err$fit - 1.96 * err$se.fit

qplot(wt, mpg, data=mtcars, colour=factor(cyl)) +
  geom_smooth(aes(ymin = lcl, ymax = ucl), data=grid, stat="identity")
```

---

geom\_step

*Connect observations by stairs.*

---

### Description

Connect observations by stairs.

### Usage

```
geom_step(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", direction = "hv", ...)
```



**Arguments**

direction	direction of stairs: 'vh' for vertical then horizontal, or 'hv' for horizontal then vertical
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Aesthetics**

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

- alpha
- colour
- linetype
- size

**Examples**

```
# Simple quantiles/ECDF from examples(plot)
x <- sort(rnorm(47))
qplot(seq_along(x), x, geom="step")

# Steps go horizontally, then vertically (default)
qplot(seq_along(x), x, geom="step", direction = "hv")
plot(x, type = "s")
# Steps go vertically, then horizontally
qplot(seq_along(x), x, geom="step", direction = "vh")
plot(x, type = "S")

# Also works with other aesthetics
df <- data.frame(
  x = sort(rnorm(50)),
  trt = sample(c("a", "b"), 50, rep = TRUE)
)
qplot(seq_along(x), x, data = df, geom="step", colour = trt)
```

---

`geom_text`*Textual annotations.*

---

## Description

Textual annotations.

## Usage

```
geom_text(mapping = NULL, data = NULL, stat = "identity",  
          position = "identity", parse = FALSE, ...)
```

## Arguments

<code>parse</code>	If TRUE, the labels will be parsed into expressions and displayed as described in <code>?plotmath</code>
<code>mapping</code>	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
<code>data</code>	A layer specific dataset - only needed if you want to override the plot defaults.
<code>stat</code>	The statistical transformation to use on the data for this layer.
<code>position</code>	The position adjustment to use for overlapping points on this layer
<code>...</code>	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

`geom_text` understands the following aesthetics (required aesthetics are in bold):

- **label**
- **x**
- **y**
- alpha
- angle
- colour
- family
- fontface
- hjust
- lineheight
- size
- vjust

**Examples**

```

p <- ggplot(mtcars, aes(x=wt, y=mpg, label=rownames(mtcars)))

p + geom_text()
# Change size of the label
p + geom_text(size=10)
p <- p + geom_point()

# Set aesthetics to fixed value
p + geom_text()
p + geom_point() + geom_text(hjust=0, vjust=0)
p + geom_point() + geom_text(angle = 45)

# Add aesthetic mappings
p + geom_text(aes(colour=factor(cyl)))
p + geom_text(aes(colour=factor(cyl))) + scale_colour_discrete(l=40)

p + geom_text(aes(size=wt))
p + geom_text(aes(size=wt)) + scale_size(range=c(3,6))

# You can display expressions by setting parse = TRUE. The
# details of the display are described in ?plotmath, but note that
# geom_text uses strings, not expressions.
p + geom_text(aes(label = paste(wt, "^(", cyl, ")", sep = "")),
  parse = TRUE)

# Add an annotation not from a variable source
c <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
c + geom_text(data = NULL, x = 5, y = 30, label = "plot mpg vs. wt")
# Or, you can use annotate
c + annotate("text", label = "plot mpg vs. wt", x = 2, y = 15, size = 8, colour = "red")

# Use qplot instead
qplot(wt, mpg, data = mtcars, label = rownames(mtcars),
  geom=c("point", "text"))
qplot(wt, mpg, data = mtcars, label = rownames(mtcars), size = wt) +
  geom_text(colour = "red")

# You can specify family, fontface and lineheight
p <- ggplot(mtcars, aes(x=wt, y=mpg, label=rownames(mtcars)))
p + geom_text(fontface=3)
p + geom_text(aes(fontface=am+1))
p + geom_text(aes(family=c("serif", "mono")[am+1]))

```

**Description**

Similar to [levelplot](#) and [image](#).

**Usage**

```
geom_tile(mapping = NULL, data = NULL, stat = "identity",
          position = "identity", ...)
```

**Arguments**

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Aesthetics**

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

- **x**
- **y**
- alpha
- colour
- fill
- linetype
- size

**Examples**

```
# Generate data
pp <- function (n,r=4) {
  x <- seq(-r*pi, r*pi, len=n)
  df <- expand.grid(x=x, y=x)
  df$r <- sqrt(df$x^2 + df$y^2)
  df$z <- cos(df$r^2)*exp(-df$r/6)
  df
}
p <- ggplot(pp(20), aes(x=x,y=y))

p + geom_tile() #pretty useless!

# Add aesthetic mappings
p + geom_tile(aes(fill=z))
```

```

# Change scale
p + geom_tile(aes(fill=z)) + scale_fill_gradient(low="green", high="red")

# Use qplot instead
qplot(x, y, data=pp(20), geom="tile", fill=z)
qplot(x, y, data=pp(100), geom="tile", fill=z)

# Missing values
p <- ggplot(pp(20)[sample(20*20, size=200),], aes(x=x,y=y,fill=z))
p + geom_tile()

# Input that works with image
image(t(volcano)[ncol(volcano):1,])
library(reshape2) # for melt
ggplot(melt(volcano), aes(x=Var1, y=Var2, fill=value)) + geom_tile()

# inspired by the image-density plots of Ken Knoblauch
cars <- ggplot(mtcars, aes(y=factor(cyl), x=mpg))
cars + geom_point()
cars + stat_bin(aes(fill=..count..), geom="tile", binwidth=3, position="identity")
cars + stat_bin(aes(fill=..density..), geom="tile", binwidth=3, position="identity")

cars + stat_density(aes(fill=..density..), geom="tile", position="identity")
cars + stat_density(aes(fill=..count..), geom="tile", position="identity")

# Another example with with unequal tile sizes
x.cell.boundary <- c(0, 4, 6, 8, 10, 14)
example <- data.frame(
  x = rep(c(2, 5, 7, 9, 12), 2),
  y = factor(rep(c(1,2), each=5)),
  z = rep(1:5, each=2),
  w = rep(diff(x.cell.boundary), 2)
)

qplot(x, y, fill=z, data=example, geom="tile")
qplot(x, y, fill=z, data=example, geom="tile", width=w)
qplot(x, y, fill=factor(z), data=example, geom="tile", width=w)

# You can manually set the colour of the tiles using
# scale_manual
col <- c("darkblue", "blue", "green", "orange", "red")
qplot(x, y, fill=col[z], data=example, geom="tile", width=w, group=1) +
  scale_fill_identity(labels=letters[1:5], breaks=col)

```

**Description**

Violin plot.

**Usage**

```
geom_violin(mapping = NULL, data = NULL, stat = "ydensity",
            position = "dodge", trim = TRUE, scale = "area", ...)
```

**Arguments**

trim	If TRUE (default), trim the tails of the violins to the range of the data. If FALSE, don't trim the tails.
scale	if "area" (default), all violins have the same area (before trimming the tails). If "count", areas are scaled proportionally to the number of observations. If "width", all violins have the same maximum width.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Aesthetics**

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

- **x**
- **y**
- alpha
- colour
- fill
- linetype
- size
- weight

**Examples**

```
p <- ggplot(mtcars, aes(factor(cyl), mpg))

p + geom_violin()
qplot(factor(cyl), mpg, data = mtcars, geom = "violin")

p + geom_violin() + geom_jitter(height = 0)
p + geom_violin() + coord_flip()
```

```

qplot(factor(cyl), mpg, data = mtcars, geom = "violin") +
  coord_flip()

# Scale maximum width proportional to sample size:
p + geom_violin(scale = "count")

# Scale maximum width to 1 for all violins:
p + geom_violin(scale = "width")

# Default is to trim violins to the range of the data. To disable:
p + geom_violin(trim = FALSE)

# Use a smaller bandwidth for closer density fit (default is 1).
p + geom_violin(adjust = .5)

# Add aesthetic mappings
# Note that violins are automatically dodged when any aesthetic is
# a factor
p + geom_violin(aes(fill = cyl))
p + geom_violin(aes(fill = factor(cyl)))
p + geom_violin(aes(fill = factor(vs)))
p + geom_violin(aes(fill = factor(am)))

# Set aesthetics to fixed value
p + geom_violin(fill = "grey80", colour = "#3366FF")
qplot(factor(cyl), mpg, data = mtcars, geom = "violin",
  colour = I("#3366FF"))

# Scales vs. coordinate transforms -----
# Scale transformations occur before the density statistics are computed.
# Coordinate transformations occur afterwards. Observe the effect on the
# number of outliers.
library(plyr) # to access round_any
m <- ggplot(movies, aes(y = votes, x = rating,
  group = round_any(rating, 0.5)))
m + geom_violin()
m + geom_violin() + scale_y_log10()
m + geom_violin() + coord_trans(y = "log10")
m + geom_violin() + scale_y_log10() + coord_trans(y = "log10")

# Violin plots with continuous x:
# Use the group aesthetic to group observations in violins
qplot(year, budget, data = movies, geom = "violin")
qplot(year, budget, data = movies, geom = "violin",
  group = round_any(year, 10, floor))

```

## Description

This geom allows you to annotate the plot with vertical lines (see [geom\\_hline](#) and [geom\\_abline](#) for other types of lines).

## Usage

```
geom_vline(mapping = NULL, data = NULL, stat = "vline",
           position = "identity", show_guide = FALSE, ...)
```

## Arguments

show_guide	should a legend be drawn? (defaults to FALSE)
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Details

There are two ways to use it. You can either specify the intercept of the line in the call to the geom, in which case the line will be in the same position in every panel. Alternatively, you can supply a different intercept for each panel using a data.frame. See the examples for the differences.

## Aesthetics

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

- alpha
- colour
- linetype
- size

## See Also

[geom\\_hline](#) for horizontal lines, [geom\\_abline](#) for lines defined by a slope and intercept, [geom\\_segment](#) for a more general approach"

## Examples

```
# Fixed lines
p <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()
p + geom_vline(xintercept = 5)
p + geom_vline(xintercept = 1:5)
p + geom_vline(xintercept = 1:5, colour="green", linetype = "longdash")
```



```

p + geom_vline(aes(xintercept = wt))

# With coordinate transforms
p + geom_vline(aes(xintercept = wt)) + coord_equal()
p + geom_vline(aes(xintercept = wt)) + coord_flip()
p + geom_vline(aes(xintercept = wt)) + coord_polar()

p2 <- p + aes(colour = factor(cyl))
p2 + geom_vline(xintercept = 15)

# To display different lines in different facets, you need to
# create a data frame.
p <- qplot(mpg, wt, data=mtcars, facets = vs ~ am)
vline.data <- data.frame(z = c(15, 20, 25, 30), vs = c(0, 0, 1, 1), am = c(0, 1, 0, 1))
p + geom_vline(aes(xintercept = z), vline.data)

```

---

ggplot.data.frame	Create a new ggplot plot from a data frame
-------------------	--------------------------------------------

---

## Description

Create a new ggplot plot from a data frame

## Usage

```

## S3 method for class 'data.frame'
ggplot(data, mapping = aes(), ...,
  environment = globalenv())

```

## Arguments

data	default data frame for plot
mapping	default list of aesthetic mappings (these can be colour, size, shape, line type – see individual geom functions for more details)
...	ignored
environment	in which evaluation of aesthetics should occur

## See Also

<http://had.co.nz/ggplot2>

---

ggplot2	ggplot2.
---------	----------

---

## Description

ggplot2.

ggsave

*Save a ggplot with sensible defaults***Description**

`ggsave` is a convenient function for saving a plot. It defaults to saving the last plot that you displayed, and for a default size uses the size of the current graphics device. It also guesses the type of graphics device from the extension. This means the only argument you need to supply is the filename.

**Usage**

```
ggsave(filename = default_name(plot), plot = last_plot(),
       device = default_device(filename), path = NULL, scale = 1,
       width = par("din")[1], height = par("din")[2], units = c("in", "cm",
       "mm"), dpi = 300, limitsize = TRUE, ...)
```

**Arguments**

<code>filename</code>	file name/filename of plot
<code>plot</code>	plot to save, defaults to last plot displayed
<code>device</code>	device to use, automatically extract from file name extension
<code>path</code>	path to save plot to (if you just want to set path and not filename)
<code>scale</code>	scaling factor
<code>width</code>	width (defaults to the width of current plotting window)
<code>height</code>	height (defaults to the height of current plotting window)
<code>units</code>	units for width and height when either one is explicitly specified (in, cm, or mm)
<code>dpi</code>	dpi to use for raster graphics
<code>limitsize</code>	when TRUE (the default), <code>ggsave</code> will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.
<code>...</code>	other arguments passed to graphics device

**Details**

`ggsave` currently recognises the extensions eps/ps, tex (pictex), pdf, jpeg, tiff, png, bmp, svg and wmf (windows only).

**Examples**

```
## Not run:
ratings <- qplot(rating, data=movies, geom="histogram")
qplot(length, data=movies, geom="histogram")
ggsave("length-hist.pdf")
ggsave("length-hist.png")
ggsave("ratings.pdf", ratings)
ggsave("ratings.pdf", ratings, width=4, height=4)
```

```
# make twice as big as on screen
ggsave("ratings.pdf", ratings, scale=2)

## End(Not run)
```

---

ggtheme

ggplot2 themes

---

## Description

Themes set the general aspect of the plot such as the colour of the background, gridlines, the size and colour of fonts.

## Usage

```
theme_grey(base_size = 12, base_family = "")

theme_gray(base_size = 12, base_family = "")

theme_bw(base_size = 12, base_family = "")

theme_linedraw(base_size = 12, base_family = "")

theme_light(base_size = 12, base_family = "")

theme_minimal(base_size = 12, base_family = "")

theme_classic(base_size = 12, base_family = "")
```

## Arguments

base_size	base font size
base_family	base font family

## Details

**theme\_gray** The signature ggplot2 theme with a grey background and white gridlines, designed to put the data forward yet make comparisons easy.

**theme\_bw** The classic dark-on-light ggplot2 theme. May work better for presentations displayed with a projector.

**theme\_linedraw** A theme with only black lines of various widths on white backgrounds, reminiscent of a line drawings. Serves a purpose similar to **theme\_bw**. Note that this theme has some very thin lines ( $\ll 1$  pt) which some journals may refuse.

**theme\_light** A theme similar to **theme\_linedraw** but with light grey lines and axes, to direct more attention towards the data.

**theme\_minimal** A minimalistic theme with no background annotations.

**theme\_classic** A classic-looking theme, with x and y axis lines and no gridlines.

**Examples**

```
p <- ggplot(mtcars) + geom_point(aes(x = wt, y = mpg,
  colour=factor(gear))) + facet_wrap(~am)
```

```
p
p + theme_gray()
p + theme_bw()
p + theme_linedraw()
p + theme_light()
p + theme_minimal()
p + theme_classic()
```

---

guides

---

*Set guides for each scale.*


---

**Description**

Guides for each scale can be set in call of `scale_*` with argument `guide`, or in `guides`.

**Usage**

```
guides(...)
```

**Arguments**

```
...           List of scale guide pairs
```

**Value**

A list containing the mapping between scale and guide.

**See Also**

Other guides: [guide\\_colorbar](#), [guide\\_colourbar](#); [guide\\_legend](#)

**Examples**

```
# ggplot object

dat <- data.frame(x = 1:5, y = 1:5, p = 1:5, q = factor(1:5),
  r = factor(1:5))
p <- ggplot(dat, aes(x, y, colour = p, size = q, shape = r)) + geom_point()

# without guide specification
p

# Show colorbar guide for colour.
# All these examples below have a same effect.
```

```

p + guides(colour = "colorbar", size = "legend", shape = "legend")
p + guides(colour = guide_colourbar(), size = guide_legend(),
  shape = guide_legend())
p +
  scale_colour_continuous(guide = "colorbar") +
  scale_size_discrete(guide = "legend") +
  scale_shape(guide = "legend")

# Guides are integrated where possible

p + guides(colour = guide_legend("title"), size = guide_legend("title"),
  shape = guide_legend("title"))
# same as
g <- guide_legend("title")
p + guides(colour = g, size = g, shape = g)

p + theme(legend.position = "bottom")

# position of guides

p + theme(legend.position = "bottom", legend.box = "horizontal")

# Set order for multiple guides

qplot(data = mpg, x = displ, y = cty, size = hwy, colour = cyl, shape = drv) +
  guides(colour = guide_colourbar(order = 1),
    alpha = guide_legend(order = 2),
    size = guide_legend(order = 3))

```

---

guide_colourbar	<i>Continuous colour bar guide.</i>
-----------------	-------------------------------------

---

## Description

Colour bar guide shows continuous color scales mapped onto values. Colour bar is available with `scale_fill` and `scale_colour`. For more information, see the inspiration for this function: [Matlab's colorbar function](#).

## Usage

```

guide_colourbar(title = waiver(), title.position = NULL,
  title.theme = NULL, title.hjust = NULL, title.vjust = NULL,
  label = TRUE, label.position = NULL, label.theme = NULL,
  label.hjust = NULL, label.vjust = NULL, barwidth = NULL,
  barheight = NULL, nbin = 20, raster = TRUE, ticks = TRUE,
  draw.ulim = TRUE, draw.llim = TRUE, direction = NULL,
  default.unit = "line", reverse = FALSE, order = 0, ...)

```

```
guide_colorbar(title = waiver(), title.position = NULL,
  title.theme = NULL, title.hjust = NULL, title.vjust = NULL,
  label = TRUE, label.position = NULL, label.theme = NULL,
  label.hjust = NULL, label.vjust = NULL, barwidth = NULL,
  barheight = NULL, nbin = 20, raster = TRUE, ticks = TRUE,
  draw.ulim = TRUE, draw.llim = TRUE, direction = NULL,
  default.unit = "line", reverse = FALSE, order = 0, ...)
```

## Arguments

barwidth	A numeric or a unit object specifying the width of the colorbar. Default value is legend.key.width or legend.key.size in <a href="#">theme</a> or theme.
barheight	A numeric or a unit object specifying the height of the colorbar. Default value is legend.key.height or legend.key.size in <a href="#">theme</a> or theme.
nbin	A numeric specifying the number of bins for drawing colorbar. A smoother colorbar for a larger value.
raster	A logical. If TRUE then the colorbar is rendered as a raster object. If FALSE then the colorbar is rendered as a set of rectangles. Note that not all graphics devices are capable of rendering raster image.
ticks	A logical specifying if tick marks on colorbar should be visible.
draw.ulim	A logical specifying if the upper limit tick marks should be visible.
draw.llim	A logical specifying if the lower limit tick marks should be visible.
direction	A character string indicating the direction of the guide. One of "horizontal" or "vertical."
default.unit	A character string indicating unit for barwidth
reverse	logical. If TRUE the colorbar is reversed. By default, the highest value is on the top and the lowest value is on the bottom
...	ignored.
title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <a href="#">waiver</a> ), the name of the scale object or the name specified in <a href="#">labs</a> is used for the title.
title.position	A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right."
title.theme	A theme object for rendering the title text. Usually the object of <a href="#">element_text</a> is expected. By default, the theme is specified by legend.title in <a href="#">theme</a> or theme.
title.hjust	A number specifying horizontal justification of the title text.
title.vjust	A number specifying vertical justification of the title text.
label	logical. If TRUE then the labels are drawn. If FALSE then the labels are invisible.
label.position	A character string indicating the position of a label. One of "top", "bottom" (default for horizontal guide), "left", or "right" (default for vertical guide).

label.theme	A theme object for rendering the label text. Usually the object of <a href="#">element_text</a> is expected. By default, the theme is specified by legend.text in <a href="#">theme</a> or theme.
label.hjust	A numeric specifying horizontal justification of the label text.
label.vjust	A numeric specifying vertical justification of the label text.
order	positive integer less than 99 that specify the order of this guide in the multiple guides. If 0 (default), the order is determined by a secret algorithm.

## Details

Guides can be specified in each scale or in [guides](#). `guide="legend"` in scale is syntax sugar for `guide=guide_legend()` - but the second form allows you to specify more options. As for how to specify the guide for each scales, see [guides](#).

## Value

A guide object

## See Also

Other guides: [guide\\_legend](#); [guides](#)

## Examples

```
library(reshape2) # for melt
df <- melt(outer(1:4, 1:4), varnames = c("X1", "X2"))

p1 <- ggplot(df, aes(X1, X2)) + geom_tile(aes(fill = value))
p2 <- p1 + geom_point(aes(size = value))

# Basic form
p1 + scale_fill_continuous(guide = "colorbar")
p1 + scale_fill_continuous(guide = guide_colorbar())
p1 + guides(fill = guide_colorbar())

# Control styles

# bar size
p1 + guides(fill = guide_colorbar(barwidth = 0.5, barheight = 10))

# no label
p1 + guides(fill = guide_colorbar(label = FALSE))

# no tick marks
p1 + guides(fill = guide_colorbar(ticks = FALSE))

# label position
p1 + guides(fill = guide_colorbar(label.position = "left"))

# label theme
p1 + guides(fill = guide_colorbar(label.theme = element_text(colour = "blue", angle = 0)))
```

```

# small number of bins
p1 + guides(fill = guide_colorbar(nbin = 3))

# large number of bins
p1 + guides(fill = guide_colorbar(nbin = 100))

# make top- and bottom-most ticks invisible
p1 + scale_fill_continuous(limits = c(0,20), breaks=c(0, 5, 10, 15, 20),
  guide = guide_colorbar(nbin=100, draw.ulim = FALSE, draw.llim = FALSE))

# guides can be controlled independently
p2 +
  scale_fill_continuous(guide = "colorbar") +
  scale_size(guide = "legend")
p2 + guides(fill = "colorbar", size = "legend")

p2 +
  scale_fill_continuous(guide = guide_colorbar(direction = "horizontal")) +
  scale_size(guide = guide_legend(direction = "vertical"))

```

guide\_legend

*Legend guide.***Description**

Legend type guide shows key (i.e., geoms) mapped onto values. Legend guides for various scales are integrated if possible.

**Usage**

```

guide_legend(title = waiver(), title.position = NULL, title.theme = NULL,
  title.hjust = NULL, title.vjust = NULL, label = TRUE,
  label.position = NULL, label.theme = NULL, label.hjust = NULL,
  label.vjust = NULL, keywidth = NULL, keyheight = NULL,
  direction = NULL, default.unit = "line", override.aes = list(),
  nrow = NULL, ncol = NULL, byrow = FALSE, reverse = FALSE, order = 0,
  ...)

```

**Arguments**

<code>title</code>	A character string or expression indicating a title of guide. If <code>NULL</code> , the title is not shown. By default ( <a href="#">waiver</a> ), the name of the scale object or the name specified in <a href="#">labs</a> is used for the title.
<code>title.position</code>	A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right."
<code>title.theme</code>	A theme object for rendering the title text. Usually the object of <a href="#">element_text</a> is expected. By default, the theme is specified by <code>legend.title</code> in <a href="#">theme</a> or <code>theme</code> .



title.hjust	A number specifying horizontal justification of the title text.
title.vjust	A number specifying vertical justification of the title text.
label	logical. If TRUE then the labels are drawn. If FALSE then the labels are invisible.
label.position	A character string indicating the position of a label. One of "top", "bottom" (default for horizontal guide), "left", or "right" (default for vertical guide).
label.theme	A theme object for rendering the label text. Usually the object of <code>element_text</code> is expected. By default, the theme is specified by <code>legend.text</code> in <code>theme</code> or <code>theme</code> .
label.hjust	A numeric specifying horizontal justification of the label text.
label.vjust	A numeric specifying vertical justification of the label text.
keywidth	A numeric or a unit object specifying the width of the legend key. Default value is <code>legend.key.width</code> or <code>legend.key.size</code> in <code>theme</code> or <code>theme</code> .
keyheight	A numeric or a unit object specifying the height of the legend key. Default value is <code>legend.key.height</code> or <code>legend.key.size</code> in <code>theme</code> or <code>theme</code> .
direction	A character string indicating the direction of the guide. One of "horizontal" or "vertical."
default.unit	A character string indicating unit for <code>keywidth</code> and <code>keyheight</code> .
override.aes	A list specifying aesthetic parameters of legend key. See details and examples.
nrow	The desired number of rows of legends.
ncol	The desired number of column of legends.
byrow	logical. If FALSE (the default) the legend-matrix is filled by columns, otherwise the legend-matrix is filled by rows.
reverse	logical. If TRUE the order of legends is reversed.
order	positive integer less than 99 that specify the order of this guide in the multiple guides. If 0 (default), the order is determined by a secret algorithm.
...	ignored.

## Details

Guides can be specified in each scale or in [guides](#). `guide="legend"` in scale is syntactic sugar for `guide=guide_legend()`. As for how to specify the guide for each scales in more detail, see [guides](#).

## Value

A guide object

## See Also

Other guides: [guide\\_colorbar](#), [guide\\_colourbar](#); [guides](#)

**Examples**

```

library(reshape2) # for melt
df <- melt(outer(1:4, 1:4), varnames = c("X1", "X2"))

p1 <- ggplot(df, aes(X1, X2)) + geom_tile(aes(fill = value))
p2 <- p1 + geom_point(aes(size = value))

# Basic form
p1 + scale_fill_continuous(guide = "legend")
p1 + scale_fill_continuous(guide = guide_legend())

# Guide title

p1 + scale_fill_continuous(guide = guide_legend(title = "V")) # title text
p1 + scale_fill_continuous(name = "V") # same
p1 + scale_fill_continuous(guide = guide_legend(title = NULL)) # no title

# Control styles

# key size
p1 + guides(fill = guide_legend(keywidth = 3, keyheight = 1))

# title position
p1 + guides(fill = guide_legend(title = "LEFT", title.position = "left"))

# title text styles via element_text
p1 + guides(fill = guide_legend(
  title.theme = element_text(size=15, face="italic", colour = "red", angle = 45)))

# label position
p1 + guides(fill = guide_legend(label.position = "bottom"))

# label styles
p1 + scale_fill_continuous(breaks = c(5, 10, 15),
  labels = paste("long", c(5, 10, 15)),
  guide = guide_legend(direction = "horizontal", title.position = "top",
    label.position="bottom", label.hjust = 0.5, label.vjust = 0.5,
    label.theme = element_text(angle = 90)))

# Set aesthetic of legend key

# very low alpha value make it difficult to see legend key
p3 <- qplot(carat, price, data = diamonds, colour = color,
  alpha = I(1/100))
p3

# override.aes overwrites the alpha
p3 + guides(colour = guide_legend(override.aes = list(alpha = 1)))

# multiple row/col legends
p <- qplot(1:20, 1:20, colour = letters[1:20])

```

```

p + guides(col = guide_legend(nrow = 8))
p + guides(col = guide_legend(ncol = 8))
p + guides(col = guide_legend(nrow = 8, byrow = TRUE))
p + guides(col = guide_legend(ncol = 8, byrow = TRUE))

# reversed order legend
p + guides(col = guide_legend(reverse = TRUE))

```

---

hmisc

*Wrap up a selection of summary functions from Hmisc to make it easy to use with [stat\\_summary](#).*

---

## Description

See the Hmisc documentation for details of their options.

## Usage

```

mean_cl_boot(x, ...)

mean_cl_normal(x, ...)

mean_sdl(x, ...)

median_hilow(x, ...)

```

## Arguments

**x** a numeric vector

**...** other arguments passed on to the respective Hmisc function.

## See Also

[smean.cl.boot](#), [smean.cl.normal](#), [smean.sdl](#), [smedian.hilow](#)

---

is.ggplot

*Reports whether x is a ggplot object*

---

## Description

Reports whether x is a ggplot object

## Usage

```
is.ggplot(x)
```

**Arguments**

x                      An object to test

---

<code>is.rel</code>	<i>Reports whether x is a rel object</i>
---------------------	------------------------------------------

---

**Description**

Reports whether x is a rel object

**Usage**

`is.rel(x)`

**Arguments**

x                      An object to test

---

<code>is.theme</code>	<i>Reports whether x is a theme object</i>
-----------------------	--------------------------------------------

---

**Description**

Reports whether x is a theme object

**Usage**

`is.theme(x)`

**Arguments**

x                      An object to test

---

labeller	<i>Generic labeller function for facets</i>
----------	---------------------------------------------

---

## Description

One-step function for providing methods or named character vectors for displaying labels in facets.

## Usage

```
labeller(..., keep.as.numeric = FALSE)
```

## Arguments

...                      Named arguments of the form `variable=values`, where values could be a vector or method.

keep.as.numeric                      logical, default TRUE. When FALSE, converts numeric values supplied as margins to the facet to characters.

## Details

The provided methods are checked for number of arguments. If the provided method takes less than two (e.g. `capitalize`), the method is passed values. Else (e.g. `label_both`), it is passed variable and values (in that order). If you want to be certain, use e.g. an anonymous function. If errors are returned such as “argument “.” is missing, with no default” or “unused argument (variable)”, matching the method’s arguments does not work as expected; make a wrapper function.

## Value

Function to supply to `facet_grid` for the argument `labeller`.

## Examples

```
p1 <- ggplot(mpg, aes(cty, hwy)) + geom_point()
p1 + facet_grid(cyl ~ class, labeller=label_both)
p1 + facet_grid(cyl ~ class, labeller=labeller(cyl=label_both))

ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point() +
  facet_grid(vs + am ~ gear, margins=TRUE,
             labeller=labeller(vs=label_both, am=label_both))

capitalize <- function(string) {
  substr(string, 1, 1) <- toupper(substr(string, 1, 1))
  string
}
conservation_status <- c('cd'='Conservation Dependent',
                        'en'='Endangered',
                        'lc'='Least concern',
```

```

      'nt'='Near Threatened',
      'vu'='Vulnerable',
      'domesticated'='Domesticated')
## Source: http://en.wikipedia.org/wiki/Wikipedia:Conservation_status

p2 <- ggplot(msleep, aes(x=sleep_total, y=awake)) + geom_point()
p2 + facet_grid(vore ~ conservation, labeller = labeller(vore = capitalize))

p2 + facet_grid(vore ~ conservation,
  labeller=labeller(vore = capitalize, conservation = conservation_status ))

# We could of course have renamed the levels;
# then we can apply another nifty function
msleep$conservation2 <- plyr::revalue(msleep$conservation, conservation_status)

p2 %>% msleep +
  facet_grid(vore ~ conservation2, labeller = labeller(vore = capitalize))
p2 %>% msleep +
  facet_grid(vore ~ conservation2, labeller = labeller(conservation2 =
    label_wrap_gen(10)))

```

---

label\_both

*Label facets with value and variable.*


---

## Description

Label facets with value and variable.

## Usage

```
label_both(variable, value)
```

## Arguments

variable	variable name passed in by facetter
value	variable value passed in by facetter

## See Also

Other facet labellers: [label\\_bquote](#); [label\\_parsed](#); [label\\_value](#)

## Examples

```

p <- qplot(wt, mpg, data = mtcars)
p + facet_grid(. ~ cyl)
p + facet_grid(. ~ cyl, labeller = label_both)

```

---

label_bquote	<i>Label facet with 'bquoted' expressions</i>
--------------	-----------------------------------------------

---

**Description**

See [bquote](#) for details on the syntax of the argument. The label value is x.

**Usage**

```
label_bquote(expr = beta^(x))
```

**Arguments**

expr	labelling expression to use
------	-----------------------------

**See Also**

[plotmath](#)

Other facet labellers: [label\\_both](#); [label\\_parsed](#); [label\\_value](#)

**Examples**

```
p <- qplot(wt, mpg, data = mtcars)
p + facet_grid(. ~ vs, labeller = label_bquote(alpha ^ .(x)))
p + facet_grid(. ~ vs, labeller = label_bquote(. (x) ^ .(x)))
```

---

label_parsed	<i>Label facets with parsed label.</i>
--------------	----------------------------------------

---

**Description**

Label facets with parsed label.

**Usage**

```
label_parsed(variable, value)
```

**Arguments**

variable	variable name passed in by facetter
value	variable value passed in by facetter

**See Also**

[plotmath](#)

Other facet labellers: [label\\_both](#); [label\\_bquote](#); [label\\_value](#)

**Examples**

```
mtcars$cyl2 <- factor(mtcars$cyl, labels = c("alpha", "beta", "gamma"))
qplot(wt, mpg, data = mtcars) + facet_grid(. ~ cyl2)
qplot(wt, mpg, data = mtcars) + facet_grid(. ~ cyl2,
  labeller = label_parsed)
```

---

label_value	<i>Label facets with their value. This is the default labelling scheme.</i>
-------------	-----------------------------------------------------------------------------

---

**Description**

Label facets with their value. This is the default labelling scheme.

**Usage**

```
label_value(variable, value)
```

**Arguments**

variable	variable name passed in by facetter
value	variable value passed in by facetter

**See Also**

Other facet labellers: [label\\_both](#); [label\\_bquote](#); [label\\_parsed](#)

**Examples**

```
p <- qplot(wt, mpg, data = mtcars)
p + facet_grid(. ~ cyl)
p + facet_grid(. ~ cyl, labeller = label_value)
```

---

label_wrap_gen	<i>Label facets with a word wrapped label.</i>
----------------	------------------------------------------------

---

**Description**

Uses [strwrap](#) for line wrapping.

**Usage**

```
label_wrap_gen(width = 25)
```

**Arguments**

width	integer, target column width for output.
-------	------------------------------------------



**See Also**

[, labeller](#)

---

labs

*Change axis labels and legend titles*

---

**Description**

Change axis labels and legend titles

**Usage**

```
labs(...)  
  
xlab(label)  
  
ylab(label)  
  
ggtitle(label)
```

**Arguments**

label	The text for the axis or plot title.
...	a list of new names in the form <code>aesthetic = "new name"</code>

**Examples**

```
p <- qplot(mpg, wt, data = mtcars)  
p + labs(title = "New plot title")  
p + labs(x = "New x label")  
p + xlab("New x label")  
p + ylab("New y label")  
p + ggtitle("New plot title")  
  
# This should work independently of other functions that modify the  
# the scale names  
p + ylab("New y label") + ylim(2, 4)  
p + ylim(2, 4) + ylab("New y label")  
  
# The labs function also modifies legend labels  
p <- qplot(mpg, wt, data = mtcars, colour = cyl)  
p + labs(colour = "Cylinders")  
  
# Can also pass in a list, if that is more convenient  
p + labs(list(title = "Title", x = "X", y = "Y"))
```

---

last_plot	<i>Retrieve the last plot to be modified or created.</i>
-----------	----------------------------------------------------------

---

### Description

Retrieve the last plot to be modified or created.

### Usage

```
last_plot()
```

### See Also

[ggsave](#)

---

map_data	<i>Create a data frame of map data.</i>
----------	-----------------------------------------

---

### Description

Create a data frame of map data.

### Usage

```
map_data(map, region = ".", exact = FALSE, ...)
```

### Arguments

map	name of map provided by the <b>maps</b> package. These include <a href="#">county</a> , <a href="#">france</a> , <a href="#">italy</a> , <a href="#">nz</a> , <a href="#">state</a> , <a href="#">usa</a> , <a href="#">world</a> , <a href="#">world2</a> .
region	name of subregions to include. Defaults to . which includes all subregion. See documentation for <a href="#">map</a> for more details.
exact	should the region be treated as a regular expression (FALSE) or as a fixed string (TRUE).
...	all other arguments passed on to <a href="#">map</a>

### Examples

```
if (require("maps")) {
  states <- map_data("state")
  arrests <- USArrests
  names(arrests) <- tolower(names(arrests))
  arrests$region <- tolower(rownames(USArrests))

  choro <- merge(states, arrests, sort = FALSE, by = "region")
}
```

```
choro <- choro[order(choro$order), ]
qplot(long, lat, data = choro, group = group, fill = assault,
       geom = "polygon")
qplot(long, lat, data = choro, group = group, fill = assault / murder,
       geom = "polygon")
}
```

---

**mean\_se***Calculate mean and standard errors on either side.*

---

### Description

Calculate mean and standard errors on either side.

### Usage

```
mean_se(x, mult = 1)
```

### Arguments

x	numeric vector
mult	number of multiples of standard error

### See Also

for use with [stat\\_summary](#)

---

**midwest***Midwest demographics.*

---

### Description

Demographic information of midwest counties

### Usage

```
data(midwest)
```

### Format

A data frame with 437 rows and 28 variables

**Details**

The variables are as follows:

- PID
- county
- state
- area
- poptotal. Total population
- popdensity. Population density
- popwhite. Number of whites.
- popblack. Number of blacks.
- popamerindian. Number of American Indians.
- popasian. Number of Asians.
- popother. Number of other races.
- percwhite. Percent white.
- percblack. Percent black.
- percamerindian. Percent American Indian.
- percasian. Percent Asian.
- percother. Percent other races.
- popadults. Number of adults.
- perchsd.
- percollege. Percent college educated.
- percprof. Percent profession.
- poppovertyknown.
- percpovertyknown
- percbelowpoverty
- percchildbelowpovert
- percadultpoverty
- percelderlypoverty
- inmetro. In a metro area.
- category'

---

movies

*Movie information and user ratings from IMDB.com.*

---

## Description

The internet movie database, <http://imdb.com/>, is a website devoted to collecting movie data supplied by studios and fans. It claims to be the biggest movie database on the web and is run by amazon. More about information imdb.com can be found online, [http://imdb.com/help/show\\_leaf?about](http://imdb.com/help/show_leaf?about), including information about the data collection process, [http://imdb.com/help/show\\_leaf?infosource](http://imdb.com/help/show_leaf?infosource).

## Usage

```
data(movies)
```

## Format

A data frame with 28819 rows and 24 variables

## Details

Movies were selected for inclusion if they had a known length and had been rated by at least one imdb user. The data set contains the following fields:

- title. Title of the movie.
- year. Year of release.
- budget. Total budget (if known) in US dollars
- length. Length in minutes.
- rating. Average IMDB user rating.
- votes. Number of IMDB users who rated this movie.
- r1-10. Multiplying by ten gives percentile (to nearest 10%) of users who rated this movie a 1.
- mpaa. MPAA rating.
- action, animation, comedy, drama, documentary, romance, short. Binary variables representing if movie was classified as belonging to that genre.

## References

<http://had.co.nz/data/movies/>

---

`mpg`*Fuel economy data from 1999 and 2008 for 38 popular models of car*

---

**Description**

This dataset contains a subset of the fuel economy data that the EPA makes available on <http://fuelconomy.gov>. It contains only models which had a new release every year between 1999 and 2008 - this was used as a proxy for the popularity of the car.

**Usage**

```
data(mpg)
```

**Format**

A data frame with 234 rows and 11 variables

**Details**

- manufacturer.
- model.
- displ. engine displacement, in litres
- year.
- cyl. number of cylinders
- trans. type of transmission
- drv. f = front-wheel drive, r = rear wheel drive, 4 = 4wd
- ct. city miles per gallon
- hwy. highway miles per gallon
- fl.
- class.

---

`msleep`*An updated and expanded version of the mammals sleep dataset.*

---

**Description**

This is an updated and expanded version of the mammals sleep dataset. Updated sleep times and weights were taken from V. M. Savage and G. B. West. A quantitative, theoretical framework for understanding mammalian sleep. Proceedings of the National Academy of Sciences, 104 (3):1051-1056, 2007.

**Usage**

```
data(msleep)
```

**Format**

A data frame with 83 rows and 11 variables

**Details**

Additional variables order, conservation status and vore were added from wikipedia.

- name. common name
- genus.
- vore. carnivore, omnivore or herbivore?
- order.
- conservation. the conservation status of the animal
- sleep\\_total. total amount of sleep, in hours
- sleep\\_rem. rem sleep, in hours
- sleep\\_cycle. length of sleep cycle, in hours
- awake. amount of time spent awake, in hours
- brainwt. brain weight in kilograms
- bodywt. body weight in kilograms

---

opts

*Build a theme (or partial theme) from theme elements*

---

**Description**

opts is deprecated. See the [theme](#) function.

**Usage**

```
opts(...)
```

**Arguments**

... Arguments to be passed on to the theme function.

---

position\_dodge

Adjust position by dodging overlaps to the side.

---

## Description

Adjust position by dodging overlaps to the side.

## Usage

```
position_dodge(width = NULL, height = NULL)
```

## Arguments

width	Manually specify width (does not affect all position adjustments)
height	Manually specify height (does not affect all position adjustments)

## See Also

Other position adjustments: [position\\_fill](#); [position\\_identity](#); [position\\_jitterdodge](#); [position\\_jitter](#); [position\\_stack](#)

## Examples

```
ggplot(mtcars, aes(x=factor(cyl), fill=factor(vs))) +
  geom_bar(position="dodge")
ggplot(diamonds, aes(x=price, fill=cut)) + geom_bar(position="dodge")
# see ?geom_boxplot and ?geom_bar for more examples

# Dodging things with different widths is tricky
df <- data.frame(x=c("a","a","b","b"), y=1:4, g = rep(1:2, 2))
(p <- qplot(x, y, data=df, group=g, position="dodge", geom="bar",
  stat="identity"))

p + geom_linerange(aes(ymin = y-1, ymax = y+1), position="dodge")
# You need to explicitly specify the width for dodging
p + geom_linerange(aes(ymin = y-1, ymax = y+1),
  position = position_dodge(width = 0.9))

# Similarly with error bars:
p + geom_errorbar(aes(ymin = y-1, ymax = y+1), width = 0.2,
  position="dodge")
p + geom_errorbar(aes(ymin = y-1, ymax = y+1, width = 0.2),
  position = position_dodge(width = 0.90))
```



---

position_fill	<i>Stack overlapping objects on top of one another, and standardise to have equal height.</i>
---------------	-----------------------------------------------------------------------------------------------

---

### Description

Stack overlapping objects on top of one another, and standardise to have equal height.

### Usage

```
position_fill(width = NULL, height = NULL)
```

### Arguments

width	Manually specify width (does not affect all position adjustments)
height	Manually specify height (does not affect all position adjustments)

### See Also

See [geom\\_bar](#) and [geom\\_area](#) for more examples.

Other position adjustments: [position\\_dodge](#); [position\\_identity](#); [position\\_jitterdodge](#); [position\\_jitter](#); [position\\_stack](#)

### Examples

```
# See ?geom_bar and ?geom_area for more examples
ggplot(mtcars, aes(x=factor(cyl), fill=factor(vs))) +
  geom_bar(position="fill")

cde <- geom_histogram(position="fill", binwidth = 500)

ggplot(diamonds, aes(x=price)) + cde
ggplot(diamonds, aes(x=price, fill=cut)) + cde
ggplot(diamonds, aes(x=price, fill=clarity)) + cde
ggplot(diamonds, aes(x=price, fill=color)) + cde
```

---

position_identity	<i>Don't adjust position</i>
-------------------	------------------------------

---

### Description

Don't adjust position

**Usage**

```
position_identity(width = NULL, height = NULL)
```

**Arguments**

width	Manually specify width (does not affect all position adjustments)
height	Manually specify height (does not affect all position adjustments)

**See Also**

Other position adjustments: [position\\_dodge](#); [position\\_fill](#); [position\\_jitterdodge](#); [position\\_jitter](#); [position\\_stack](#)

---

position_jitter	<i>Jitter points to avoid overplotting.</i>
-----------------	---------------------------------------------

---

**Description**

Jitter points to avoid overplotting.

**Usage**

```
position_jitter(width = NULL, height = NULL)
```

**Arguments**

width	degree of jitter in x direction. Defaults to 40% of the resolution of the data.
height	degree of jitter in y direction. Defaults to 40% of the resolution of the data

**See Also**

Other position adjustments: [position\\_dodge](#); [position\\_fill](#); [position\\_identity](#); [position\\_jitterdodge](#); [position\\_stack](#)

**Examples**

```
qplot(am, vs, data = mtcars)

# Default amount of jittering will generally be too much for
# small datasets:
qplot(am, vs, data = mtcars, position = "jitter")
# Control the amount as follows
qplot(am, vs, data = mtcars, position = position_jitter(w = 0.1, h = 0.1))

# With ggplot
ggplot(mtcars, aes(x = am, y = vs)) + geom_point(position = "jitter")
ggplot(mtcars, aes(x = am, y = vs)) + geom_point(position = position_jitter(w = 0.1, h = 0.1))
```

```
# The default works better for large datasets, where it will  
# take up as much space as a boxplot or a bar  
qplot(class, hwy, data = mpg, geom = c("boxplot", "jitter"))
```

---

position\_jitterdodge    *Adjust position by simultaneously dodging and jittering*

---

## Description

This is primarily used for aligning points generated through `geom_point()` with dodged boxplots (e.g., a `geom_boxplot()` with a fill aesthetic supplied).

## Usage

```
position_jitterdodge(jitter.width = NULL, jitter.height = NULL,  
  dodge.width = NULL)
```

## Arguments

<code>jitter.width</code>	degree of jitter in x direction. Defaults to 40% of the resolution of the data.
<code>jitter.height</code>	degree of jitter in y direction. Defaults to 0.
<code>dodge.width</code>	the amount to dodge in the x direction. Defaults to 0.75, the default <code>position_dodge()</code> width.

## See Also

Other position adjustments: [position\\_dodge](#); [position\\_fill](#); [position\\_identity](#); [position\\_jitter](#); [position\\_stack](#)

## Examples

```
dsub <- diamonds[ sample(nrow(diamonds), 1000), ]  
ggplot(dsub, aes(x = cut, y = carat, fill = clarity)) +  
  geom_boxplot(outlier.size = 0) +  
  geom_point(pch = 21, position = position_jitterdodge())
```

---

position\_stack

Stack overlapping objects on top of one another.

---

## Description

Stack overlapping objects on top of one another.

## Usage

```
position_stack(width = NULL, height = NULL)
```

## Arguments

width	Manually specify width (does not affect all position adjustments)
height	Manually specify height (does not affect all position adjustments)

## See Also

Other position adjustments: [position\\_dodge](#); [position\\_fill](#); [position\\_identity](#); [position\\_jitterdodge](#); [position\\_jitter](#)

## Examples

```
# Stacking is the default behaviour for most area plots:
ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) + geom_bar()

# To change stacking order, use factor() to change order of levels
mtcars$vs <- factor(mtcars$vs, levels = c(1,0))
ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) + geom_bar()

ggplot(diamonds, aes(price)) + geom_histogram(binwidth=500)
ggplot(diamonds, aes(price, fill = cut)) + geom_histogram(binwidth=500)

# Stacking is also useful for time series
data.set <- data.frame(
  Time = c(rep(1, 4), rep(2, 4), rep(3, 4), rep(4, 4)),
  Type = rep(c('a', 'b', 'c', 'd'), 4),
  Value = rpois(16, 10)
)

qplot(Time, Value, data = data.set, fill = Type, geom = "area")
# If you want to stack lines, you need to say so:
qplot(Time, Value, data = data.set, colour = Type, geom = "line")
qplot(Time, Value, data = data.set, colour = Type, geom = "line",
  position = "stack")
# But realise that this makes it *much* harder to compare individual
# trends
```

---

presidential	<i>Terms of 10 presidents from Eisenhower to Bush W.</i>
--------------	----------------------------------------------------------

---

**Description**

The names of each president, the start and end date of their term, and their party of 10 US presidents from Eisenhower to Bush W.

**Usage**

```
data(presidential)
```

**Format**

A data frame with 10 rows and 4 variables

---

print.ggplot	<i>Draw plot on current graphics device.</i>
--------------	----------------------------------------------

---

**Description**

Draw plot on current graphics device.

**Usage**

```
## S3 method for class 'ggplot'
print(x, newpage = is.null(vp), vp = NULL, ...)

## S3 method for class 'ggplot'
plot(x, newpage = is.null(vp), vp = NULL, ...)
```

**Arguments**

x	plot to display
newpage	draw new (empty) page first?
vp	viewport to draw plot in
...	other arguments not used by this method

qplot

*Quick plot***Description**

qplot is the basic plotting function in the ggplot2 package, designed to be familiar if you're used to `plot` from the base package. It is a convenient wrapper for creating a number of different types of plots using a consistent calling scheme. See <http://had.co.nz/ggplot2/book/qplot.pdf> for the chapter in the ggplot2 book which describes the usage of qplot in detail.

**Usage**

```
qplot(x, y = NULL, ..., data, facets = NULL, margins = FALSE,
      geom = "auto", stat = list(NULL), position = list(NULL), xlim = c(NA,
      NA), ylim = c(NA, NA), log = "", main = NULL,
      xlab = deparse(substitute(x)), ylab = deparse(substitute(y)), asp = NA)
```

**Arguments**

x	x values
y	y values
...	other aesthetics passed for each layer
data	data frame to use (optional). If not specified, will create one, extracting vectors from the current environment.
facets	faceting formula to use. Picks <code>facet_wrap</code> or <code>facet_grid</code> depending on whether the formula is one sided or two-sided
margins	whether or not margins will be displayed
geom	character vector specifying geom to use. Defaults to "point" if x and y are specified, and "histogram" if only x is specified.
stat	character vector specifying statistics to use
position	character vector giving position adjustment to use
xlim	limits for x axis
ylim	limits for y axis
log	which variables to log transform ("x", "y", or "xy")
main	character vector or expression for plot title
xlab	character vector or expression for x axis label
ylab	character vector or expression for y axis label
asp	the y/x aspect ratio

## Examples

```
# Use data from data.frame
qplot(mpg, wt, data=mtcars)
qplot(mpg, wt, data=mtcars, colour=cyl)
qplot(mpg, wt, data=mtcars, size=cyl)
qplot(mpg, wt, data=mtcars, facets=vs ~ am)

# It will use data from local environment
hp <- mtcars$hp
wt <- mtcars$wt
cyl <- mtcars$cyl
vs <- mtcars$vs
am <- mtcars$am
qplot(hp, wt)
qplot(hp, wt, colour=cyl)
qplot(hp, wt, size=cyl)
qplot(hp, wt, facets=vs ~ am)

qplot(1:10, rnorm(10), colour = runif(10))
qplot(1:10, letters[1:10])
mod <- lm(mpg ~ wt, data=mtcars)
qplot(resid(mod), fitted(mod))
qplot(resid(mod), fitted(mod), facets = . ~ vs)

f <- function() {
  a <- 1:10
  b <- a ^ 2
  qplot(a, b)
}
f()

# qplot will attempt to guess what geom you want depending on the input
# both x and y supplied = scatterplot
qplot(mpg, wt, data = mtcars)
# just x supplied = histogram
qplot(mpg, data = mtcars)
# just y supplied = scatterplot, with x = seq_along(y)
qplot(y = mpg, data = mtcars)

# Use different geoms
qplot(mpg, wt, data = mtcars, geom="path")
qplot(factor(cyl), wt, data = mtcars, geom=c("boxplot", "jitter"))
qplot(mpg, data = mtcars, geom = "dotplot")
```

**Description**

Relative sizing for theme elements

**Usage**

```
rel(x)
```

**Arguments**

x                      A number representing the relative size

**Examples**

```
qplot(1:3, 1:3) + theme(axis.title.x = element_text(size = rel(2.5)))
```

---

resolution	<i>Compute the "resolution" of a data vector.</i>
------------	---------------------------------------------------

---

**Description**

The resolution is the smallest non-zero distance between adjacent values. If there is only one unique value, then the resolution is defined to be one.

**Usage**

```
resolution(x, zero = TRUE)
```

**Arguments**

x                      numeric vector

zero                    should a zero value be automatically included in the computation of resolution

**Details**

If x is an integer vector, then it is assumed to represent a discrete variable, and the resolution is 1.

**Examples**

```
resolution(1:10)
resolution((1:10) - 0.5)
resolution((1:10) - 0.5, FALSE)
resolution(c(1,2, 10, 20, 50))
resolution(as.integer(c(1, 10, 20, 50))) # Returns 1
```



---

scale_alpha	<i>Alpha scales.</i>
-------------	----------------------

---

### Description

scale\_alpha is an alias for scale\_alpha\_continuous since that is the most common use of alpha, and it saves a bit of typing.

### Usage

```
scale_alpha(..., range = c(0.1, 1))

scale_alpha_continuous(..., range = c(0.1, 1))

scale_alpha_discrete(..., range = c(0.1, 1))
```

### Arguments

...	Other arguments passed on to <a href="#">continuous_scale</a> or <a href="#">discrete_scale</a> as appropriate, to control name, limits, breaks, labels and so forth.
range	range of output alpha values. Should lie between 0 and 1.

### Examples

```
(p <- qplot(mpg, cyl, data = mtcars, alpha = cyl))
p + scale_alpha("cylinders")
p + scale_alpha("number\nof\ncylinders")

p + scale_alpha(range = c(0.4, 0.8))

(p <- qplot(mpg, cyl, data=mtcars, alpha = factor(cyl)))
p + scale_alpha_discrete(range = c(0.4, 0.8))
```

---

scale_area	<i>Scale area instead of radius (for size).</i>
------------	-------------------------------------------------

---

### Description

[scale\\_area](#) is deprecated and will be removed in a future version of ggplot2. Use [scale\\_size\\_area](#) instead. Note that the default behavior of [scale\\_size\\_area](#) is slightly different: by default, it makes the area proportional to the numeric value.

### Usage

```
scale_area(..., range = c(1, 6))
```

**Arguments**

...	Other arguments passed on to <a href="#">continuous_scale</a> to control name, limits, breaks, labels and so forth.
range	Range of output sizes. Should be greater than 0.

---

scale_colour_brewer	<i>Sequential, diverging and qualitative colour scales from <a href="http://colorbrewer.org">colorbrewer.org</a></i>
---------------------	----------------------------------------------------------------------------------------------------------------------

---

**Description**

ColorBrewer provides sequential, diverging and qualitative colour schemes which are particularly suited and tested to display discrete values (levels of a factor) on a map. `ggplot2` can use those colours in discrete scales. It also allows to smoothly interpolate 6 colours from any palette to a continuous scale (6 colours per palette gives nice gradients; more results in more saturated colours which do not look as good). However, the original colour schemes (particularly the qualitative ones) were not intended for this and the perceptual result is left to the appreciation of the user.

**Usage**

```
scale_colour_brewer(..., type = "seq", palette = 1)

scale_fill_brewer(..., type = "seq", palette = 1)

scale_colour_distiller(..., type = "seq", palette = 1, values = NULL,
  space = "Lab", na.value = "grey50")

scale_fill_distiller(..., type = "seq", palette = 1, values = NULL,
  space = "Lab", na.value = "grey50")

scale_color_brewer(..., type = "seq", palette = 1)

scale_color_distiller(..., type = "seq", palette = 1, values = NULL,
  space = "Lab", na.value = "grey50")
```

**Arguments**

type	One of seq (sequential), div (diverging) or qual (qualitative)
palette	If a string, will use that named palette. If a number, will index into the list of palettes of appropriate type
...	Other arguments passed on to <a href="#">discrete_scale</a> to control name, limits, breaks, labels and so forth.
na.value	Colour to use for missing values
values	if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the colours vector. See <a href="#">rescale</a> for a convenience function to map an arbitrary range to between 0 and 1.

space                      colour space in which to calculate gradient. "Lab" usually best unless gradient goes through white.

## Details

See <http://colorbrewer2.org> for more information.

## See Also

Other colour scales: `scale_color_continuous`, `scale_color_gradient`, `scale_colour_continuous`, `scale_colour_gradient`, `scale_fill_continuous`, `scale_fill_gradient`; `scale_color_discrete`, `scale_color_hue`, `scale_colour_discrete`, `scale_colour_hue`, `scale_fill_discrete`, `scale_fill_hue`; `scale_color_gradient2`, `scale_colour_gradient2`, `scale_fill_gradient2`; `scale_color_gradientn`, `scale_colour_gradientn`, `scale_fill_gradientn`; `scale_color_grey`, `scale_colour_grey`, `scale_fill_grey`

## Examples

```
dsamp <- diamonds[sample(nrow(diamonds), 1000), ]
(d <- qplot(carat, price, data = dsamp, colour = clarity))

# Change scale label
d + scale_colour_brewer()
d + scale_colour_brewer("clarity")
d + scale_colour_brewer(expression(clarity[beta]))

# Select brewer palette to use, see ?scales::brewer_pal for more details
d + scale_colour_brewer(type = "seq")
d + scale_colour_brewer(type = "seq", palette = 3)

d + scale_colour_brewer(palette = "Blues")
d + scale_colour_brewer(palette = "Set1")

# scale_fill_brewer works just the same as
# scale_colour_brewer but for fill colours
ggplot(diamonds, aes(x = price, fill = cut)) +
  geom_histogram(position = "dodge", binwidth = 1000) +
  scale_fill_brewer()

# Generate map data
library(reshape2) # for melt
volcano3d <- melt(volcano)
names(volcano3d) <- c("x", "y", "z")

# Basic plot
v <- ggplot() + geom_tile(aes(x = x, y = y, fill = z), data = volcano3d)
v
v + scale_fill_distiller()
v + scale_fill_distiller(palette = 2)
v + scale_fill_distiller(type = "div")
v + scale_fill_distiller(palette = "Spectral")
v + scale_fill_distiller(palette = "Spectral", trans = "reverse")
```

```
v + scale_fill_distiller(type = "qual")
# Not appropriate for continuous data, issues a warning
```

---

scale\_colour\_gradient *Smooth gradient between two colours*

---

## Description

Default colours are generated with **munsell** and `mns1(c("2.5PB 2/4", "2.5PB 7/10"))`. Generally, for continuous colour scales you want to keep hue constant, but vary chroma and luminance. The **munsell** package makes this easy to do using the Munsell colour system.

## Usage

```
scale_colour_gradient(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "colourbar")

scale_fill_gradient(..., low = "#132B43", high = "#56B1F7", space = "Lab",
  na.value = "grey50", guide = "colourbar")

scale_colour_continuous(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "colourbar")

scale_fill_continuous(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "colourbar")

scale_color_continuous(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "colourbar")

scale_color_gradient(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "colourbar")
```

## Arguments

guide	Type of legend. Use "colourbar" for continuous colour bar, or "legend" for discrete colour legend.
...	Other arguments passed on to <a href="#">discrete_scale</a> to control name, limits, breaks, labels and so forth.
na.value	Colour to use for missing values
low	colour for low end of gradient.
high	colour for high end of gradient.
space	colour space in which to calculate gradient. "Lab" usually best unless gradient goes through white.

**See Also**

[seq\\_gradient\\_pal](#) for details on underlying palette

Other colour scales: [scale\\_color\\_brewer](#), [scale\\_color\\_distiller](#), [scale\\_colour\\_brewer](#), [scale\\_colour\\_distiller](#), [scale\\_fill\\_brewer](#), [scale\\_fill\\_distiller](#); [scale\\_color\\_discrete](#), [scale\\_color\\_hue](#), [scale\\_colour\\_discrete](#), [scale\\_colour\\_hue](#), [scale\\_fill\\_discrete](#), [scale\\_fill\\_hue](#); [scale\\_color\\_gradient2](#), [scale\\_colour\\_gradient2](#), [scale\\_fill\\_gradient2](#); [scale\\_color\\_gradientn](#), [scale\\_colour\\_gradientn](#), [scale\\_fill\\_gradientn](#); [scale\\_color\\_grey](#), [scale\\_colour\\_grey](#), [scale\\_fill\\_grey](#)

**Examples**

```
# It's hard to see, but look for the bright yellow dot
# in the bottom right hand corner
dsub <- subset(diamonds, x > 5 & x < 6 & y > 5 & y < 6)
(d <- qplot(x, y, data=dsub, colour=z))
# That one point throws our entire scale off. We could
# remove it, or manually tweak the limits of the scale

# Tweak scale limits. Any points outside these limits will not be
# plotted, and will not affect the calculation of statistics, etc
d + scale_colour_gradient(limits=c(3, 10))
d + scale_colour_gradient(limits=c(3, 4))
# Setting the limits manually is also useful when producing
# multiple plots that need to be comparable

# Alternatively we could try transforming the scale:
d + scale_colour_gradient(trans = "log")
d + scale_colour_gradient(trans = "sqrt")

# Other more trivial manipulations, including changing the name
# of the scale and the colours.

d + scale_colour_gradient("Depth")
d + scale_colour_gradient(expression(Depth[mm]))

d + scale_colour_gradient(limits=c(3, 4), low="red")
d + scale_colour_gradient(limits=c(3, 4), low="red", high="white")
# Much slower
d + scale_colour_gradient(limits=c(3, 4), low="red", high="white", space="Lab")
d + scale_colour_gradient(limits=c(3, 4), space="Lab")

# scale_fill_continuous works similarly, but for fill colours
(h <- qplot(x - y, data=dsub, geom="histogram", binwidth=0.01, fill=..count..))
h + scale_fill_continuous(low="black", high="pink", limits=c(0,3100))

# Colour of missing values is controlled with na.value:
miss <- sample(c(NA, 1:5), nrow(mtcars), rep = TRUE)
qplot(mpg, wt, data = mtcars, colour = miss)
qplot(mpg, wt, data = mtcars, colour = miss) +
  scale_colour_gradient(na.value = "black")
```

---

scale\_colour\_gradient2

*Diverging colour gradient*


---

## Description

Diverging colour gradient

## Usage

```
scale_colour_gradient2(..., low = muted("red"), mid = "white",
  high = muted("blue"), midpoint = 0, space = "rgb",
  na.value = "grey50", guide = "colourbar")
```

```
scale_fill_gradient2(..., low = muted("red"), mid = "white",
  high = muted("blue"), midpoint = 0, space = "rgb",
  na.value = "grey50", guide = "colourbar")
```

```
scale_color_gradient2(..., low = muted("red"), mid = "white",
  high = muted("blue"), midpoint = 0, space = "rgb",
  na.value = "grey50", guide = "colourbar")
```

## Arguments

midpoint	The midpoint (in data value) of the diverging scale. Defaults to 0.
guide	Type of legend. Use "colourbar" for continuous colour bar, or "legend" for discrete colour legend.
...	Other arguments passed on to <a href="#">discrete_scale</a> to control name, limits, breaks, labels and so forth.
na.value	Colour to use for missing values
low	colour for low end of gradient.
mid	colour for mid point
high	colour for high end of gradient.
space	colour space in which to calculate gradient. "Lab" usually best unless gradient goes through white.

## See Also

Other colour scales: [scale\\_color\\_brewer](#), [scale\\_color\\_distiller](#), [scale\\_colour\\_brewer](#), [scale\\_colour\\_distiller](#), [scale\\_fill\\_brewer](#), [scale\\_fill\\_distiller](#); [scale\\_color\\_continuous](#), [scale\\_color\\_gradient](#), [scale\\_colour\\_continuous](#), [scale\\_colour\\_gradient](#), [scale\\_fill\\_continuous](#), [scale\\_fill\\_gradient](#); [scale\\_color\\_discrete](#), [scale\\_color\\_hue](#), [scale\\_colour\\_discrete](#), [scale\\_colour\\_hue](#), [scale\\_fill\\_discrete](#), [scale\\_fill\\_hue](#); [scale\\_color\\_gradientn](#), [scale\\_colour\\_gradientn](#), [scale\\_fill\\_gradientn](#); [scale\\_color\\_grey](#), [scale\\_colour\\_grey](#), [scale\\_fill\\_grey](#)

**Examples**

```

dsub <- subset(diamonds, x > 5 & x < 6 & y > 5 & y < 6)
dsub$diff <- with(dsub, sqrt(abs(x-y))* sign(x-y))
(d <- qplot(x, y, data=dsub, colour=diff))

d + scale_colour_gradient2()
# Change scale name
d + scale_colour_gradient2(expression(sqrt(abs(x - y))))
d + scale_colour_gradient2("Difference\nbetween\nwidth and\nheight")

# Change limits and colours
d + scale_colour_gradient2(limits=c(-0.2, 0.2))

# Using "muted" colours makes for pleasant graphics
# (and they have better perceptual properties too)
library(scales) # for muted
d + scale_colour_gradient2(low="red", high="blue")
d + scale_colour_gradient2(low=muted("red"), high=muted("blue"))

# Using the Lab colour space also improves perceptual properties
# at the price of slightly slower operation
d + scale_colour_gradient2(space="Lab")

# About 5% of males are red-green colour blind, so it's a good
# idea to avoid that combination
d + scale_colour_gradient2(high=muted("green"))

# We can also make the middle stand out
d + scale_colour_gradient2(mid=muted("green"), high="white", low="white")

# or use a non zero mid point
(d <- qplot(carat, price, data=diamonds, colour=price/carat))
d + scale_colour_gradient2(midpoint=mean(diamonds$price / diamonds$carat))

# Fill gradients work much the same way
p <- qplot(letters[1:5], 1:5, fill= c(-3, 3, 5, 2, -2), geom = "bar",
  stat = "identity")
p + scale_fill_gradient2("fill")
# Note how positive and negative values of the same magnitude
# have similar intensity

```

---

scale\_colour\_gradientn

*Smooth colour gradient between n colours*


---

**Description**

Smooth colour gradient between n colours

**Usage**

```
scale_colour_gradientn(..., colours, values = NULL, space = "Lab",
  na.value = "grey50", guide = "colourbar")

scale_fill_gradientn(..., colours, values = NULL, space = "Lab",
  na.value = "grey50", guide = "colourbar")

scale_color_gradientn(..., colours, values = NULL, space = "Lab",
  na.value = "grey50", guide = "colourbar")
```

**Arguments**

guide	Type of legend. Use "colourbar" for continuous colour bar, or "legend" for discrete colour legend.
colours	vector of colours
values	if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the colours vector. See <a href="#">rescale</a> for a convenience function to map an arbitrary range to between 0 and 1.
space	colour space in which to calculate gradient. "Lab" usually best unless gradient goes through white.
...	Other arguments passed on to <a href="#">discrete_scale</a> to control name, limits, breaks, labels and so forth.
na.value	Colour to use for missing values

**See Also**

Other colour scales: [scale\\_color\\_brewer](#), [scale\\_color\\_distiller](#), [scale\\_colour\\_brewer](#), [scale\\_colour\\_distiller](#), [scale\\_fill\\_brewer](#), [scale\\_fill\\_distiller](#); [scale\\_color\\_continuous](#), [scale\\_color\\_gradient](#), [scale\\_colour\\_continuous](#), [scale\\_colour\\_gradient](#), [scale\\_fill\\_continuous](#), [scale\\_fill\\_gradient](#); [scale\\_color\\_discrete](#), [scale\\_color\\_hue](#), [scale\\_colour\\_discrete](#), [scale\\_colour\\_hue](#), [scale\\_fill\\_discrete](#), [scale\\_fill\\_hue](#); [scale\\_color\\_gradient2](#), [scale\\_colour\\_gradient2](#), [scale\\_fill\\_gradient2](#); [scale\\_color\\_grey](#), [scale\\_colour\\_grey](#), [scale\\_fill\\_grey](#)

**Examples**

```
# scale_colour_gradient make it easy to use existing colour palettes

dsub <- subset(diamonds, x > 5 & x < 6 & y > 5 & y < 6)
dsub$diff <- with(dsub, sqrt(abs(x-y))* sign(x-y))
(d <- qplot(x, y, data=dsub, colour=diff))

d + scale_colour_gradientn(colours = rainbow(7))
breaks <- c(-0.5, 0, 0.5)
d + scale_colour_gradientn(colours = rainbow(7),
  breaks = breaks, labels = format(breaks))

d + scale_colour_gradientn(colours = topo.colors(10))
```



```
d + scale_colour_gradientn(colours = terrain.colors(10))

# You can force them to be symmetric by supplying a vector of
# values, and turning rescaling off
max_val <- max(abs(dsub$diff))
values <- seq(-max_val, max_val, length = 11)

d + scale_colour_gradientn(colours = topo.colors(10),
  values = values, rescaler = function(x, ...) x, oob = identity)
d + scale_colour_gradientn(colours = terrain.colors(10),
  values = values, rescaler = function(x, ...) x, oob = identity)
```

---

scale_colour_grey	<i>Sequential grey colour scale.</i>
-------------------	--------------------------------------

---

## Description

Based on [gray.colors](#)

## Usage

```
scale_colour_grey(..., start = 0.2, end = 0.8, na.value = "red")

scale_fill_grey(..., start = 0.2, end = 0.8, na.value = "grey50")

scale_color_grey(..., start = 0.2, end = 0.8, na.value = "red")
```

## Arguments

start	gray value at low end of palette
end	gray value at high end of palette
...	Other arguments passed on to <a href="#">discrete_scale</a> to control name, limits, breaks, labels and so forth.
na.value	Colour to use for missing values

## See Also

Other colour scales: [scale\\_color\\_brewer](#), [scale\\_color\\_distiller](#), [scale\\_colour\\_brewer](#), [scale\\_colour\\_distiller](#), [scale\\_fill\\_brewer](#), [scale\\_fill\\_distiller](#); [scale\\_color\\_continuous](#), [scale\\_color\\_gradient](#), [scale\\_colour\\_continuous](#), [scale\\_colour\\_gradient](#), [scale\\_fill\\_continuous](#), [scale\\_fill\\_gradient](#); [scale\\_color\\_discrete](#), [scale\\_color\\_hue](#), [scale\\_colour\\_discrete](#), [scale\\_colour\\_hue](#), [scale\\_fill\\_discrete](#), [scale\\_fill\\_hue](#); [scale\\_color\\_gradient2](#), [scale\\_colour\\_gradient2](#), [scale\\_fill\\_gradient2](#); [scale\\_color\\_gradientn](#), [scale\\_colour\\_gradientn](#), [scale\\_fill\\_gradientn](#)

**Examples**

```
p <- qplot(mpg, wt, data=mtcars, colour=factor(cyl))
p + scale_colour_grey()
p + scale_colour_grey(end = 0)

# You may want to turn off the pale grey background with this scale
p + scale_colour_grey() + theme_bw()

# Colour of missing values is controlled with na.value:
miss <- factor(sample(c(NA, 1:5), nrow(mtcars), rep = TRUE))
qplot(mpg, wt, data = mtcars, colour = miss) + scale_colour_grey()
qplot(mpg, wt, data = mtcars, colour = miss) +
  scale_colour_grey(na.value = "green")
```

---

scale_colour_hue	<i>Qualitative colour scale with evenly spaced hues.</i>
------------------	----------------------------------------------------------

---

**Description**

Qualitative colour scale with evenly spaced hues.

**Usage**

```
scale_colour_hue(..., h = c(0, 360) + 15, c = 100, l = 65, h.start = 0,
  direction = 1, na.value = "grey50")

scale_fill_hue(..., h = c(0, 360) + 15, c = 100, l = 65, h.start = 0,
  direction = 1, na.value = "grey50")

scale_colour_discrete(..., h = c(0, 360) + 15, c = 100, l = 65,
  h.start = 0, direction = 1, na.value = "grey50")

scale_fill_discrete(..., h = c(0, 360) + 15, c = 100, l = 65,
  h.start = 0, direction = 1, na.value = "grey50")

scale_color_discrete(..., h = c(0, 360) + 15, c = 100, l = 65,
  h.start = 0, direction = 1, na.value = "grey50")

scale_color_hue(..., h = c(0, 360) + 15, c = 100, l = 65, h.start = 0,
  direction = 1, na.value = "grey50")
```

**Arguments**

na.value	Colour to use for missing values
...	Other arguments passed on to <a href="#">discrete_scale</a> to control name, limits, breaks, labels and so forth.
h	range of hues to use, in [0, 360]

c	chroma (intensity of colour), maximum value varies depending on
l	luminance (lightness), in [0, 100]
h.start	hue to start at
direction	direction to travel around the colour wheel, 1 = clockwise, -1 = counter-clockwise

### See Also

Other colour scales: [scale\\_color\\_brewer](#), [scale\\_color\\_distiller](#), [scale\\_colour\\_brewer](#), [scale\\_colour\\_distiller](#), [scale\\_fill\\_brewer](#), [scale\\_fill\\_distiller](#); [scale\\_color\\_continuous](#), [scale\\_color\\_gradient](#), [scale\\_colour\\_continuous](#), [scale\\_colour\\_gradient](#), [scale\\_fill\\_continuous](#), [scale\\_fill\\_gradient](#); [scale\\_color\\_gradient2](#), [scale\\_colour\\_gradient2](#), [scale\\_fill\\_gradient2](#); [scale\\_color\\_gradientn](#), [scale\\_colour\\_gradientn](#), [scale\\_fill\\_gradientn](#); [scale\\_color\\_grey](#), [scale\\_colour\\_grey](#), [scale\\_fill\\_grey](#)

### Examples

```
dsamp <- diamonds[sample(nrow(diamonds), 1000), ]
(d <- qplot(carat, price, data=dsamp, colour=clarity))

# Change scale label
d + scale_colour_hue()
d + scale_colour_hue("clarity")
d + scale_colour_hue(expression(clarity[beta]))

# Adjust luminosity and chroma
d + scale_colour_hue(l=40, c=30)
d + scale_colour_hue(l=70, c=30)
d + scale_colour_hue(l=70, c=150)
d + scale_colour_hue(l=80, c=150)

# Change range of hues used
d + scale_colour_hue(h=c(0, 90))
d + scale_colour_hue(h=c(90, 180))
d + scale_colour_hue(h=c(180, 270))
d + scale_colour_hue(h=c(270, 360))

# Vary opacity
# (only works with pdf, quartz and cairo devices)
d <- ggplot(dsamp, aes(carat, price, colour = clarity))
d + geom_point(alpha = 0.9)
d + geom_point(alpha = 0.5)
d + geom_point(alpha = 0.2)

# Colour of missing values is controlled with na.value:
miss <- factor(sample(c(NA, 1:5), nrow(mtcars), rep = TRUE))
qplot(mpg, wt, data = mtcars, colour = miss)
qplot(mpg, wt, data = mtcars, colour = miss) +
  scale_colour_hue(na.value = "black")
```

---

scale_identity	<i>Use values without scaling.</i>
----------------	------------------------------------

---

## Description

Use values without scaling.

## Usage

```
scale_colour_identity(..., guide = "none")
scale_fill_identity(..., guide = "none")
scale_shape_identity(..., guide = "none")
scale_linetype_identity(..., guide = "none")
scale_alpha_identity(..., guide = "none")
scale_size_identity(..., guide = "none")
scale_color_identity(..., guide = "none")
```

## Arguments

...	Other arguments passed on to <a href="#">discrete_scale</a> or <a href="#">continuous_scale</a>
guide	Guide to use for this scale - defaults to "none".

## Examples

```
colour <- c("red", "green", "blue", "yellow")
qplot(1:4, 1:4, fill = colour, geom = "tile")
qplot(1:4, 1:4, fill = colour, geom = "tile") + scale_fill_identity()

# To get a legend guide, specify guide = "legend"
qplot(1:4, 1:4, fill = colour, geom = "tile") +
  scale_fill_identity(guide = "legend")
# But you'll typically also need to supply breaks and labels:
qplot(1:4, 1:4, fill = colour, geom = "tile") +
  scale_fill_identity("trt", labels = letters[1:4], breaks = colour,
    guide = "legend")

# cyl scaled to appropriate size
qplot(mpg, wt, data = mtcars, size = cyl)

# cyl used as point size
qplot(mpg, wt, data = mtcars, size = cyl) + scale_size_identity()
```

---

scale_linetype	<i>Scale for line patterns.</i>
----------------	---------------------------------

---

### Description

Default line types based on a set supplied by Richard Pearson, University of Manchester. Line types can not be mapped to continuous values.

### Usage

```
scale_linetype(..., na.value = "blank")

scale_linetype_continuous(...)

scale_linetype_discrete(..., na.value = "blank")
```

### Arguments

na.value	The linetype to use for NA values.
...	common discrete scale parameters: name, breaks, labels, na.value, limits and guide. See <a href="#">discrete_scale</a> for more details

### Examples

```
library(reshape2) # for melt
library(plyr) # for ddply
ecm <- melt(economics, id = "date")
rescale01 <- function(x) (x - min(x)) / diff(range(x))
ecm <- ddply(ecm, "variable", transform, value = rescale01(value))

qplot(date, value, data=ecm, geom="line", group=variable)
qplot(date, value, data=ecm, geom="line", linetype=variable)
qplot(date, value, data=ecm, geom="line", colour=variable)

# See scale_manual for more flexibility
```

---

scale_manual	<i>Create your own discrete scale.</i>
--------------	----------------------------------------

---

### Description

Create your own discrete scale.

**Usage**

```

scale_colour_manual(..., values)

scale_fill_manual(..., values)

scale_size_manual(..., values)

scale_shape_manual(..., values)

scale_linetype_manual(..., values)

scale_alpha_manual(..., values)

scale_color_manual(..., values)

```

**Arguments**

<code>values</code>	a set of aesthetic values to map data values to. If this is a named vector, then the values will be matched based on the names. If unnamed, values will be matched in order (usually alphabetical) with the limits of the scale. Any data values that don't match will be given <code>na.value</code> .
<code>...</code>	common discrete scale parameters: <code>name</code> , <code>breaks</code> , <code>labels</code> , <code>na.value</code> , <code>limits</code> and <code>guide</code> . See <a href="#">discrete_scale</a> for more details

**Examples**

```

p <- qplot(mpg, wt, data = mtcars, colour = factor(cyl))

p + scale_colour_manual(values = c("red", "blue", "green"))
p + scale_colour_manual(
  values = c("8" = "red", "4" = "blue", "6" = "green"))
# With rgb hex values
p + scale_colour_manual(values = c("#FF0000", "#0000FF", "#00FF00"))

# As with other scales you can use breaks to control the appearance
# of the legend
cols <- c("8" = "red", "4" = "blue", "6" = "darkgreen", "10" = "orange")
p + scale_colour_manual(values = cols)
p + scale_colour_manual(values = cols, breaks = c("4", "6", "8"))
p + scale_colour_manual(values = cols, breaks = c("8", "6", "4"))
p + scale_colour_manual(values = cols, breaks = c("4", "6", "8"),
  labels = c("four", "six", "eight"))

# And limits to control the possible values of the scale
p + scale_colour_manual(values = cols, limits = c("4", "8"))
p + scale_colour_manual(values = cols, limits = c("4", "6", "8", "10"))

# Notice that the values are matched with limits, and not breaks
p + scale_colour_manual(limits = c(6, 8, 4), breaks = c(8, 4, 6),

```

```
values = c("grey50", "grey80", "black"))
```

---

scale_shape	<i>Scale for shapes, aka glyphs.</i>
-------------	--------------------------------------

---

## Description

A continuous variable can not be mapped to shape.

## Usage

```
scale_shape(..., solid = TRUE)

scale_shape_discrete(..., solid = TRUE)

scale_shape_continuous(...)
```

## Arguments

<code>solid</code>	Are the shapes solid, TRUE, or hollow FALSE?
<code>...</code>	common discrete scale parameters: name, breaks, labels, na.value, limits and guide. See <a href="#">discrete_scale</a> for more details

## Examples

```
dsmall <- diamonds[sample(nrow(diamonds), 100), ]

(d <- qplot(carat, price, data=dsmall, shape=cut))
d + scale_shape(solid = TRUE) # the default
d + scale_shape(solid = FALSE)
d + scale_shape(name="Cut of diamond")
d + scale_shape(name="Cut of\ndiamond")

# To change order of levels, change order of
# underlying factor
levels(dsmall$cut) <- c("Fair", "Good", "Very Good", "Premium", "Ideal")

# Need to recreate plot to pick up new data
qplot(price, carat, data=dsmall, shape=cut)

# Or for short:
d %>% dsmall
```

---

scale\_size

*Size scale.*


---

### Description

Size scale.

### Usage

```
scale_size_continuous(..., range = c(1, 6))
```

```
scale_size(..., range = c(1, 6))
```

```
scale_size_discrete(..., range = c(1, 6))
```

### Arguments

range	a numeric vector of length 2 that specifies the minimum and maximum size of the plotting symbol after transformation.
...	common continuous scale parameters: name, breaks, labels, na.value, limits and trans. See <a href="#">continuous_scale</a> for more details

### Examples

```
(p <- qplot(mpg, cyl, data=mtcars, size=cyl))
p + scale_size("cylinders")
p + scale_size("number\nof\n cylinders")

p + scale_size(range = c(0, 10))
p + scale_size(range = c(1, 2))

# Map area, instead of width/radius
# Perceptually, this is a little better
p + scale_size_area()
p + scale_size_area(max_size = 25)

# Also works with factors, but not a terribly good
# idea, unless your factor is ordered, as in this example
qplot(mpg, cyl, data=mtcars, size=factor(cyl))

# To control the size mapping for discrete variable, use
# scale_size_manual:
last_plot() + scale_size_manual(values=c(2,4,6))
```



---

scale_size_area	<i>Scale area instead of radius, for size.</i>
-----------------	------------------------------------------------

---

### Description

When `scale_size_area` is used, the default behavior is to scale the area of points to be proportional to the value.

### Usage

```
scale_size_area(..., max_size = 6)
```

### Arguments

<code>...</code>	Other arguments passed on to <a href="#">continuous_scale</a> to control name, limits, breaks, labels and so forth.
<code>max_size</code>	Size of largest points.

### Details

Note that this controls the size scale, so it will also control the thickness of lines. Line thickness will be proportional to the square root of the value, which is probably undesirable in most cases.

---

scale_x_continuous	<i>Continuous position scales (x &amp; y).</i>
--------------------	------------------------------------------------

---

### Description

Continuous position scales (x & y).

### Usage

```
scale_x_continuous(..., expand = waiver())
```

```
scale_y_continuous(..., expand = waiver())
```

```
scale_x_log10(...)
```

```
scale_y_log10(...)
```

```
scale_x_reverse(...)
```

```
scale_y_reverse(...)
```

```
scale_x_sqrt(...)
```

```
scale_y_sqrt(...)
```

**Arguments**

... common continuous scale parameters: name, breaks, labels, na.value, limits and trans. See [continuous\\_scale](#) for more details

expand a numeric vector of length two giving multiplicative and additive expansion constants. These constants ensure that the data is placed some distance away from the axes.

**See Also**

Other position scales: [scale\\_x\\_datetime](#), [scale\\_y\\_datetime](#); [scale\\_x\\_date](#), [scale\\_y\\_date](#); [scale\\_x\\_discrete](#), [scale\\_y\\_discrete](#)

**Examples**

```
(m <- qplot(rating, votes, data=subset(movies, votes > 1000),
  na.rm = TRUE))

# Manipulating the default position scales lets you:

# * change the axis labels
m + scale_y_continuous("number of votes")
m + scale_y_continuous(expression(votes^alpha))

# * modify the axis limits
m + scale_y_continuous(limits=c(0, 5000))
m + scale_y_continuous(limits=c(1000, 10000))
m + scale_x_continuous(limits=c(7, 8))

# you can also use the short hand functions xlim and ylim
m + ylim(0, 5000)
m + ylim(1000, 10000)
m + xlim(7, 8)

# * choose where the ticks appear
m + scale_x_continuous(breaks=1:10)
m + scale_x_continuous(breaks=c(1,3,7,9))

# * manually label the ticks
m + scale_x_continuous(breaks=c(2,5,8), labels=c("two", "five", "eight"))
m + scale_x_continuous(breaks=c(2,5,8), labels=c("horrible", "ok", "awesome"))
m + scale_x_continuous(breaks=c(2,5,8), labels=expression(Alpha, Beta, Omega))

# There are a few built in transformation that you can use:
m + scale_y_log10()
m + scale_y_sqrt()
m + scale_y_reverse()
# You can also create your own and supply them to the trans argument.
# See ?scale::trans_new

# You can control the formatting of the labels with the formatter
```

```
# argument. Some common formats are built into the scales package:
x <- rnorm(10) * 100000
y <- seq(0, 1, length = 10)
p <- qplot(x, y)
library(scales)
p + scale_y_continuous(labels = percent)
p + scale_y_continuous(labels = dollar)
p + scale_x_continuous(labels = comma)

# qplot allows you to do some of this with a little less typing:
# * axis limits
qplot(rating, votes, data=movies, ylim=c(1e4, 5e4))
# * axis labels
qplot(rating, votes, data=movies, xlab="My x axis", ylab="My y axis")
# * log scaling
qplot(rating, votes, data=movies, log="xy")
```

---

scale\_x\_date

*Position scale, date*


---

## Description

Position scale, date

## Usage

```
scale_x_date(..., expand = waiver(), breaks = pretty_breaks(),
  minor_breaks = waiver())
```

```
scale_y_date(..., expand = waiver(), breaks = pretty_breaks(),
  minor_breaks = waiver())
```

## Arguments

breaks	A vector of breaks, a function that given the scale limits returns a vector of breaks, or a character vector, specifying the width between breaks. For more information about the first two, see <a href="#">continuous_scale</a> , for more information about the last, see <a href="#">date_breaks</a> .
minor_breaks	Either NULL for no minor breaks, <code>waiver()</code> for the default breaks (one minor break between each major break), a numeric vector of positions, or a function that given the limits returns a vector of minor breaks.
...	common continuous scale parameters: name, breaks, labels, na.value, limits and trans. See <a href="#">continuous_scale</a> for more details
expand	a numeric vector of length two giving multiplicative and additive expansion constants. These constants ensure that the data is placed some distance away from the axes.

**See Also**

Other position scales: [scale\\_x\\_continuous](#), [scale\\_x\\_log10](#), [scale\\_x\\_reverse](#), [scale\\_x\\_sqrt](#), [scale\\_y\\_continuous](#), [scale\\_y\\_log10](#), [scale\\_y\\_reverse](#), [scale\\_y\\_sqrt](#); [scale\\_x\\_datetime](#), [scale\\_y\\_datetime](#); [scale\\_x\\_discrete](#), [scale\\_y\\_discrete](#)

**Examples**

```
# We'll start by creating some nonsense data with dates
df <- data.frame(
  date = seq(Sys.Date(), len=100, by="1 day")[sample(100, 50)],
  price = runif(50)
)
df <- df[order(df$date), ]
dt <- qplot(date, price, data=df, geom="line") + theme(aspect.ratio = 1/4)

# We can control the format of the labels, and the frequency of
# the major and minor tickmarks. See ?format.Date and ?seq.Date
# for more details.
library(scales) # to access breaks/formatting functions
dt + scale_x_date()
dt + scale_x_date(labels = date_format("%m/%d"))
dt + scale_x_date(labels = date_format("%W"))
dt + scale_x_date(labels = date_format("%W"), breaks = date_breaks("week"))

dt + scale_x_date(breaks = date_breaks("months"),
  labels = date_format("%b"))
dt + scale_x_date(breaks = date_breaks("4 weeks"),
  labels = date_format("%d-%b"))

# We can use character string for breaks.
# See \code{\link{by}} argument in \code{\link{seq.Date}}.
dt + scale_x_date(breaks = "2 weeks")
dt + scale_x_date(breaks = "1 month", minor_breaks = "1 week")

# The date scale will attempt to pick sensible defaults for
# major and minor tick marks
qplot(date, price, data=df[1:10,], geom="line")
qplot(date, price, data=df[1:4,], geom="line")

df <- data.frame(
  date = seq(Sys.Date(), len=1000, by="1 day"),
  price = runif(500)
)
qplot(date, price, data=df, geom="line")

# A real example using economic time series data
qplot(date, psavert, data=economics)
qplot(date, psavert, data=economics, geom="path")

end <- max(economics$date)
last_plot() + scale_x_date(limits = c(as.Date("2000-1-1"), end))
last_plot() + scale_x_date(limits = c(as.Date("2005-1-1"), end))
```

```
last_plot() + scale_x_date(limits = c(as.Date("2006-1-1"), end))

# If we want to display multiple series, one for each variable
# it's easiest to first change the data from a "wide" to a "long"
# format:
library(reshape2) # for melt
em <- melt(economics, id = "date")

# Then we can group and facet by the new "variable" variable
qplot(date, value, data = em, geom = "line", group = variable)
qplot(date, value, data = em, geom = "line", group = variable) +
  facet_grid(variable ~ ., scale = "free_y")
```

---

scale_x_datetime	<i>Position scale, date</i>
------------------	-----------------------------

---

## Description

Position scale, date

## Usage

```
scale_x_datetime(..., expand = waiver(), breaks = pretty_breaks(),
  minor_breaks = waiver())
```

```
scale_y_datetime(..., expand = waiver(), breaks = pretty_breaks(),
  minor_breaks = waiver())
```

## Arguments

breaks	A vector of breaks, a function that given the scale limits returns a vector of breaks, or a character vector, specifying the width between breaks. For more information about the first two, see <a href="#">continuous_scale</a> , for more information about the last, see <a href="#">date_breaks</a> .
minor_breaks	Either NULL for no minor breaks, <code>waiver()</code> for the default breaks (one minor break between each major break), a numeric vector of positions, or a function that given the limits returns a vector of minor breaks.
...	common continuous scale parameters: <code>name</code> , <code>breaks</code> , <code>labels</code> , <code>na.value</code> , <code>limits</code> and <code>trans</code> . See <a href="#">continuous_scale</a> for more details
expand	a numeric vector of length two giving multiplicative and additive expansion constants. These constants ensure that the data is placed some distance away from the axes.

## See Also

Other position scales: [scale\\_x\\_continuous](#), [scale\\_x\\_log10](#), [scale\\_x\\_reverse](#), [scale\\_x\\_sqrt](#), [scale\\_y\\_continuous](#), [scale\\_y\\_log10](#), [scale\\_y\\_reverse](#), [scale\\_y\\_sqrt](#); [scale\\_x\\_date](#), [scale\\_y\\_date](#); [scale\\_x\\_discrete](#), [scale\\_y\\_discrete](#)

## Examples

```

start <- ISOdate(2001, 1, 1, tz = "")
df <- data.frame(
  day30 = start + round(runif(100, max = 30 * 86400)),
  day7 = start + round(runif(100, max = 7 * 86400)),
  day = start + round(runif(100, max = 86400)),
  hour10 = start + round(runif(100, max = 10 * 3600)),
  hour5 = start + round(runif(100, max = 5 * 3600)),
  hour = start + round(runif(100, max = 3600)),
  min10 = start + round(runif(100, max = 10 * 60)),
  min5 = start + round(runif(100, max = 5 * 60)),
  min = start + round(runif(100, max = 60)),
  sec10 = start + round(runif(100, max = 10)),
  y = runif(100)
)

# Automatic scale selection
qplot(sec10, y, data = df)
qplot(min, y, data = df)
qplot(min5, y, data = df)
qplot(min10, y, data = df)
qplot(hour, y, data = df)
qplot(hour5, y, data = df)
qplot(hour10, y, data = df)
qplot(day, y, data = df)
qplot(day30, y, data = df)

# Manual scale selection
qplot(day30, y, data = df)
library(scales) # to access breaks/formatting functions
last_plot() + scale_x_datetime(breaks = date_breaks("2 weeks"))
last_plot() + scale_x_datetime(breaks = date_breaks("10 days"))
library(scales) # to access breaks/formatting functions
last_plot() + scale_x_datetime(breaks = date_breaks("10 days"),
  labels = date_format("%d/%m"))
last_plot() + scale_x_datetime(breaks = date_breaks("1 day"),
  minor_breaks = date_breaks("2 hour"))

```

---

scale\_x\_discrete

*Discrete position.*

---

## Description

You can use continuous positions even with a discrete position scale - this allows you (e.g.) to place labels between bars in a bar chart. Continuous positions are numeric values starting at one for the first level, and increasing by one for each level (i.e. the labels are placed at integer positions). This is what allows jittering to work.

**Usage**

```
scale_x_discrete(..., expand = waiver())
```

```
scale_y_discrete(..., expand = waiver())
```

**Arguments**

... common discrete scale parameters: name, breaks, labels, na.value, limits and guide. See [discrete\\_scale](#) for more details

expand a numeric vector of length two giving multiplicative and additive expansion constants. These constants ensure that the data is placed some distance away from the axes.

**See Also**

Other position scales: [scale\\_x\\_continuous](#), [scale\\_x\\_log10](#), [scale\\_x\\_reverse](#), [scale\\_x\\_sqrt](#), [scale\\_y\\_continuous](#), [scale\\_y\\_log10](#), [scale\\_y\\_reverse](#), [scale\\_y\\_sqrt](#); [scale\\_x\\_datetime](#), [scale\\_y\\_datetime](#); [scale\\_x\\_date](#), [scale\\_y\\_date](#)

**Examples**

```
qplot(cut, data=diamonds, stat="bin")
qplot(cut, data=diamonds, geom="bar")

# The discrete position scale is added automatically whenever you
# have a discrete position.

(d <- qplot(cut, clarity, data=subset(diamonds, carat > 1), geom="jitter"))

d + scale_x_discrete("Cut")
d + scale_x_discrete("Cut", labels = c("Fair" = "F", "Good" = "G",
  "Very Good" = "VG", "Perfect" = "P", "Ideal" = "I"))

d + scale_y_discrete("Clarity")
d + scale_x_discrete("Cut") + scale_y_discrete("Clarity")

# Use limits to adjust the which levels (and in what order)
# are displayed
d + scale_x_discrete(limits=c("Fair", "Ideal"))

# you can also use the short hand functions xlim and ylim
d + xlim("Fair", "Ideal", "Good")
d + ylim("I1", "IF")

# See ?reorder to reorder based on the values of another variable
qplot(manufacturer, cty, data=mpg)
qplot(reorder(manufacturer, cty), cty, data=mpg)
qplot(reorder(manufacturer, displ), cty, data=mpg)

# Use abbreviate as a formatter to reduce long names
```

```
qplot(reorder(manufacturer, cty), cty, data=mpg) +
  scale_x_discrete(labels = abbreviate)
```

---

seals

*Vector field of seal movements.*


---

### Description

This vector field was produced from the data described in Brillinger, D.R., Preisler, H.K., Ager, A.A. and Kie, J.G. "An exploratory data analysis (EDA) of the paths of moving animals". J. Statistical Planning and Inference 122 (2004), 43-63, using the methods of Brillinger, D.R., "Learning a potential function from a trajectory", Signal Processing Letters. December (2007).

### Usage

```
data(seals)
```

### Format

A data frame with 1155 rows and 4 variables

### References

<http://www.stat.berkeley.edu/~brill/Papers/jspifinal.pdf>

---

stat\_bin

*Bin data.*


---

### Description

Missing values are currently silently dropped.

### Usage

```
stat_bin(mapping = NULL, data = NULL, geom = "bar", position = "stack",
  width = 0.9, drop = FALSE, right = FALSE, binwidth = NULL,
  origin = NULL, breaks = NULL, ...)
```



**Arguments**

binwidth	Bin width to use. Defaults to 1/30 of the range of the data
breaks	Actual breaks to use. Overrides bin width and origin
origin	Origin of first bin
width	Width of bars when used with categorical data
right	If TRUE, right-closed, left-open, if FALSE, the default, right-open, left-closed.
drop	If TRUE, remove all bins with zero counts
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Value**

New data frame with additional columns:

count	number of points in bin
density	density of points in bin, scaled to integrate to 1
ncount	count, scaled to maximum of 1
ndensity	density, scaled to maximum of 1

**Aesthetics**

stat\_bin understands the following aesthetics (required aesthetics are in bold):

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

**Examples**

```
simple <- data.frame(x = rep(1:10, each = 2))
base <- ggplot(simple, aes(x))
# By default, right = FALSE intervals are of the form [a, b)
base + stat_bin(binwidth = 1, drop = FALSE, right = FALSE, col = "black")
# If right = TRUE, and intervals are of the form (a, b]
base + stat_bin(binwidth = 1, drop = FALSE, right = TRUE, col = "black")

m <- ggplot(movies, aes(x=rating))
m + stat_bin()
m + stat_bin(binwidth=0.1)
m + stat_bin(breaks=seq(4,6, by=0.1))
# See geom_histogram for more histogram examples
```

```
# To create a unit area histogram, use aes(y = ..density..)
(linehist <- m + stat_bin(aes(y = ..density..), binwidth=0.1,
  geom="line", position="identity"))
linehist + stat_density(colour="blue", fill=NA)

# Also works with categorical variables
ggplot(movies, aes(x=mpaa)) + stat_bin()
qplot(mpaa, data=movies, stat="bin")
```

stat\_bin2d

*Count number of observation in rectangular bins.***Description**

Count number of observation in rectangular bins.

**Usage**

```
stat_bin2d(mapping = NULL, data = NULL, geom = NULL,
  position = "identity", bins = 30, drop = TRUE, ...)
```

**Arguments**

<b>bins</b>	numeric vector giving number of bins in both vertical and horizontal directions. Set to 30 by default.
<b>drop</b>	if TRUE removes all cells with 0 counts.
<b>mapping</b>	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
<b>data</b>	A layer specific dataset - only needed if you want to override the plot defaults.
<b>geom</b>	The geometric object to use display the data
<b>position</b>	The position adjustment to use for overlapping points on this layer
<b>...</b>	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Aesthetics**

stat\_bin2d understands the following aesthetics (required aesthetics are in bold):

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

**See Also**

[stat\\_binhex](#) for hexagonal binning

## Examples

```
d <- ggplot(diamonds, aes(carat, price))
d + stat_bin2d()
d + geom_bin2d()

# You can control the size of the bins by specifying the number of
# bins in each direction:
d + stat_bin2d(bins = 10)
d + stat_bin2d(bins = 30)

# Or by specifying the width of the bins
d + stat_bin2d(binwidth = c(1, 1000))
d + stat_bin2d(binwidth = c(.1, 500))

# Or with a list of breaks
x <- seq(min(diamonds$carat), max(diamonds$carat), by = 0.1)
y <- seq(min(diamonds$price), max(diamonds$price), length = 50)
d + stat_bin2d(breaks = list(x = x, y = y))

# With qplot
qplot(x, y, data = diamonds, geom="bin2d",
      xlim = c(4, 10), ylim = c(4, 10))
qplot(x, y, data = diamonds, geom="bin2d", binwidth = c(0.1, 0.1),
      xlim = c(4, 10), ylim = c(4, 10))
```

---

stat\_bindot

*Bin data for dot plot.*


---

## Description

Missing values are currently silently dropped. If weights are used, they must be integer values.

## Usage

```
stat_bindot(mapping = NULL, data = NULL, geom = "dotplot",
  position = "identity", binwidth = NULL, origin = NULL, width = 0.9,
  binaxis = "x", method = "dotdensity", binpositions = "bygroup",
  drop = FALSE, right = TRUE, na.rm = FALSE, ...)
```

## Arguments

binaxis	The axis to bin along, "x" (default) or "y"
method	"dotdensity" (default) for dot-density binning, or "histodot" for fixed bin widths (like stat_bin)
binwidth	When method is "dotdensity", this specifies maximum bin width. When method is "histodot", this specifies bin width. Defaults to 1/30 of the range of the data

binpositions	When method is "dotdensity", "bygroup" (default) determines positions of the bins for each group separately. "all" determines positions of the bins with all the data taken together; this is used for aligning dot stacks across multiple groups.
origin	When method is "histodot", origin of first bin
right	When method is "histodot", should intervals be closed on the right (a, b], or not [a, b)
width	When binaxis is "y", the spacing of the dot stacks for dodging.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
drop	If TRUE, remove all bins with zero counts
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Value

New data frame with additional columns:

x	center of each bin, if binaxis is "x"
y	center of each bin, if binaxis is "x"
binwidth	max width of each bin if method is "dotdensity"; width of each bin if method is "histodot"
count	number of points in bin
ncount	count, scaled to maximum of 1
density	density of points in bin, scaled to integrate to 1, if method is "histodot"
ndensity	density, scaled to maximum of 1, if method is "histodot"

## Aesthetics

stat\_bindot understands the following aesthetics (required aesthetics are in bold):

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

## See Also

See [geom\\_dotplot](#) for examples.

## Examples

```
# See geom_dotplot for examples
```

---

stat_binhex	<i>Bin 2d plane into hexagons.</i>
-------------	------------------------------------

---

### Description

Bin 2d plane into hexagons.

### Usage

```
stat_binhex(mapping = NULL, data = NULL, geom = "hex",  
            position = "identity", bins = 30, na.rm = FALSE, ...)
```

### Arguments

bins	numeric vector specifying number of bins in both x and y directions. Set to 30 by default.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

### Aesthetics

stat\_binhex understands the following aesthetics (required aesthetics are in bold):

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

### See Also

[stat\\_bin2d](#) for rectangular binning

### Examples

```
d <- ggplot(diamonds, aes(carat, price))  
d + stat_binhex()  
d + geom_hex()  
  
# You can control the size of the bins by specifying the number of
```

```
# bins in each direction:
d + stat_binhex(bins = 10)
d + stat_binhex(bins = 30)

# Or by specifying the width of the bins
d + stat_binhex(binwidth = c(1, 1000))
d + stat_binhex(binwidth = c(.1, 500))

# With qplot
qplot(x, y, data = diamonds, geom="hex", xlim = c(4, 10), ylim = c(4, 10))
qplot(x, y, data = diamonds, geom="hex", xlim = c(4, 10), ylim = c(4, 10),
      binwidth = c(0.1, 0.1))
```

stat\_boxplot

*Calculate components of box and whisker plot.***Description**

Calculate components of box and whisker plot.

**Usage**

```
stat_boxplot(mapping = NULL, data = NULL, geom = "boxplot",
             position = "dodge", na.rm = FALSE, coef = 1.5, ...)
```

**Arguments**

coef	length of the whiskers as multiple of IQR. Defaults to 1.5
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Value**

A data frame with additional columns:

width	width of boxplot
ymin	lower whisker = smallest observation greater than or equal to lower hinge - 1.5 * IQR

lower	lower hinge, 25% quantile
notchlower	lower edge of notch = median - 1.58 * IQR / sqrt(n)
middle	median, 50% quantile
notchupper	upper edge of notch = median + 1.58 * IQR / sqrt(n)
upper	upper hinge, 75% quantile
ymin	upper whisker = largest observation less than or equal to upper hinge + 1.5 * IQR

### Aesthetics

stat\_boxplot understands the following aesthetics (required aesthetics are in bold):

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

### See Also

See [geom\\_boxplot](#) for examples.

### Examples

```
# See geom_boxplot for examples
```

---

stat_contour	<i>Calculate contours of 3d data.</i>
--------------	---------------------------------------

---

### Description

Calculate contours of 3d data.

### Usage

```
stat_contour(mapping = NULL, data = NULL, geom = "path",
  position = "identity", na.rm = FALSE, ...)
```

### Arguments

na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Value**

A data frame with additional column:

level	height of contour
-------	-------------------

**Aesthetics**

stat\_contour understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **z**
- order

**Examples**

```
# Generate data
library(reshape2) # for melt
volcano3d <- melt(volcano)
names(volcano3d) <- c("x", "y", "z")

# Basic plot
v <- ggplot(volcano3d, aes(x, y, z = z))
v + stat_contour()

# Setting bins creates evenly spaced contours in the range of the data
v + stat_contour(bins = 2)
v + stat_contour(bins = 10)

# Setting binwidth does the same thing, parameterised by the distance
# between contours
v + stat_contour(binwidth = 2)
v + stat_contour(binwidth = 5)
v + stat_contour(binwidth = 10)
v + stat_contour(binwidth = 2, size = 0.5, colour = "grey50") +
  stat_contour(binwidth = 10, size = 1)

# Add aesthetic mappings
v + stat_contour(aes(size = ..level..))
v + stat_contour(aes(colour = ..level..))

# Change scale
v + stat_contour(aes(colour = ..level..), size = 2) +
  scale_colour_gradient(low = "brown", high = "white")

# Set aesthetics to fixed value
v + stat_contour(colour = "red")
v + stat_contour(size = 2, linetype = 4)

# Try different geoms
```



```
v + stat_contour(geom="polygon", aes(fill=..level..))
v + geom_tile(aes(fill = z)) + stat_contour()

# Use qplot instead
qplot(x, y, z = z, data = volcano3d, geom = "contour")
qplot(x, y, z = z, data = volcano3d, stat = "contour", geom = "path")
```

---

stat_density	<i>1d kernel density estimate.</i>
--------------	------------------------------------

---

## Description

1d kernel density estimate.

## Usage

```
stat_density(mapping = NULL, data = NULL, geom = "area",
  position = "stack", adjust = 1, kernel = "gaussian", trim = FALSE,
  na.rm = FALSE, ...)
```

## Arguments

adjust	see <a href="#">density</a> for details
kernel	kernel used for density estimation, see <a href="#">density</a> for details
trim	if TRUE, the default, densities are trimmed to the actual range of the data. If FALSE, they are extended by the default 3 bandwidths (as specified by the cut parameter to <a href="#">density</a> )
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Value

data.frame with additional columns:

density	density estimate
count	density * number of points - useful for stacked density plots
scaled	density estimate, scaled to maximum of 1

## Aesthetics

stat\_density understands the following aesthetics (required aesthetics are in bold):

- **x**
- fill
- y

## See Also

[stat\\_bin](#) for the histogram

## Examples

```
m <- ggplot(movies, aes(x = rating))
m + geom_density()

# Adjust parameters
m + geom_density(kernel = "rectangular")
m + geom_density(kernel = "biweight")
m + geom_density(kernel = "epanechnikov")
m + geom_density(adjust=1/5) # Very rough
m + geom_density(adjust=5) # Very smooth

# Adjust aesthetics
m + geom_density(aes(fill=factor(Drama)), size=2)
# Scale so peaks have same height:
m + geom_density(aes(fill=factor(Drama), y = ..scaled..), size=2)

m + geom_density(colour="darkgreen", size=2)
m + geom_density(colour="darkgreen", size=2, fill=NA)
m + geom_density(colour="darkgreen", size=2, fill="green")

# Change scales
(m <- ggplot(movies, aes(x=votes)) + geom_density(trim = TRUE))
m + scale_x_log10()
m + coord_trans(x="log10")
m + scale_x_log10() + coord_trans(x="log10")

# Also useful with
m + stat_bin()

# Make a volcano plot
ggplot(diamonds, aes(x = price)) +
  stat_density(aes(ymax = ..density.., ymin = -..density..),
    fill = "grey50", colour = "grey50",
    geom = "ribbon", position = "identity") +
  facet_grid(. ~ cut) +
  coord_flip()

# Stacked density plots
```

```

# If you want to create a stacked density plot, you need to use
# the 'count' (density * n) variable instead of the default density

# Loses marginal densities
qplot(rating, ..density.., data=movies, geom="density", fill=mpaa, position="stack")
# Preserves marginal densities
qplot(rating, ..count.., data=movies, geom="density", fill=mpaa, position="stack")

# You can use position="fill" to produce a conditional density estimate
qplot(rating, ..count.., data=movies, geom="density", fill=mpaa, position="fill")

# Need to be careful with weighted data
m <- ggplot(movies, aes(x=rating, weight=votes))
m + geom_histogram(aes(y = ..count..)) + geom_density(fill=NA)

m <- ggplot(movies, aes(x=rating, weight=votes/sum(votes)))
m + geom_histogram(aes(y=..density..)) + geom_density(fill=NA, colour="black")

library(plyr) # to access round_any
movies$decade <- round_any(movies$year, 10)
m <- ggplot(movies, aes(x=rating, colour=decade, group=decade))
m + geom_density(fill=NA)
m + geom_density(fill=NA) + aes(y = ..count..)

# Use qplot instead
qplot(length, data=movies, geom="density", weight=rating)
qplot(length, data=movies, geom="density", weight=rating/sum(rating))

```

---

stat_density2d	<i>2d density estimation.</i>
----------------	-------------------------------

---

## Description

2d density estimation.

## Usage

```
stat_density2d(mapping = NULL, data = NULL, geom = "density2d",
  position = "identity", na.rm = FALSE, contour = TRUE, n = 100, ...)
```

## Arguments

contour	If TRUE, contour the results of the 2d density estimation
n	number of grid points in each direction
...	other arguments passed on to <a href="#">kde2d</a>
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

mapping	The aesthetic mapping, usually constructed with <code>aes</code> or <code>aes_string</code> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer

### Value

A data frame in the same format as `stat_contour`

### Aesthetics

`stat_density2d` understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- colour
- size

### Examples

```
library("MASS")
data(geyser, "MASS")

m <- ggplot(geyser, aes(x = duration, y = waiting)) +
  geom_point() + xlim(0.5, 6) + ylim(40, 110)
m + geom_density2d()

dens <- kde2d(geyser$duration, geyser$waiting, n = 50,
  lims = c(0.5, 6, 40, 110))
densdf <- data.frame(expand.grid(duration = dens$x, waiting = dens$y),
  z = as.vector(dens$z))
m + geom_contour(aes(z=z), data=densdf)

m + geom_density2d() + scale_y_log10()
m + geom_density2d() + coord_trans(y="log10")

m + stat_density2d(aes(fill = ..level..), geom="polygon")

qplot(duration, waiting, data=geyser, geom=c("point", "density2d")) +
  xlim(0.5, 6) + ylim(40, 110)

# If you map an aesthetic to a categorical variable, you will get a
# set of contours for each value of that variable
set.seed(4393)
dsmall <- diamonds[sample(nrow(diamonds), 1000), ]
qplot(x, y, data = dsmall, geom = "density2d", colour = cut)
qplot(x, y, data = dsmall, geom = "density2d", linetype = cut)
qplot(carat, price, data = dsmall, geom = "density2d", colour = cut)
```

```
d <- ggplot(dsmall, aes(carat, price)) + xlim(1,3)
d + geom_point() + geom_density2d()

# If we turn contouring off, we can use use geoms like tiles:
d + stat_density2d(geom="tile", aes(fill = ..density..), contour = FALSE)
last_plot() + scale_fill_gradient(limits=c(1e-5,8e-4))

# Or points:
d + stat_density2d(geom="point", aes(size = ..density..), contour = FALSE)
```

stat\_ecdf

*Empirical Cumulative Density Function***Description**

Empirical Cumulative Density Function

**Usage**

```
stat_ecdf(mapping = NULL, data = NULL, geom = "step",
  position = "identity", n = NULL, ...)
```

**Arguments**

n	if NULL, do not interpolate. If not NULL, this is the number of points to interpolate with.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Value**

a data.frame with additional columns:

x	x in data
y	cumulative density corresponding x

Examples

```
qplot(rnorm(1000), stat = "ecdf", geom = "step")

df <- data.frame(x = c(rnorm(100, 0, 3), rnorm(100, 0, 10)),
                 g = gl(2, 100))

ggplot(df, aes(x, colour = g)) + stat_ecdf()
```

---

stat_ellipse	<i>Plot data ellipses.</i>
--------------	----------------------------

---

Description

Plot data ellipses.

Usage

```
stat_ellipse(mapping = NULL, data = NULL, geom = "path",
             position = "identity", type = "t", level = 0.95, segments = 51,
             na.rm = FALSE, ...)
```

Arguments

level	The confidence level at which to draw an ellipse (default is 0.95), or, if type="euclid", the radius of the circle to be drawn.
type	The type of ellipse. The default "t" assumes a multivariate t-distribution, and "norm" assumes a multivariate normal distribution. "euclid" draws a circle with the radius equal to level, representing the euclidian distance from the center. This ellipse probably won't appear circular unless coord_fixed() is applied.
segments	The number of segments to be used in drawing the ellipse.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Details

The method for calculating the ellipses has been modified from `car::ellipse` (Fox and Weisberg, 2011)

## References

John Fox and Sanford Weisberg (2011). An R Companion to Applied Regression, Second Edition. Thousand Oaks CA: Sage. URL: <http://socserv.socsci.mcmaster.ca/jfox/Books/Companion>

## Examples

```
ggplot(faithful, aes(waiting, eruptions))+  
  geom_point()+  
  stat_ellipse()  
  
ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3))+  
  geom_point()+  
  stat_ellipse()  
  
ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3))+  
  geom_point()+  
  stat_ellipse(type = "norm", linetype = 2)+  
  stat_ellipse(type = "t")  
  
ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3))+  
  geom_point()+  
  stat_ellipse(type = "norm", linetype = 2)+  
  stat_ellipse(type = "euclid", level = 3)+  
  coord_fixed()  
  
ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3))+  
  stat_ellipse(geom = "polygon")
```

---

stat_function	<i>Superimpose a function.</i>
---------------	--------------------------------

---

## Description

Superimpose a function.

## Usage

```
stat_function(mapping = NULL, data = NULL, geom = "path",  
  position = "identity", fun, n = 101, args = list(), ...)
```

**Arguments**

fun	function to use
n	number of points to interpolate along
args	list of additional arguments to pass to fun
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Value**

a data.frame with additional columns:

x	x's along a grid
y	value of function evaluated at corresponding x

**Aesthetics**

stat\_function understands the following aesthetics (required aesthetics are in bold):

- y

**Examples**

```
x <- rnorm(100)
base <- qplot(x, geom = "density")
base + stat_function(fun = dnorm, colour = "red")
base + stat_function(fun = dnorm, colour = "red", arg = list(mean = 3))

# Plot functions without data
# Examples adapted from Kokske Takahashi

# Specify range of x-axis
qplot(c(0, 2), stat = "function", fun = exp, geom = "line")
ggplot(data.frame(x = c(0, 2)), aes(x)) + stat_function(fun = exp)
# Plot a normal curve
ggplot(data.frame(x = c(-5, 5)), aes(x)) + stat_function(fun = dnorm)
# With qplot
qplot(c(-5, 5), stat = "function", fun = dnorm, geom = "line")
# Or
qplot(c(-5, 5), geom = "blank") + stat_function(fun = dnorm)
# To specify a different mean or sd, use the args parameter to supply new values
ggplot(data.frame(x = c(-5, 5)), aes(x)) +
  stat_function(fun = dnorm, args = list(mean = 2, sd = .5))
```



```
# Two functions on the same plot
f <- ggplot(data.frame(x = c(0, 10)), aes(x))
f + stat_function(fun = sin, colour = "red") +
  stat_function(fun = cos, colour = "blue")

# Using a custom function
test <- function(x) {x ^ 2 + x + 20}
f + stat_function(fun = test)
```

---

stat_identity	<i>Identity statistic.</i>
---------------	----------------------------

---

## Description

Identity statistic.

## Usage

```
stat_identity(mapping = NULL, data = NULL, geom = "point",
  position = "identity", width = NULL, height = NULL, ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
width	The width of the tiles.
height	The height of the tiles.
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Aesthetics

stat\_identity understands the following aesthetics (required aesthetics are in bold):

- 

## Examples

```
# Doesn't do anything, so hard to come up a useful example
```

stat\_qq

*Calculation for quantile-quantile plot.***Description**

Calculation for quantile-quantile plot.

**Usage**

```
stat_qq(mapping = NULL, data = NULL, geom = "point",
        position = "identity", distribution = qnorm, dparams = list(),
        na.rm = FALSE, ...)
```

**Arguments**

distribution	Distribution function to use, if x not specified
dparams	Parameters for distribution function
...	Other arguments passed to distribution function
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer

**Value**

a data.frame with additional columns:

sample	sample quantiles
theoretical	theoretical quantiles

**Aesthetics**

stat\_qq understands the following aesthetics (required aesthetics are in bold):

- **sample**
- x
- y

## Examples

```
# From ?qqplot
y <- rt(200, df = 5)
qqplot(sample = y, stat="qq")

# qqplot is smart enough to use stat_qq if you use sample
qqplot(sample = y)
qqplot(sample = precip)

qqplot(sample = y, dist = qt, dparams = list(df = 5))

df <- data.frame(y)
ggplot(df, aes(sample = y)) + stat_qq()
ggplot(df, aes(sample = y)) + geom_point(stat = "qq")

# Use fitdistr from MASS to estimate distribution params
library(MASS)
params <- as.list(fitdistr(y, "t")$estimate)
ggplot(df, aes(sample = y)) + stat_qq(dist = qt, dparam = params)

# Using to explore the distribution of a variable
qqplot(sample = mpg, data = mtcars)
qqplot(sample = mpg, data = mtcars, colour = factor(cyl))
```

---

stat_quantile	<i>Continuous quantiles.</i>
---------------	------------------------------

---

## Description

Continuous quantiles.

## Usage

```
stat_quantile(mapping = NULL, data = NULL, geom = "quantile",
  position = "identity", quantiles = c(0.25, 0.5, 0.75), formula = NULL,
  method = "rq", na.rm = FALSE, ...)
```

## Arguments

quantiles	conditional quantiles of y to calculate and display
formula	formula relating y variables to x variables
method	Quantile regression method to use. Currently only supports <a href="#">rq</a> .
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.

<code>data</code>	A layer specific dataset - only needed if you want to override the plot defaults.
<code>geom</code>	The geometric object to use display the data
<code>position</code>	The position adjustment to use for overlapping points on this layer
<code>...</code>	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Value**

a data.frame with additional columns:

<code>quantile</code>	quantile of distribution
-----------------------	--------------------------

**Aesthetics**

`stat_quantile` understands the following aesthetics (required aesthetics are in bold):

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

**Examples**

```
msamp <- movies[sample(nrow(movies), 1000), ]
m <- ggplot(msamp, aes(year, rating)) + geom_point()
m + stat_quantile()
m + stat_quantile(quantiles = 0.5)
q10 <- seq(0.05, 0.95, by=0.05)
m + stat_quantile(quantiles = q10)

# You can also use rqss to fit smooth quantiles
m + stat_quantile(method = "rqss")
# Note that rqss doesn't pick a smoothing constant automatically, so
# you'll need to tweak lambda yourself
m + stat_quantile(method = "rqss", lambda = 10)
m + stat_quantile(method = "rqss", lambda = 100)

# Use 'votes' as weights for the quantile calculation
m + stat_quantile(aes(weight=votes))

# Change scale
m + stat_quantile(aes(colour = ..quantile..), quantiles = q10)
m + stat_quantile(aes(colour = ..quantile..), quantiles = q10) +
  scale_colour_gradient2(midpoint = 0.5)

# Set aesthetics to fixed value
m + stat_quantile(colour = "red", size = 2, linetype = 2)

# Use qplot instead
qplot(year, rating, data=movies, geom="quantile")
```

stat\_smooth

*Add a smoother.***Description**

Aids the eye in seeing patterns in the presence of overplotting.

**Usage**

```
stat_smooth(mapping = NULL, data = NULL, geom = "smooth",
  position = "identity", method = "auto", formula = y ~ x, se = TRUE,
  n = 80, fullrange = FALSE, level = 0.95, na.rm = FALSE, ...)
```

**Arguments**

method	smoothing method (function) to use, eg. lm, glm, gam, loess, rlm. For datasets with $n < 1000$ default is <a href="#">loess</a> . For datasets with 1000 or more observations defaults to gam, see <a href="#">gam</a> for more details.
formula	formula to use in smoothing function, eg. $y \sim x$ , $y \sim \text{poly}(x, 2)$ , $y \sim \log(x)$
se	display confidence interval around smooth? (TRUE by default, see level to control)
fullrange	should the fit span the full range of the plot, or just the data
level	level of confidence interval to use (0.95 by default)
n	number of points to evaluate smoother at
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
...	other arguments are passed to smoothing function
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer

**Details**

Calculation is performed by the (currently undocumented) `predictdf` generic function and its methods. For most methods the confidence bounds are computed using the [predict](#) method - the exceptions are `loess` which uses a t-based approximation, and for `glm` where the normal confidence interval is constructed on the link scale, and then back-transformed to the response scale.

**Value**

a data.frame with additional columns

y	predicted value
ymin	lower pointwise confidence interval around the mean
ymax	upper pointwise confidence interval around the mean
se	standard error

**Aesthetics**

stat\_smooth understands the following aesthetics (required aesthetics are in bold):

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

**See Also**

[lm](#) for linear smooths, [glm](#) for generalised linear smooths, [loess](#) for local smooths

**Examples**

```
c <- ggplot(mtcars, aes(qsec, wt))
c + stat_smooth()
c + stat_smooth() + geom_point()

# Adjust parameters
c + stat_smooth(se = FALSE) + geom_point()

c + stat_smooth(span = 0.9) + geom_point()
c + stat_smooth(level = 0.99) + geom_point()
c + stat_smooth(method = "lm") + geom_point()

library(splines)
library(MASS)
c + stat_smooth(method = "lm", formula = y ~ ns(x,3)) +
  geom_point()
c + stat_smooth(method = rlm, formula= y ~ ns(x,3)) + geom_point()

# The default confidence band uses a transparent colour.
# This currently only works on a limited number of graphics devices
# (including Quartz, PDF, and Cairo) so you may need to set the
# fill colour to a opaque colour, as shown below
c + stat_smooth(fill = "grey50", size = 2, alpha = 1)
c + stat_smooth(fill = "blue", size = 2, alpha = 1)

# The colour of the line can be controlled with the colour aesthetic
c + stat_smooth(fill="blue", colour="darkblue", size=2)
c + stat_smooth(fill="blue", colour="darkblue", size=2, alpha = 0.2)
c + geom_point() +
```

```

stat_smooth(fill="blue", colour="darkblue", size=2, alpha = 0.2)

# Smoothers for subsets
c <- ggplot(mtcars, aes(y=wt, x=mpg)) + facet_grid(. ~ cyl)
c + stat_smooth(method=lm) + geom_point()
c + stat_smooth(method=lm, fullrange = TRUE) + geom_point()

# Geoms and stats are automatically split by aesthetics that are factors
c <- ggplot(mtcars, aes(y=wt, x=mpg, colour=factor(cyl)))
c + stat_smooth(method=lm) + geom_point()
c + stat_smooth(method=lm, aes(fill = factor(cyl))) + geom_point()
c + stat_smooth(method=lm, fullrange=TRUE, alpha = 0.1) + geom_point()

# Use qplot instead
qplot(qsec, wt, data=mtcars, geom=c("smooth", "point"))

# Example with logistic regression
data("kyphosis", package="rpart")
qplot(Age, Kyphosis, data=kyphosis)
qplot(Age, data=kyphosis, facets = . ~ Kyphosis, binwidth = 10)
qplot(Age, Kyphosis, data=kyphosis, position="jitter")
qplot(Age, Kyphosis, data=kyphosis, position=position_jitter(height=0.1))

qplot(Age, as.numeric(Kyphosis) - 1, data = kyphosis) +
  stat_smooth(method="glm", family="binomial")
qplot(Age, as.numeric(Kyphosis) - 1, data=kyphosis) +
  stat_smooth(method="glm", family="binomial", formula = y ~ ns(x, 2))

```

stat\_spoke

*Convert angle and radius to xend and yend.*

## Description

Convert angle and radius to xend and yend.

## Usage

```
stat_spoke(mapping = NULL, data = NULL, geom = "segment",
           position = "identity", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Value**

a data.frame with additional columns

xend                    x position of end of line segment

yend                    x position of end of line segment

**Aesthetics**

stat\_spoke understands the following aesthetics (required aesthetics are in bold):

- **angle**
- **radius**
- **x**
- **y**
- xend
- yend

**Examples**

```
df <- expand.grid(x = 1:10, y=1:10)
df$angle <- runif(100, 0, 2*pi)
df$speed <- runif(100, 0, 0.5)

qplot(x, y, data=df) + stat_spoke(aes(angle=angle), radius = 0.5)
last_plot() + scale_y_reverse()

qplot(x, y, data=df) + stat_spoke(aes(angle=angle, radius=speed))
```

---

stat_sum	<i>Sum unique values. Useful for overplotting on scatterplots.</i>
----------	--------------------------------------------------------------------

---

**Description**

Sum unique values. Useful for overplotting on scatterplots.

**Usage**

```
stat_sum(mapping = NULL, data = NULL, geom = "point",
          position = "identity", ...)
```



**Arguments**

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

**Value**

a data.frame with additional columns	
n	number of observations at position
prop	percent of points in that panel at that position

**Aesthetics**

stat\_sum understands the following aesthetics (required aesthetics are in bold):

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

**See Also**

[ggfluctuation](#) for a fluctuation diagram,

**Examples**

```
d <- ggplot(diamonds, aes(x = cut, y = clarity))
# By default, all categorical variables in the plot form grouping
# variables, and the default behavior in stat_sum is to show the
# proportion. Specifying stat_sum with no group identifier leads to
# a plot which is not meaningful:
d + stat_sum()
# To correct this problem and achieve a more desirable plot, we need
# to specify which group the proportion is to be calculated over.
# There are several ways to do this:

# by overall proportion
d + stat_sum(aes(group = 1))
d + stat_sum(aes(group = 1)) + scale_size(range = c(3, 10))
d + stat_sum(aes(group = 1)) + scale_size_area(max_size = 10)

# by cut
d + stat_sum(aes(group = cut))
d + stat_sum(aes(group = cut, colour = cut))
```

```
# by clarity
d + stat_sum(aes(group = clarity))
d + stat_sum(aes(group = clarity, colour = cut))

# Instead of proportions, can also use sums
d + stat_sum(aes(size = ..n..))

# Can also weight by another variable
d + stat_sum(aes(group = 1, weight = price))
d + stat_sum(aes(group = 1, weight = price, size = ..n..))

# Or using qplot
qplot(cut, clarity, data = diamonds)
qplot(cut, clarity, data = diamonds, stat = "sum", group = 1)
```

stat\_summary

*Summarise y values at every unique x.*

## Description

stat\_summary allows for tremendous flexibility in the specification of summary functions. The summary function can either supply individual summary functions for each of y, ymin and ymax (with fun.y, fun.ymax, fun.ymin), or return a data frame containing any number of aesthetics with fun.data. All summary functions are called with a single vector of values, x.

## Usage

```
stat_summary(mapping = NULL, data = NULL, geom = "pointrange",
  position = "identity", ...)
```

## Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Details

A simple vector function is easiest to work with as you can return a single number, but is somewhat less flexible. If your summary function operates on a data.frame it should return a data frame with variables that the geom can use.

**Value**

a data.frame with additional columns:

fun.data	Complete summary function. Should take data frame as input and return data frame as output
fun.ymin	ymin summary function (should take numeric vector and return single number)
fun.y	y summary function (should take numeric vector and return single number)
fun.ymax	ymax summary function (should take numeric vector and return single number)

**Aesthetics**

stat\_summary understands the following aesthetics (required aesthetics are in bold):

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

**See Also**

[geom\\_errorbar](#), [geom\\_pointrange](#), [geom\\_linerange](#), [geom\\_crossbar](#) for geoms to display summarised data

**Examples**

```
# Basic operation on a small dataset
d <- qplot(cyl, mpg, data=mtcars)
d + stat_summary(fun.data = "mean_cl_boot", colour = "red")

p <- qplot(cyl, mpg, data = mtcars, stat="summary", fun.y = "mean")
p
# Don't use ylim to zoom into a summary plot - this throws the
# data away
p + ylim(15, 30)
# Instead use coord_cartesian
p + coord_cartesian(ylim = c(15, 30))

# You can supply individual functions to summarise the value at
# each x:

stat_sum_single <- function(fun, geom="point", ...) {
  stat_summary(fun.y=fun, colour="red", geom=geom, size = 3, ...)
}

d + stat_sum_single(mean)
d + stat_sum_single(mean, geom="line")
d + stat_sum_single(median)
d + stat_sum_single(sd)

d + stat_summary(fun.y = mean, fun.ymin = min, fun.ymax = max,
  colour = "red")
```

```

d + aes(colour = factor(vs)) + stat_summary(fun.y = mean, geom="line")

# Alternatively, you can supply a function that operates on a data.frame.
# A set of useful summary functions is provided from the Hmisc package:

stat_sum_df <- function(fun, geom="crossbar", ...) {
  stat_summary(fun.data=fun, colour="red", geom=geom, width=0.2, ...)
}

# The crossbar geom needs grouping to be specified when used with
# a continuous x axis.
d + stat_sum_df("mean_cl_boot", mapping = aes(group = cyl))
d + stat_sum_df("mean_sdl", mapping = aes(group = cyl))
d + stat_sum_df("mean_sdl", mult = 1, mapping = aes(group = cyl))
d + stat_sum_df("median_hilow", mapping = aes(group = cyl))

# There are lots of different geoms you can use to display the summaries

d + stat_sum_df("mean_cl_normal", mapping = aes(group = cyl))
d + stat_sum_df("mean_cl_normal", geom = "errorbar")
d + stat_sum_df("mean_cl_normal", geom = "pointrange")
d + stat_sum_df("mean_cl_normal", geom = "smooth")

# Summaries are more useful with a bigger data set:
mpg2 <- subset(mpg, cyl != 5L)
m <- ggplot(mpg2, aes(x=cyl, y=hwy)) +
  geom_point() +
  stat_summary(fun.data = "mean_sdl", geom = "linerange",
              colour = "red", size = 2, mult = 1) +
  xlab("cyl")

m

# An example with highly skewed distributions:
set.seed(596)
mov <- movies[sample(nrow(movies), 1000), ]
m2 <- ggplot(mov, aes(x= factor(round(rating)), y=votes)) + geom_point()
m2 <- m2 + stat_summary(fun.data = "mean_cl_boot", geom = "crossbar",
                      colour = "red", width = 0.3) + xlab("rating")

m2

# Notice how the overplotting skews off visual perception of the mean
# supplementing the raw data with summary statistics is very important

# Next, we'll look at votes on a log scale.

# Transforming the scale means the data are transformed
# first, after which statistics are computed:
m2 + scale_y_log10()

# Transforming the coordinate system occurs after the
# statistic has been computed. This means we're calculating the summary on the raw data
# and stretching the geoms onto the log scale. Compare the widths of the
# standard errors.
m2 + coord_trans(y="log10")

```

---

stat_summary2d	<i>Apply function for 2D rectangular bins.</i>
----------------	------------------------------------------------

---

### Description

Apply function for 2D rectangular bins.

### Usage

```
stat_summary2d(mapping = NULL, data = NULL, geom = NULL,  
  position = "identity", bins = 30, drop = TRUE, fun = mean, ...)
```

### Arguments

bins	see <a href="#">stat_bin2d</a>
drop	drop if the output of fun is NA.
fun	function for summary.
...	parameters passed to fun
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer

### Aesthetics

stat\_summary2d understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **z**
- fill

stat\_summary2d is 2D version of [stat\\_summary](#). The data are divided by x and y. z in each cell is passed to arbitrary summary function.

stat\_summary2d requires the following aesthetics:

- x: horizontal position
- y: vertical position
- z: value passed to the summary function

### See Also

[stat\\_summary\\_hex](#) for hexagonal summarization. [stat\\_bin2d](#) for the binning options.

## Examples

```
d <- ggplot(diamonds, aes(carat, depth, z = price))
d + stat_summary2d()

# Specifying function
d + stat_summary2d(fun = function(x) sum(x^2))
d + stat_summary2d(fun = var)
```

---

stat_summary_hex	<i>Apply function for 2D hexagonal bins.</i>
------------------	----------------------------------------------

---

## Description

Apply function for 2D hexagonal bins.

## Usage

```
stat_summary_hex(mapping = NULL, data = NULL, geom = "hex",
  position = "identity", bins = 30, drop = TRUE, fun = mean, ...)
```

## Arguments

bins	see <a href="#">stat_binhex</a>
drop	drop if the output of fun is NA.
fun	function for summary.
...	parameters passed to fun
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer

## Aesthetics

stat\_summaryhex understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **z**
- fill

stat\_summary2d is hexagonal version of [stat\\_summary](#). The data are divided by x and y. z in each cell is passed to arbitrary summary function.

stat\_summary\_hex requires the following aesthetics:

- x: horizontal position
- y: vertical position
- z: value passed to the summary function

### See Also

[stat\\_summary2d](#) for rectangular summarization. [stat\\_bin2d](#) for the hexagon-ing options.

### Examples

```
d <- ggplot(diamonds, aes(carat, depth, z = price))
d + stat_summary_hex()

# Specifying function
d + stat_summary_hex(fun = function(x) sum(x^2))
d + stat_summary_hex(fun = var, na.rm = TRUE)
```

---

stat_unique	<i>Remove duplicates.</i>
-------------	---------------------------

---

### Description

Remove duplicates.

### Usage

```
stat_unique(mapping = NULL, data = NULL, geom = "point",
  position = "identity", ...)
```

### Arguments

mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

### Aesthetics

stat\_unique understands the following aesthetics (required aesthetics are in bold):

-

## Examples

```
ggplot(mtcars, aes(vs, am)) + geom_point(alpha = 0.1)
ggplot(mtcars, aes(vs, am)) + geom_point(alpha = 0.1, stat="unique")
```

---

stat_ydensity	<i>1d kernel density estimate along y axis, for violin plot.</i>
---------------	------------------------------------------------------------------

---

## Description

1d kernel density estimate along y axis, for violin plot.

## Usage

```
stat_ydensity(mapping = NULL, data = NULL, geom = "violin",
  position = "dodge", adjust = 1, kernel = "gaussian", trim = TRUE,
  scale = "area", na.rm = FALSE, ...)
```

## Arguments

trim	If TRUE (default), trim the tails of the violins to the range of the data. If FALSE, don't trim the tails.
scale	if "area" (default), all violins have the same area (before trimming the tails). If "count", areas are scaled proportionally to the number of observations. If "width", all violins have the same maximum width.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
mapping	The aesthetic mapping, usually constructed with <a href="#">aes</a> or <a href="#">aes_string</a> . Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
adjust	see <a href="#">density</a> for details
kernel	kernel used for density estimation, see <a href="#">density</a> for details
...	other arguments passed on to <a href="#">layer</a> . This can include aesthetics whose values you want to set, not map. See <a href="#">layer</a> for more details.

## Value

A data frame with additional columns:

density	density estimate
scaled	density estimate, scaled to maximum of 1
count	density * number of points - probably useless for violin plots



violinwidth	density scaled for the violin plot, according to area, counts or to a constant maximum width
n	number of points
width	width of violin bounding box

### Aesthetics

`stat_ydensity` understands the following aesthetics (required aesthetics are in bold):

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

### See Also

[geom\\_violin](#) for examples, and [stat\\_density](#) for examples with data along the x axis.

### Examples

```
# See geom_violin for examples
# Also see stat_density for similar examples with data along x axis
```

---

theme	<i>Set theme elements</i>
-------	---------------------------

---

### Description

Use this function to modify theme settings.

### Usage

```
theme(..., complete = FALSE)
```

### Arguments

...	a list of element name, element pairings that modify the existing theme.
complete	set this to TRUE if this is a complete theme, such as the one returned by <code>theme_grey()</code> . Complete themes behave differently when added to a ggplot object.

### Details

Theme elements can inherit properties from other theme elements. For example, `axis.title.x` inherits from `axis.title`, which in turn inherits from `text`. All text elements inherit directly or indirectly from `text`; all lines inherit from `line`, and all rectangular objects inherit from `rect`.

For more examples of modifying properties using inheritance, see [+.gg](#) and [%+replace%](#).

To see a graphical representation of the inheritance tree, see the last example below.

### Theme elements

The individual theme elements are:

line	all line elements (element_line)
rect	all rectangular elements (element_rect)
text	all text elements (element_text)
title	all title elements: plot, axes, legends (element_text; inherits from text)
axis.title	label of axes (element_text; inherits from text)
axis.title.x	x axis label (element_text; inherits from axis.title)
axis.title.y	y axis label (element_text; inherits from axis.title)
axis.text	tick labels along axes (element_text; inherits from text)
axis.text.x	x axis tick labels (element_text; inherits from axis.text)
axis.text.y	y axis tick labels (element_text; inherits from axis.text)
axis.ticks	tick marks along axes (element_line; inherits from line)
axis.ticks.x	x axis tick marks (element_line; inherits from axis.ticks)
axis.ticks.y	y axis tick marks (element_line; inherits from axis.ticks)
axis.ticks.length	length of tick marks (unit)
axis.ticks.margin	space between tick mark and tick label (unit)
axis.line	lines along axes (element_line; inherits from line)
axis.line.x	line along x axis (element_line; inherits from axis.line)
axis.line.y	line along y axis (element_line; inherits from axis.line)
legend.background	background of legend (element_rect; inherits from rect)
legend.margin	extra space added around legend (unit)
legend.key	background underneath legend keys (element_rect; inherits from rect)
legend.key.size	size of legend keys (unit; inherits from legend.key.size)
legend.key.height	key background height (unit; inherits from legend.key.size)
legend.key.width	key background width (unit; inherits from legend.key.size)
legend.text	legend item labels (element_text; inherits from text)
legend.text.align	alignment of legend labels (number from 0 (left) to 1 (right))
legend.title	title of legend (element_text; inherits from title)
legend.title.align	alignment of legend title (number from 0 (left) to 1 (right))
legend.position	the position of legends ("none", "left", "right", "bottom", "top", or two-element numeric vector)
legend.direction	layout of items in legends ("horizontal" or "vertical")
legend.justification	anchor point for positioning legend inside plot ("center" or two-element numeric vector)
legend.box	arrangement of multiple legends ("horizontal" or "vertical")
legend.box.just	justification of each legend within the overall bounding box, when there are multiple legends ("top", "bottom", "left", "right", "center", or two-element numeric vector)
panel.background	background of plotting area, drawn underneath plot (element_rect; inherits from rect)
panel.border	border around plotting area, drawn on top of plot so that it covers tick marks and grid lines. This should be drawn on top of the plot area.
panel.margin	margin around facet panels (unit)
panel.margin.x	horizontal margin around facet panels (unit; inherits from panel.margin)
panel.margin.y	vertical margin around facet panels (unit; inherits from panel.margin)
panel.grid	grid lines (element_line; inherits from line)
panel.grid.major	major grid lines (element_line; inherits from panel.grid)
panel.grid.minor	minor grid lines (element_line; inherits from panel.grid)
panel.grid.major.x	vertical major grid lines (element_line; inherits from panel.grid.major)
panel.grid.major.y	horizontal major grid lines (element_line; inherits from panel.grid.major)
panel.grid.minor.x	vertical minor grid lines (element_line; inherits from panel.grid.minor)
panel.grid.minor.y	horizontal minor grid lines (element_line; inherits from panel.grid.minor)
plot.background	background of the entire plot (element_rect; inherits from rect)
plot.title	plot title (text appearance) (element_text; inherits from title)
plot.margin	margin around entire plot (unit with the sizes of the top, right, bottom, and left margins)

<code>strip.background</code>	background of facet labels ( <code>element_rect</code> ; inherits from <code>rect</code> )
<code>strip.text</code>	facet labels ( <code>element_text</code> ; inherits from <code>text</code> )
<code>strip.text.x</code>	facet labels along horizontal direction ( <code>element_text</code> ; inherits from <code>strip.text</code> )
<code>strip.text.y</code>	facet labels along vertical direction ( <code>element_text</code> ; inherits from <code>strip.text</code> )

## See Also

[+.gg](#)  
[%+replace%](#)  
[rel](#)  
[element\\_blank](#)  
[element\\_line](#)  
[element\\_rect](#)  
[element\\_text](#)

## Examples

```

p <- qplot(mpg, wt, data = mtcars)
p
p + theme(panel.background = element_rect(colour = "pink"))
p + theme_bw()

# Scatter plot of gas mileage by vehicle weight
p <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()
# Calculate slope and intercept of line of best fit
coef(lm(mpg ~ wt, data = mtcars))
p + geom_abline(intercept = 37, slope = -5)
# Calculate correlation coefficient
with(mtcars, cor(wt, mpg, use = "everything", method = "pearson"))
#annotate the plot
p + geom_abline(intercept = 37, slope = -5) +
  geom_text(data = data.frame(), aes(4.5, 30, label = "Pearson-R = -.87"))

# Change the axis labels
# Original plot
p
p + xlab("Vehicle Weight") + ylab("Miles per Gallon")
# Or
p + labs(x = "Vehicle Weight", y = "Miles per Gallon")

# Change title appearance
p <- p + labs(title = "Vehicle Weight-Gas Mileage Relationship")
# Set title to twice the base font size
p + theme(plot.title = element_text(size = rel(2)))
p + theme(plot.title = element_text(size = rel(2), colour = "blue"))

# Changing plot look with themes

```

```

DF <- data.frame(x = rnorm(400))
m <- ggplot(DF, aes(x = x)) + geom_histogram()
# Default is theme_grey()
m
# Compare with
m + theme_bw()

# Manipulate Axis Attributes
library(grid) # for unit
m + theme(axis.line = element_line(size = 3, colour = "red", linetype = "dotted"))
m + theme(axis.text = element_text(colour = "blue"))
m + theme(axis.text.y = element_blank())
m + theme(axis.ticks = element_line(size = 2))
m + theme(axis.title.y = element_text(size = rel(1.5), angle = 90))
m + theme(axis.title.x = element_blank())
m + theme(axis.ticks.length = unit(.85, "cm"))

# Legend Attributes
z <- ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) + geom_point()
z
z + theme(legend.position = "none")
z + theme(legend.position = "bottom")
# Or use relative coordinates between 0 and 1
z + theme(legend.position = c(.5, .5))
z + theme(legend.background = element_rect(colour = "black"))
# Legend margin controls extra space around outside of legend:
z + theme(legend.background = element_rect(), legend.margin = unit(1, "cm"))
z + theme(legend.background = element_rect(), legend.margin = unit(0, "cm"))
# Or to just the keys
z + theme(legend.key = element_rect(colour = "black"))
z + theme(legend.key = element_rect(fill = "yellow"))
z + theme(legend.key.size = unit(2.5, "cm"))
z + theme(legend.text = element_text(size = 20, colour = "red", angle = 45))
z + theme(legend.title = element_text(face = "italic"))

# To change the title of the legend use the name argument
# in one of the scale options
z + scale_colour_brewer(name = "My Legend")
z + scale_colour_grey(name = "Number of \nCylinders")

# Panel and Plot Attributes
z + theme(panel.background = element_rect(fill = "black"))
z + theme(panel.border = element_rect(linetype = "dashed", colour = "black"))
z + theme(panel.grid.major = element_line(colour = "blue"))
z + theme(panel.grid.minor = element_line(colour = "red", linetype = "dotted"))
z + theme(panel.grid.major = element_line(size = 2))
z + theme(panel.grid.major.y = element_blank(), panel.grid.minor.y = element_blank())
z + theme(plot.background = element_rect())
z + theme(plot.background = element_rect(fill = "green"))

# Faceting Attributes
set.seed(4940)
dsmall <- diamonds[sample(nrow(diamonds), 1000), ]

```

```

k <- ggplot(dsmall, aes(carat, ..density..)) + geom_histogram(binwidth = 0.2) +
  facet_grid(. ~ cut)
k + theme(strip.background = element_rect(colour = "purple", fill = "pink",
                                           size = 3, linetype = "dashed"))
k + theme(strip.text.x = element_text(colour = "red", angle = 45, size = 10,
                                       hjust = 0.5, vjust = 0.5))
k + theme(panel.margin = unit(5, "lines"))
k + theme(panel.margin.y = unit(0, "lines"))

# Modify a theme and save it
mytheme <- theme_grey() + theme(plot.title = element_text(colour = "red"))
p + mytheme

## Run this to generate a graph of the element inheritance tree
build_element_graph <- function(tree) {
  require(igraph)
  require(plyr)

  inheritdf <- function(name, item) {
    if (length(item$inherit) == 0)
      data.frame()
    else
      data.frame(child = name, parent = item$inherit)
  }

  edges <- rbind.fill(mapply(inheritdf, names(tree), tree))

  # Explicitly add vertices (since not all are in edge list)
  vertices <- data.frame(name = names(tree))
  graph.data.frame(edges, vertices = vertices)
}

g <- build_element_graph(ggplot2:::element_tree)
V(g)$label <- V(g)$name

set.seed(324)
par(mar=c(0,0,0,0)) # Remove unnecessary margins
plot(g, layout=layout.fruchterman.reingold, vertex.size=4, vertex.label.dist=.25)

```

---

theme\_blank

*Deprecated theme\_xx functions*


---

## Description

The theme\_xx functions have been deprecated. They are replaced with the element\_xx functions.

**Usage**

```
theme_blank(...)  
  
theme_rect(...)  
  
theme_line(...)  
  
theme_segment(...)  
  
theme_text(...)
```

**Arguments**

... Arguments to be passed to the appropriate `element_xx` function.

---

theme_update	<i>Get, set and update themes.</i>
--------------	------------------------------------

---

**Description**

Use `theme_update` to modify a small number of elements of the current theme or use `theme_set` to completely override it.

**Usage**

```
theme_update(...)  
  
theme_get()  
  
theme_set(new)
```

**Arguments**

...	named list of theme settings
new	new theme (a list of theme elements)

**See Also**

[%+replace%](#) and [+.gg](#)

**Examples**

```
qplot(mpg, wt, data = mtcars)  
old <- theme_set(theme_bw())  
qplot(mpg, wt, data = mtcars)  
theme_set(old)  
qplot(mpg, wt, data = mtcars)
```

```

old <- theme_update(panel.background = element_rect(colour = "pink"))
qplot(mpg, wt, data = mtcars)
theme_set(old)
theme_get()

qplot(mpg, wt, data=mtcars, colour=mpg) +
  theme(legend.position=c(0.95, 0.95), legend.justification = c(1, 1))
last_plot() +
  theme(legend.background = element_rect(fill = "white", colour = "white", size = 3))

```

---

translate\_qplot\_base    *Translating between qplot and base graphics*

---

## Description

There are two types of graphics functions in base graphics, those that draw complete graphics and those that add to existing graphics.

## Details

qplot() has been designed to mimic plot(), and can do the job of all other high-level plotting commands. There are only two graph types from base graphics that cannot be replicated with ggplot2: filled.contour() and persp()

## Examples

```

# High-level plotting commands

x <- runif(10)
y <- 1:10
plot(x, y); dotchart(x, y)
qplot(x, y)

plot(x, y, type = "l")
qplot(x, y, geom = "line")

plot(x, y, type = "s")
qplot(x, y, geom = "step")

plot(x, y, type = "b")
qplot(x, y, geom = c("point", "line"))

boxplot(x, y)
qplot(x, y, geom = "boxplot")

hist(x)
qplot(x, geom = "histogram")

```

```

# cdplot(factor(x), y)
# qplot(x, fill = y, geom = "density", position = "fill")

# coplot(y ~ x | a + b)
# qplot(x, y, facets = a ~ b)

# Many of the geoms are parameterised differently than base graphics. For
# example, hist() is parameterised in terms of the number of bins, while
# geom_histogram() is parameterised in terms of the width of each bin.
hist(x, bins = 10)
qplot(x, geom = "histogram", binwidth = .1)

# qplot() often requires data in a slightly different format to the base
# graphics functions. For example, the bar geom works with untabulated data,
# not tabulated data like barplot(); the tile and contour geoms expect data
# in a data frame, not a matrix like image() and contour().
barplot(table(x))
qplot(x, geom = "bar")

barplot(x)
qplot(seq_along(x), x, geom = "bar", stat = "identity")

# image(x)
# qplot(X1, X2, data = melt(x), geom = "tile", fill = value)

# contour(x)
# qplot(X1, X2, data = melt(x), geom = "contour", fill = value)

# Generally, the base graphics functions work with individual vectors, not
# data frames like ggplot2. qplot() will try to construct a data frame if one
# is not specified, but it is not always possible. If you get strange errors,
# you may need to create the data frame yourself.
df <- data.frame(x = x, y = y)
with(df, plot(x, y))
qplot(x, y, data = df)

# By default, qplot() maps values to aesthetics with a scale. To override
# this behaviour and set aesthetics, overriding the defaults, you need to use I().
plot(x, y, col = "red", cex = 1)
qplot(x, y, colour = I("red"), size = I(1))

# Low-level drawing

# The low-level drawing functions which add to an existing plot are equivalent
# to adding a new layer in ggplot2.

# Base function      ggplot2 layer
# curve()            geom_curve()
# hline()            geom_hline()
# lines()            geom_line()
# points()           geom_point()
# polygon()          geom_polygon()

```



```

# rect()          geom_rect()
# rug()           geom_rug()
# segments()      geom_segment()
# text()          geom_text()
# vline()         geom_vline()
# abline(lm(y ~ x)) geom_smooth(method = "lm")
# lines(density(x)) geom_density()
# lines(loess(x, y)) geom_smooth()

plot(x, y)
lines(x, y)

qplot(x, y) + geom_line()

# Or, building up piece-meal
qplot(x, y)
last_plot() + geom_line()

# Legends, axes and grid lines

# In ggplot2, the appearance of legends and axes is controlled by the scales.
# Axes are produced by the x and y scales, while all other scales produce legends.
# See ?theme for help changing the appearance of axes and legends.
# The appearance of grid lines is controlled by the grid.major and grid.minor
# theme options, and their position by the breaks of the x and y scales.

# Colour palettes

# Instead of global colour palettes, ggplot2 has scales for individual plots. Much
# of the time you can rely on the default colour scale (which has somewhat better
# perceptual properties), but if you want to reuse an existing colour palette, you
# can use scale_colour_manual(). You will need to make sure that the colour
# is a factor for this to work.

palette(rainbow(5))
plot(1:5, 1:5, col = 1:5, pch = 19, cex = 4)

qplot(1:5, 1:5, col = factor(1:5), size = I(4))
last_plot() + scale_colour_manual(values = rainbow(5))

# In ggplot2, you can also use palettes with continuous values,
# with intermediate values being linearly interpolated.

qplot(0:100, 0:100, col = 0:100, size = I(4)) +
  scale_colour_gradientn(colours = rainbow(7))
last_plot() + scale_colour_gradientn(colours = terrain.colors(7))

# Graphical parameters

# The majority of par settings have some analogue within the theme system, or
# in the defaults of the geoms and scales. The appearance plot border drawn
# by box() can be controlled in a similar way by the panel.background and
# plot.background theme elements. Instead of using title(), the plot title is

```

```
# set with the title option. See ?theme for more theme elements.
last_plot() + labs(title = "My Plot Title")
```

---

```
translate_qplot_ggplot
```

*Translating between qplot and ggplot*

---

## Description

Within ggplot2, there are two basic methods to create plots, with qplot() and ggplot(). qplot() is designed primarily for interactive use: it makes a number of assumptions that speed most cases, but when designing multilayered plots with different data sources it can get in the way. This section describes what those defaults are, and how they map to the fuller ggplot() syntax.

## Examples

```
# By default, qplot() assumes that you want a scatterplot,
# i.e., you want to use geom_point()
# qplot(x, y, data = data)
# ggplot(data, aes(x, y)) + geom_point()

# Using Aesthetics

# If you map additional aesthetics, these will be added to the defaults. With
# qplot() there is no way to use different aesthetic mappings (or data) in
# different layers
# qplot(x, y, data = data, shape = shape, colour = colour)
# ggplot(data, aes(x, y, shape = shape, colour = colour)) + geom_point()
#
# Aesthetic parameters in qplot() always try to map the aesthetic to a
# variable. If the argument is not a variable but a value, effectively a new column
# is added to the original dataset with that value. To set an aesthetic to a
# value and override the default appearance, you surround the value with I() in
# qplot(), or pass it as a parameter to the layer.
# qplot(x, y, data = data, colour = I("red"))
# ggplot(data, aes(x, y)) + geom_point(colour = "red")

# Changing the geom parameter changes the geom added to the plot
# qplot(x, y, data = data, geom = "line")
# ggplot(data, aes(x, y)) + geom_line()

# Not all geoms require both x and y, e.g., geom_bar() and geom_histogram().
# For these two geoms, if the y aesthetic is not supplied, both qplot and
# ggplot commands default to "count" on the y-axis
# ggplot(data, aes(x)) + geom_bar()
# qplot(x, data = data, geom = "bar")

# If a vector of multiple geom names is supplied to the geom argument, each
# geom will be added in turn
```

```

# qplot(x, y, data = data, geom = c("point", "smooth"))
# ggplot(data, aes(x, y)) + geom_point() + geom_smooth()

# Unlike the rest of ggplot2, stats and geoms are independent
# qplot(x, y, data = data, stat = "bin")
# ggplot(data, aes(x, y)) + geom_point(stat = "bin")
#
# Any layer parameters will be passed on to all layers. Most layers will ignore
# parameters that they don't need
# qplot(x, y, data = data, geom = c("point", "smooth"), method = "lm")
# ggplot(data, aes(x, y)) + geom_point(method = "lm") + geom_smooth(method = "lm")

# Scales and axes

# You can control basic properties of the x and y scales with the xlim, ylim,
# xlab and ylab arguments
# qplot(x, y, data = data, xlim = c(1, 5), xlab = "my label")
# ggplot(data, aes(x, y)) + geom_point() +
# scale_x_continuous("my label", limits = c(1, 5))

# qplot(x, y, data = data, xlim = c(1, 5), ylim = c(10, 20))
# ggplot(data, aes(x, y)) + geom_point() +
# scale_x_continuous(limits = c(1, 5)) + scale_y_continuous(limits = c(10, 20))

# Like plot(), qplot() has a convenient way of log transforming the axes.
# qplot(x, y, data = data, log = "xy")
# ggplot(data, aes(x, y)) + geom_point() + scale_x_log10() + scale_y_log10()
# There are many other possible transformations, but not all are
# accessible from within qplot(), see ?scale_continuous for more

# Plot options

# qplot() recognises the same options as plot does, and converts them to their
# ggplot2 equivalents. See ?theme for more on ggplot options
# qplot(x, y, data = data, main="title", asp = 1)
# ggplot(data, aes(x, y)) + geom_point() + labs(title = "title") + theme(aspect.ratio = 1)

```

---

translate\_qplot\_gpl     *Translating between qplot and Graphics Production Library (GPL)*

---

## Description

The Grammar of Graphics uses two specifications. A concise format is used to caption figures, and a more detailed xml format stored on disk.

## Examples

```

# The following example of the concise format is adapted from Figure 1.5,
# page 13, of Leland Wilkinson's "The Grammar of Graphics."
# Springer, 2nd edition, 2005.

```

```

# DATA: source("demographics")
# DATA: longitude, latitude = map(source("World"))
# TRANS: bd = max(birth - death, 0)
# COORD: project.mercator()
# ELEMENT: point(position(lon * lat), size(bd), color(color.red))
# ELEMENT: polygon(position(longitude * latitude))

# This is relatively simple to adapt to the syntax of ggplot2:

# ggplot() is used to specify the default data and default aesthetic mappings.
# Data is provided as standard R data.frames existing in the global environment;
# it does not need to be explicitly loaded. We also use a slightly
# different world dataset, with columns lat and long. This lets us use the
# same aesthetic mappings for both datasets. Layers can override the default
# data and aesthetic mappings provided by the plot.

# We replace TRANS with an explicit transformation by R code.

# ELEMENTs are replaced with layers, which explicitly specify the data
# source. Each geom has a default statistic which is used to transform the
# data prior to plotting. For the geoms in this example, the default statistic
# is the identity function. Fixed aesthetics (the colour red in this example)
# are supplied as additional arguments to the layer, rather than as special
# constants.

# The SCALE component has been omitted from this example (so that the
# defaults are used). In both the ggplot2 and GoG examples, scales are
# defined by default. In ggplot you can override the defaults by adding a
# scale object, e.g., scale_colour or scale_size.

# COORD uses a slightly different format. In general, most of the components
# specifications in ggplot are slightly different to those in GoG, in order to
# be more familiar to R users.

# Each component is added together with + to create the final plot.

# Resulting ggplot2 code:
# demographics <- transform(demographics, bd = pmax(birth - death, 0))
# p <- ggplot(demographic, aes(lon, lat))
# p <- p + geom_polygon(data = world)
# p <- p + geom_point(aes(size = bd), colour = "red")
# p <- p + coord_map(projection = "mercator")
# print(p)

```

## Description

The major difference between lattice and ggplot2 is that lattice uses a formula based interface. ggplot2 does not because the formula does not generalise well to more complicated situations.

## Examples

```
library(lattice)

xyplot(rating ~ year, data=movies)
qplot(year, rating, data=movies)

xyplot(rating ~ year | Comedy + Action, data = movies)
qplot(year, rating, data = movies, facets = ~ Comedy + Action)
# Or maybe
qplot(year, rating, data = movies, facets = Comedy ~ Action)

# While lattice has many different functions to produce different types of
# graphics (which are all basically equivalent to setting the panel argument),
# ggplot2 has qplot().

stripplot(~ rating, data = movies, jitter.data = TRUE)
qplot(rating, 1, data = movies, geom = "jitter")

histogram(~ rating, data = movies)
qplot(rating, data = movies, geom = "histogram")

bwplot(Comedy ~ rating ,data = movies)
qplot(factor(Comedy), rating, data = movies, type = "boxplot")

xyplot(wt ~ mpg, mtcars, type = c("p","smooth"))
qplot(mpg, wt, data = mtcars, geom = c("point","smooth"))

xyplot(wt ~ mpg, mtcars, type = c("p","r"))
qplot(mpg, wt, data = mtcars, geom = c("point","smooth"), method = "lm")

# The capabilities for scale manipulations are similar in both ggplot2 and
# lattice, although the syntax is a little different.

xyplot(wt ~ mpg | cyl, mtcars, scales = list(y = list(relation = "free")))
qplot(mpg, wt, data = mtcars) + facet_wrap(~ cyl, scales = "free")

xyplot(wt ~ mpg | cyl, mtcars, scales = list(log = 10))
qplot(mpg, wt, data = mtcars, log = "xy")

xyplot(wt ~ mpg | cyl, mtcars, scales = list(log = 2))
library(scales) # Load scales for log2_trans
qplot(mpg, wt, data = mtcars) + scale_x_continuous(trans = log2_trans()) +
  scale_y_continuous(trans = log2_trans())

xyplot(wt ~ mpg, mtcars, group = cyl, auto.key = TRUE)
# Map directly to an aesthetic like colour, size, or shape.
```

```

qplot(mpg, wt, data = mtcars, colour = cyl)

xyplot(wt ~ mpg, mtcars, xlim = c(20,30))
# Works like lattice, except you can't specify a different limit
# for each panel/facet
qplot(mpg, wt, data = mtcars, xlim = c(20,30))

# Both lattice and ggplot2 have similar options for controlling labels on the plot.

xyplot(wt ~ mpg, mtcars, xlab = "Miles per gallon", ylab = "Weight",
  main = "Weight-efficiency tradeoff")
qplot(mpg, wt, data = mtcars, xlab = "Miles per gallon", ylab = "Weight",
  main = "Weight-efficiency tradeoff")

xyplot(wt ~ mpg, mtcars, aspect = 1)
qplot(mpg, wt, data = mtcars, asp = 1)

# par.settings() is equivalent to + theme() and trellis.options.set()
# and trellis.par.get() to theme_set() and theme_get().
# More complicated lattice formulas are equivalent to rearranging the data
# before using ggplot2.

```

---

update_element	<i>Update theme param</i>
----------------	---------------------------

---

## Description

Update contents of a theme. (Deprecated)

## Usage

```
update_element(name, ...)
```

## Arguments

name	name of a theme element
...	Pairs of name and value of theme parameters.

## Details

This function is deprecated. Use `%+replace%` or `+.gg` instead.

## Value

Updated theme element

## See Also

`%+replace%` and `+.gg`

**Examples**

```
## Not run:
x <- element_text(size = 15)
update_element(x, colour = "red")
# Partial matching works
update_element(x, col = "red")
# So does positional
update_element(x, "Times New Roman")
# And it throws an error if you use an argument that doesn't exist
update_element(x, noargument = 12)
# Or multiple arguments with the same name
update_element(x, size = 12, size = 15)

# Will look up element if given name
update_element("axis.text.x", colour = 20)
# Throws error if incorrectly named
update_element("axis.text", colour = 20)

## End(Not run)
```

---

update\_geom\_defaults    *Modify geom/stat aesthetic defaults for future plots*

---

**Description**

Modify geom/stat aesthetic defaults for future plots

**Usage**

```
update_geom_defaults(geom, new)

update_stat_defaults(stat, new)
```

**Arguments**

stat, geom	name of geom/stat to modify
new	named list of aesthetics

**Examples**

```
update_geom_defaults("point", list(colour = "darkblue"))
qplot(mpg, wt, data = mtcars)
update_geom_defaults("point", list(colour = "black"))
```

---

update_labels	<i>Update axis/legend labels</i>
---------------	----------------------------------

---

**Description**

Update axis/legend labels

**Usage**

```
update_labels(p, labels)
```

**Arguments**

p	plot to modify
labels	named list of new labels

**Examples**

```
p <- qplot(mpg, wt, data = mtcars)
update_labels(p, list(x = "New x"))
update_labels(p, list(x = expression(x / y ^ 2)))
update_labels(p, list(x = "New x", y = "New Y"))
update_labels(p, list(colour = "Fail silently"))
```

---

xlim	<i>Convenience functions to set the limits of the x and y axis.</i>
------	---------------------------------------------------------------------

---

**Description**

Observations not in this range will be dropped completely and not passed to any other layers. If a NA value is substituted for one of the limits that limit is automatically calculated.

**Usage**

```
xlim(...)
ylim(...)
```

**Arguments**

...	if numeric, will create a continuous scale, if factor or character, will create a discrete scale.
-----	---------------------------------------------------------------------------------------------------

**See Also**

For changing x or y axis limits **without** dropping data observations, see [coord\\_cartesian](#).



**Examples**

```
# xlim
xlim(15, 20)
xlim(20, 15)
xlim(c(10, 20))
xlim("a", "b", "c")
qplot(mpg, wt, data=mtcars) + xlim(15, 20)
# with automatic lower limit
qplot(mpg, wt, data=mtcars) + xlim(NA, 20)

# ylim
ylim(15, 20)
ylim(c(10, 20))
ylim("a", "b", "c")
qplot(mpg, wt, data=mtcars) + ylim(0, 4)
# with automatic upper limit
qplot(mpg, wt, data=mtcars) + ylim(0, NA)
```

# Index

## \*Topic **datasets**

- diamonds, [33](#)
- economics, [34](#)
- midwest, [123](#)
- movies, [125](#)
- mpg, [126](#)
- msleep, [126](#)
- presidential, [133](#)
- seals, [160](#)

## \*Topic **hplot**

- print.ggplot, [133](#)
- +.gg, [6](#), [193](#), [195](#), [198](#), [206](#)
- %%(+.gg), [6](#)
- %+replace%(+.gg), [6](#)
- %+replace%, [193](#), [195](#), [198](#), [206](#)

add\_theme, [8](#)

aes, [8](#), [15](#), [16](#), [47](#), [49](#), [50](#), [52](#), [53](#), [55](#), [57–60](#),  
[62](#), [64](#), [66–69](#), [72–74](#), [76](#), [78](#), [79](#), [82](#),  
[84](#), [85](#), [87](#), [88](#), [90](#), [91](#), [93–95](#), [97](#), [98](#),  
[100](#), [102](#), [104](#), [161](#), [162](#), [164–167](#),  
[169](#), [172–174](#), [176–179](#), [181](#), [183](#),  
[185](#), [186](#), [189–192](#)

aes\_all, [9](#)

aes\_auto, [9](#)

aes\_colour\_fill\_alpha, [8](#), [10](#)

aes\_group\_order, [8](#), [11](#)

aes\_linetype\_size\_shape, [8](#), [13](#)

aes\_position, [8](#), [14](#)

aes\_q, [8](#)

aes\_q(aes\_string), [15](#)

aes\_string, [8](#), [15](#), [47](#), [49](#), [50](#), [52](#), [53](#), [55](#),  
[57–60](#), [62](#), [64](#), [66–69](#), [72–74](#), [76](#), [78](#),  
[79](#), [82](#), [84](#), [85](#), [87](#), [88](#), [90](#), [91](#), [93–95](#),  
[97](#), [98](#), [100](#), [102](#), [104](#), [161](#), [162](#),  
[164–167](#), [169](#), [172–174](#), [176–179](#),  
[181](#), [183](#), [185](#), [186](#), [189–192](#)

alpha(aes\_colour\_fill\_alpha), [10](#)

annotate, [16](#)

annotation\_custom, [17](#)

annotation\_logticks, [18](#)

annotation\_map, [20](#)

annotation\_raster, [21](#)

autoplot, [22](#)

borders, [22](#), [45](#)

boxplot.stats, [54](#)

bquote, [119](#)

calc\_element, [23](#)

capitalize, [117](#)

cld, [43](#)

color(aes\_colour\_fill\_alpha), [10](#)

colour(aes\_colour\_fill\_alpha), [10](#)

continuous\_scale, [137](#), [138](#), [148](#), [152–155](#),  
[157](#)

cooks.distance, [44](#)

coord\_cartesian, [24](#), [26](#), [208](#)

coord\_equal(coord\_fixed), [25](#)

coord\_fixed, [25](#)

coord\_flip, [26](#)

coord\_map, [26](#), [29](#)

coord\_polar, [28](#)

coord\_quickmap, [29](#), [29](#)

coord\_trans, [19](#), [30](#)

county, [122](#)

cut, [32](#), [33](#)

cut\_interval, [32](#), [33](#)

cut\_number, [32](#), [32](#)

date\_breaks, [155](#), [157](#)

density, [169](#), [192](#)

diamonds, [33](#)

discrete\_scale, [137](#), [138](#), [140](#), [142](#),  
[144–146](#), [148–151](#), [159](#)

economics, [34](#)

element\_blank, [34](#), [195](#)

element\_line, [35](#), [195](#)

element\_rect, [35](#), [195](#)

- element\_text, 36, 110–113, 195
- eqscplot, 25
- expand\_limits, 36
- facet\_grid, 37, 117, 134
- facet\_null, 40
- facet\_wrap, 41, 134
- fill (aes\_colour\_fill\_alpha), 10
- fortify, 20, 22, 42, 78
- fortify-multcomp, 43
- fortify.cld (fortify-multcomp), 43
- fortify.confint.glht  
(fortify-multcomp), 43
- fortify.glht (fortify-multcomp), 43
- fortify.Line (fortify.sp), 46
- fortify.Lines (fortify.sp), 46
- fortify.lm, 42, 44
- fortify.map, 45
- fortify.Polygon (fortify.sp), 46
- fortify.Polygons (fortify.sp), 46
- fortify.sp, 46
- fortify.SpatialLinesDataFrame  
(fortify.sp), 46
- fortify.SpatialPolygons (fortify.sp), 46
- fortify.SpatialPolygonsDataFrame  
(fortify.sp), 46
- fortify.summary.glht  
(fortify-multcomp), 43
- france, 122
- gam, 181
- geom\_abline, 47, 72, 104
- geom\_area, 48, 129
- geom\_bar, 48, 49, 69, 91, 129
- geom\_bin2d, 52
- geom\_blank, 36, 53
- geom\_boxplot, 54, 74, 82, 167
- geom\_contour, 57, 61
- geom\_crossbar, 58, 65, 77, 85, 187
- geom\_density, 59
- geom\_density2d, 58, 60
- geom\_dotplot, 61, 164
- geom\_errorbar, 59, 64, 66, 77, 85, 187
- geom\_errorbarh, 65
- geom\_freqpoly, 67
- geom\_hex, 68
- geom\_histogram, 60, 67, 69
- geom\_hline, 48, 71, 104
- geom\_jitter, 55, 73, 82, 83
- geom\_line, 74, 80, 94
- geom\_linerange, 59, 65, 76, 85, 91, 187
- geom\_map, 77
- geom\_path, 75, 79, 86, 94
- geom\_point, 74, 82
- geom\_pointrange, 59, 65, 77, 84, 187
- geom\_polygon, 22, 80, 85, 91
- geom\_quantile, 87
- geom\_raster, 21, 88
- geom\_rect, 89
- geom\_ribbon, 49, 75, 86, 91
- geom\_rug, 92
- geom\_segment, 48, 72, 75, 80, 93, 104
- geom\_smooth, 59, 65, 77, 85, 95
- geom\_step, 96
- geom\_text, 98
- geom\_tile, 30, 88, 99
- geom\_violin, 101, 193
- geom\_vline, 48, 72, 103
- ggfluctuation, 185
- ggplot, 22
- ggplot.data.frame, 105
- ggplot2, 105
- ggplot2-package (ggplot2), 105
- ggsave, 106, 122
- ggtheme, 107
- ggtitle (labs), 121
- glm, 182
- gray.colors, 145
- group (aes\_group\_order), 11
- guide\_colorbar, 108, 113
- guide\_colorbar (guide\_colourbar), 109
- guide\_colourbar, 108, 109, 113
- guide\_legend, 108, 111, 112
- guides, 108, 111, 113
- hmisc, 115
- image, 100
- is.ggplot, 115
- is.rel, 116
- is.theme, 116
- italy, 122
- kde2d, 171
- label\_both, 117, 118, 119, 120
- label\_bquote, 118, 119, 119, 120
- label\_parsed, 118, 119, 119, 120

- label\_value, [37](#), [118](#), [119](#), [120](#)
- label\_wrap\_gen, [120](#)
- labeller, [117](#), [121](#)
- labs, [110](#), [112](#), [121](#)
- last\_plot, [122](#)
- layer, [47](#), [49](#), [50](#), [52](#), [53](#), [55](#), [57–60](#), [62](#), [64](#),  
[66–69](#), [72–74](#), [76](#), [78](#), [80](#), [82](#), [84](#), [85](#),  
[87](#), [88](#), [90](#), [91](#), [93–95](#), [97](#), [98](#), [100](#),  
[102](#), [104](#), [161](#), [162](#), [164–167](#), [169](#),  
[173](#), [174](#), [176](#), [177](#), [180](#), [183](#), [185](#),  
[186](#), [191](#), [192](#)
- levelplot, [100](#)
- linetype(aes\_linetype\_size\_shape), [13](#)
- lm, [182](#)
- loess, [181](#), [182](#)
- map, [22](#), [122](#)
- map\_data, [45](#), [122](#)
- mapproject, [27](#)
- mean\_cl\_boot(hmisc), [115](#)
- mean\_cl\_normal(hmisc), [115](#)
- mean\_sdl(hmisc), [115](#)
- mean\_se, [123](#)
- median\_hilow(hmisc), [115](#)
- midwest, [123](#)
- movies, [125](#)
- mpg, [126](#)
- msleep, [126](#)
- nz, [122](#)
- opts, [127](#)
- order(aes\_group\_order), [11](#)
- plot, [134](#)
- plot.ggplot(print.ggplot), [133](#)
- plotmath, [119](#)
- position\_dodge, [50](#), [128](#), [129–132](#)
- position\_fill, [50](#), [128](#), [129](#), [130–132](#)
- position\_identity, [128](#), [129](#), [129](#), [130–132](#)
- position\_jitter, [73](#), [74](#), [128–130](#), [130](#), [131](#),  
[132](#)
- position\_jitterdodge, [128–130](#), [131](#), [132](#)
- position\_stack, [50](#), [128–131](#), [132](#)
- predict, [181](#)
- presidential, [133](#)
- print.ggplot, [133](#)
- qplot, [134](#)
- quickplot(qplot), [134](#)
- rel, [135](#), [195](#)
- rescale, [138](#), [144](#)
- resolution, [136](#)
- rq, [179](#)
- scale\_alpha, [137](#)
- scale\_alpha\_continuous(scale\_alpha),  
[137](#)
- scale\_alpha\_discrete(scale\_alpha), [137](#)
- scale\_alpha\_identity(scale\_identity),  
[148](#)
- scale\_alpha\_manual(scale\_manual), [149](#)
- scale\_area, [137](#), [137](#)
- scale\_color\_brewer, [141](#), [142](#), [144](#), [145](#),  
[147](#)
- scale\_color\_brewer  
(scale\_colour\_brewer), [138](#)
- scale\_color\_continuous, [139](#), [142](#), [144](#),  
[145](#), [147](#)
- scale\_color\_continuous  
(scale\_colour\_gradient), [140](#)
- scale\_color\_discrete, [139](#), [141](#), [142](#), [144](#),  
[145](#)
- scale\_color\_discrete  
(scale\_colour\_hue), [146](#)
- scale\_color\_distiller, [141](#), [142](#), [144](#), [145](#),  
[147](#)
- scale\_color\_distiller  
(scale\_colour\_brewer), [138](#)
- scale\_color\_gradient, [139](#), [142](#), [144](#), [145](#),  
[147](#)
- scale\_color\_gradient  
(scale\_colour\_gradient), [140](#)
- scale\_color\_gradient2, [139](#), [141](#), [144](#), [145](#),  
[147](#)
- scale\_color\_gradient2  
(scale\_colour\_gradient2), [142](#)
- scale\_color\_gradientn, [139](#), [141](#), [142](#), [145](#),  
[147](#)
- scale\_color\_gradientn  
(scale\_colour\_gradientn), [143](#)
- scale\_color\_grey, [139](#), [141](#), [142](#), [144](#), [147](#)
- scale\_color\_grey(scale\_colour\_grey),  
[145](#)
- scale\_color\_hue, [139](#), [141](#), [142](#), [144](#), [145](#)
- scale\_color\_hue(scale\_colour\_hue), [146](#)

- `scale_color_identity` (`scale_identity`), 148
- `scale_color_manual` (`scale_manual`), 149
- `scale_colour_brewer`, 138, 141, 142, 144, 145, 147
- `scale_colour_continuous`, 139, 142, 144, 145, 147
- `scale_colour_continuous` (`scale_colour_gradient`), 140
- `scale_colour_discrete`, 139, 141, 142, 144, 145
- `scale_colour_discrete` (`scale_colour_hue`), 146
- `scale_colour_distiller`, 141, 142, 144, 145, 147
- `scale_colour_distiller` (`scale_colour_brewer`), 138
- `scale_colour_gradient`, 139, 140, 142, 144, 145, 147
- `scale_colour_gradient2`, 139, 141, 142, 144, 145, 147
- `scale_colour_gradientn`, 139, 141, 142, 143, 145, 147
- `scale_colour_grey`, 139, 141, 142, 144, 145, 147
- `scale_colour_hue`, 139, 141, 142, 144, 145, 146
- `scale_colour_identity` (`scale_identity`), 148
- `scale_colour_manual` (`scale_manual`), 149
- `scale_fill_brewer`, 141, 142, 144, 145, 147
- `scale_fill_brewer` (`scale_colour_brewer`), 138
- `scale_fill_continuous`, 139, 142, 144, 145, 147
- `scale_fill_continuous` (`scale_colour_gradient`), 140
- `scale_fill_discrete`, 139, 141, 142, 144, 145
- `scale_fill_discrete` (`scale_colour_hue`), 146
- `scale_fill_distiller`, 141, 142, 144, 145, 147
- `scale_fill_distiller` (`scale_colour_brewer`), 138
- `scale_fill_gradient`, 139, 142, 144, 145, 147
- `scale_fill_gradient` (`scale_colour_gradient`), 140
- `scale_fill_gradient2`, 139, 141, 144, 145, 147
- `scale_fill_gradient2` (`scale_colour_gradient2`), 142
- `scale_fill_gradientn`, 139, 141, 142, 145, 147
- `scale_fill_gradientn` (`scale_colour_gradientn`), 143
- `scale_fill_grey`, 139, 141, 142, 144, 147
- `scale_fill_grey` (`scale_colour_grey`), 145
- `scale_fill_hue`, 139, 141, 142, 144, 145
- `scale_fill_hue` (`scale_colour_hue`), 146
- `scale_fill_identity` (`scale_identity`), 148
- `scale_fill_manual` (`scale_manual`), 149
- `scale_identity`, 148
- `scale_linetype`, 149
- `scale_linetype_continuous` (`scale_linetype`), 149
- `scale_linetype_discrete` (`scale_linetype`), 149
- `scale_linetype_identity` (`scale_identity`), 148
- `scale_linetype_manual` (`scale_manual`), 149
- `scale_manual`, 149
- `scale_shape`, 151
- `scale_shape_continuous` (`scale_shape`), 151
- `scale_shape_discrete` (`scale_shape`), 151
- `scale_shape_identity` (`scale_identity`), 148
- `scale_shape_manual` (`scale_manual`), 149
- `scale_size`, 83, 152
- `scale_size_area`, 137, 153
- `scale_size_continuous` (`scale_size`), 152
- `scale_size_discrete` (`scale_size`), 152
- `scale_size_identity` (`scale_identity`), 148
- `scale_size_manual` (`scale_manual`), 149
- `scale_x_continuous`, 153, 156, 157, 159
- `scale_x_date`, 154, 155, 157, 159
- `scale_x_datetime`, 154, 156, 157, 159
- `scale_x_discrete`, 154, 156, 157, 158
- `scale_x_log10`, 156, 157, 159
- `scale_x_log10` (`scale_x_continuous`), 153
- `scale_x_reverse`, 156, 157, 159

- scale\_x\_reverse (scale\_x\_continuous), 153
- scale\_x\_sqrt, 156, 157, 159
- scale\_x\_sqrt (scale\_x\_continuous), 153
- scale\_y\_continuous, 19, 156, 157, 159
- scale\_y\_continuous (scale\_x\_continuous), 153
- scale\_y\_date, 154, 157, 159
- scale\_y\_date (scale\_x\_date), 155
- scale\_y\_datetime, 154, 156, 159
- scale\_y\_datetime (scale\_x\_datetime), 157
- scale\_y\_discrete, 154, 156, 157
- scale\_y\_discrete (scale\_x\_discrete), 158
- scale\_y\_log10, 19, 156, 157, 159
- scale\_y\_log10 (scale\_x\_continuous), 153
- scale\_y\_reverse, 156, 157, 159
- scale\_y\_reverse (scale\_x\_continuous), 153
- scale\_y\_sqrt, 156, 157, 159
- scale\_y\_sqrt (scale\_x\_continuous), 153
- seals, 160
- seq\_gradient\_pal, 141
- shape (aes\_linetype\_size\_shape), 13
- size (aes\_linetype\_size\_shape), 13
- smean.cl.boot, 115
- smean.cl.normal, 115
- smean.sdl, 115
- smedian.hilow, 115
- stat\_bin, 50, 69, 160, 170
- stat\_bin2d, 162, 165, 189, 191
- stat\_bindot, 163
- stat\_binhex, 162, 165, 190
- stat\_boxplot, 166
- stat\_contour, 167, 172
- stat\_density, 59, 60, 169, 193
- stat\_density2d, 82, 171
- stat\_ecdf, 173
- stat\_ellipse, 174
- stat\_function, 175
- stat\_identity, 177
- stat\_qq, 178
- stat\_quantile, 55, 82, 88, 179
- stat\_smooth, 48, 82, 96, 181
- stat\_spoke, 183
- stat\_sum, 61, 82, 184
- stat\_summary, 59, 65, 77, 85, 115, 123, 186, 189, 191
- stat\_summary2d, 189, 191
- stat\_summary\_hex, 189, 190
- stat\_unique, 191
- stat\_ydensity, 192
- state, 122
- strwrap, 120
- substitute, 15
- theme, 7, 110–113, 127, 193
- theme\_blank, 197
- theme\_bw (ggtheme), 107
- theme\_classic (ggtheme), 107
- theme\_get (theme\_update), 198
- theme\_gray (ggtheme), 107
- theme\_grey (ggtheme), 107
- theme\_light (ggtheme), 107
- theme\_line (theme\_blank), 197
- theme\_linedraw (ggtheme), 107
- theme\_minimal (ggtheme), 107
- theme\_rect (theme\_blank), 197
- theme\_segment (theme\_blank), 197
- theme\_set (theme\_update), 198
- theme\_text (theme\_blank), 197
- theme\_update, 198
- trans\_new, 31
- translate\_qplot\_base, 199
- translate\_qplot\_ggplot, 202
- translate\_qplot\_gpl, 203
- translate\_qplot\_lattice, 204
- update\_element, 206
- update\_geom\_defaults, 207
- update\_labels, 208
- update\_stat\_defaults (update\_geom\_defaults), 207
- usa, 122
- waiver, 110, 112
- world, 122
- world2, 122
- x (aes\_position), 14
- xend (aes\_position), 14
- xlab (labs), 121
- xlim, 208
- xmax (aes\_position), 14
- xmin (aes\_position), 14
- y (aes\_position), 14
- yend (aes\_position), 14

`ylab(labs)`, [121](#)  
`ylim(xlim)`, [208](#)  
`ymax(aes_position)`, [14](#)  
`ymin(aes_position)`, [14](#)