# Package 'ggedit'

October 13, 2022

**Type** Package

**Title** Interactive 'ggplot2' Layer and Theme Aesthetic Editor

**Version** 0.3.1

**Date** 2020-06-01

**Author** Jonathan Sidi [aut, cre]

**Maintainer** Jonathan Sidi <yonicd@gmail.com>

**Description** Interactively edit 'ggplot2' layer and theme aesthetics definitions.

**Depends** R (>= 3.2.0), ggplot2 (>= 3.0.0)

**Imports** dplyr (>= 1.0.0), plyr, tidyr, purrr, rlang, scales, rstudioapi, shiny, miniUI, shinyBS, colourpicker (>= 0.2), shinyAce, magrittr, utils, graphics, grid, tools, stats

**Suggests** testthat, covr

**License** MIT + file LICENSE

**URL** https://github.com/yonicd/ggedit

**BugReports** https://github.com/yonicd/ggedit/issues

**LazyData** true

**NeedsCompilation** no

**RoxygenNote** 7.1.0

**Repository** CRAN

**Date/Publication** 2020-06-02 11:50:06 UTC

## R topics documented:

---

`+.gg` *add layer*

---

## Description

add layer from gg object

## Usage

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

## Arguments

| | |
|---|---|
| `e1` | plot |
| `e2` | layer |

## Value

gg

## -.gg *remove layer*

### Description

remove layer from gg object

### Usage

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

### Arguments

| | |
|---|---|
| e1 | plot |
| e2 | layer |

### Value

gg

## as.ggedit *Try to coerce a ggplot object into a ggedit object*

### Description

Applied to ggplot objects to use the ploting function of ggedit.

### Usage

```
as.ggedit(plot)
```

### Arguments

| | |
|---|---|
| plot | an object |

### Value

an object of class ggedit

### See Also

print.ggedit, ggplot

## Examples

```
p  <- ggplot2::ggplot(iris,ggplot2::aes(x =Sepal.Length,y=Sepal.Width))

p1 <- p +
ggplot2::geom_point(ggplot2::aes(colour=Species)) +
ggplot2::geom_line()

p2 <- p +
ggplot2::geom_point() +
ggplot2::geom_smooth(method='loess')

p3 <- list(p1,p2)

p4 <- as.ggedit(p3)

p4
```

---

# as.gglist *recasts to gglist*

---

## Description

adds gglist class to object

## Usage

```
as.gglist(plot)

## S3 method for class 'gglist'
as.gglist(plot)

## S3 method for class 'list'
as.gglist(plot)

## S3 method for class 'ggplot'
as.gglist(plot)

## S3 method for class 'ggmatrix'
as.gglist(plot)
```

## Arguments

| | |
|---|---|
| plot | object |

## Value

gglist object

## cloneFacet *Clone ggplot facet object*

### Description

Clone ggplot facet object and return either a gg object or the script to parse and evaluate

### Usage

```
cloneFacet(obj, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| obj | gg facet wrap or facet_grid object |
| verbose | boolean, toggles to return object or script (TRUE), Default: FALSE |

### Value

gg object or script

### Examples

```
obj=ggplot2::facet_grid(a+b~c+d,scales = 'free',as.table = FALSE,switch = 'x',shrink = FALSE)

cloneFacet(obj)
cloneFacet(obj,verbose=TRUE)
```

## cloneLayer *Creates an independent copy of a ggplot layer object*

### Description

Creates copies of ggplot layers from within ggplot objects that are independent of the parent object.

### Usage

```
cloneLayer(l, verbose = FALSE, showDefaults = TRUE)
```

### Arguments

| | |
|---|---|
| l | ggplot2 object layer |
| verbose | toggle to control if the output is ggproto object (verbose==FALSE,default) or string of layer call (verbose==TRUE) |
| showDefaults | toggle to control if the verbose output shows all the input arguments passed to the proto object (if verbose==FALSE then ignored) |

## Details

ggplot objects are comprimsed of layer objects. Once compiled they are part of the plot object environment and if they are changed internally regardless of where they are in the (ie different environment) it will change the original plot. This function allows to create replicates of the plot layers and edit them independent of the original plot. When setting verbose to TRUE function returns the ggplot2 call as a string to paste in regular ggplot script to generate the layer.

## Value

ggproto or string object (conditional on verbose)

## Examples

```
p <- ggplot2::ggplot(iris,ggplot2::aes(x =Sepal.Length,y=Sepal.Width))

p <- p +
ggplot2::geom_point(ggplot2::aes(colour='Species')) +
ggplot2::geom_line()

p$layers[[1]]

newLayer <- cloneLayer(l=p$layers[[1]])

all.equal(p$layers[[1]],newLayer)

(v <- cloneLayer(l=p$layers[[1]],verbose=TRUE))

eval(parse(text=v))

all.equal(p$layers[[1]],eval(parse(text=v)))
```

---

# cloneRoot *clone root of ggplot object*

---

## Description

clone root of ggplot object and return either a new gg object or a string to parse and evaluate

## Usage

```
cloneRoot(obj, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| obj | ggplot object |
| verbose | boolean, toggles to return object or script (TRUE), Default: FALSE |

## Value

gg object or script

## See Also

capture.output

## Examples

```
cloneRoot(pList$pointSmooth)
cloneRoot(pList$pointSmooth,verbose=TRUE)
```

---

cloneScales *Clone ggplot2 scales from compiled ggplot objects*

---

## Description

Clone ggplot2 scales from compiled ggplot objects returns not the original call but the nested call

## Usage

```
cloneScales(p, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| p | ggplot object |
| verbose | boolean, if TRUE then returns script to eval(parse) if FALSE returns new compiled object Default: FALSE |

## Value

ggplot scale or script (depends on verbose)

## Examples

```
# p <- pList$pointSmooth+scale_colour_continuous(low='red')
# p
# pList$pointSmooth+cloneScales(p)
```

## compare *compare*

### Description

Compare differences theme object e1 (new theme) to theme object e2 (old theme)

### Usage

```
compare(e1, e2, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| e1 | theme object |
| e2 | theme object |
| verbose | logical to control if the output is a character of script or a theme object (default TRUE) |

### Value

theme object or character depending on verbose

### Examples

```
compare(ggplot2::theme_bw(),ggplot2::theme_get())
compare(ggplot2::theme_bw(),ggplot2::theme_get(),verbose=FALSE)
```

## dput.ggedit *Convert ggplot object to a string call*

### Description

Convert ggplot object to a string call

### Usage

```
dput.ggedit(obj, file = "")
```

### Arguments

| | |
|---|---|
| obj | compiled ggplot object |
| file | either a character string naming a file or a connection. " " indicates output to the console, Default: " " |

## Value

character

## Examples

```
pList$pointSmooth #original compiled plot

this.gg <- dput.ggedit(pList$pointSmooth) #dput the plot

writeLines(this.gg) #show the output

eval(parse(text=this.gg)) #recompile the plot

#add theme change
p <- pList$pointSmooth+theme(panel.background = element_rect(fill='green'))

this.gg<-dput.ggedit(p) #dput the plot

writeLines(this.gg) #show the output

eval(parse(text=this.gg)) #recompile the plot
```

---

# ggedit *Interactive shiny gadget for editing ggplot layers and themes.*

---

## Description

Shiny gadget that takes an input ggplots and populates a user interface with objects that let the user update aesthetics of layers and theme elements.

## Usage

```
ggedit(p.in, ...)
```

## Arguments

| | |
|---|---|
| p.in | ggplot plot object or list of objects |
| ... | options that are passed to ggedit |

## Details

The user can start the gadget using the console `ggedit(plotobj)` or through the Addins menu in Rstudio.

If you are using the the Addin option highlight on the editor window the ggplot object and then click the addin.

**Options to pass to ggedit**

**viewer** shiny viewer options. It can be either paneViewer (default with minHeight=1000), dialogViewer, browserViewer

**verbose** logical to control if the output includes script for layers and themes calls for parsing to create objects (default, verbose=TRUE)

**showDefaults** toggle to control if the verbose output shows all the input arguments passed to the proto object (if verbose==FALSE then ignored)

**width,height** dimensions of the renderPlot where the active plot is displayed

Once the gadget is running the list of plots are shown in a grid and a number of objects will appear above them.

**Action buttons**

Cancel:

Returns a NULL object

Done:

Returns the list decribed below.

**Dropdown list**

Navigates through the plots in the input list. If the input list is a named list the names will be in the dropdown. The plot chosen is termed as the "active plot"

**Radio buttons**

The options to choose in the radio buttons are the layer names in the active plot.

**Links**

Update Plot Layer:

A pop up window will appear and be populated with aesthetic elements found in the layer chosen from the radio buttons. The layer is cloned using `cloneLayer` creating a layer independent of the original plot. If the aesthetic is a factor the values will be shown in dropdown lists. If it is numeric it will be shown in a slider. If it is a factor colour/fill aesthetic the `colourPicker` package will allow to choose from the full pallete of colours. If the continuous colour/fill aesthetic a dropdown list will be shown with different palletes

Update Plot Theme:

A popup modal will appear populated with the theme elements found in the active plot. Each element will appear as having a value or empty depending if it was defined or not. The user can change or fill in any element with valid values and any textboxes left empty will use ggplot defaults.

Update Grid Theme:

Copies the theme of the active plot to the other plots in the list

Update Global Theme:

Copies the theme of the active plot to the session theme and all plots created outside of the gadget will have this theme.

View Layer Code:

Opens an ace editor to compare the active layer initial script call and the updated script call.

The ggplot objects returned (layers and themes) can be used on any ggplot object.

## Value

List of elements

**updatedPlots** list containing updated ggplot objects

**updatedLayers** For each plot a list of updated layers (ggproto) objects

**UpdatedLayersElements** For each plot a list elements and their values in each layer

**UpdatedLayerCalls** For each plot a list of scripts that can be run directly from the console to create a layer

**updatedScales** For each plot a list of updated scale objects

**UpdatedScalesCalls** For each plot a list of scripts that can be run directly from the console to create a scale object

**updatedThemes** For each plot a list of updated theme objects

**UpdatedThemeCalls** For each plot a list of scripts that can be run directly from the console to create a theme

## See Also

cloneLayer,rgg,ggplot,colourPicker

## Examples

```
p <- ggplot2::ggplot(iris,ggplot2::aes(x =Sepal.Length,y=Sepal.Width))

p <- p +
ggplot2::geom_point(ggplot2::aes(colour=Species)) +
ggplot2::geom_line()

if(interactive()){
## Not run:
pnew <- ggedit(p)
pnew

## End(Not run)
}
```

---

# ggedit_opts *Default and current ggedit options*

---

## Description

Options for functions in the ggedit package. When running R code, the object `ggedit_opts` (default options) is not modified by chunk headers (local chunk options are merged with default options), whereas `ggedit_opts_current` (current options) changes with different chunk headers and it always reflects the options for the current chunk.

Normally we set up the global options once in the first code chunk in a document using `ggedit_opts$set()`, so that all *latter* chunks will use these options. Note the global options set in one chunk will not affect the options in this chunk itself, and that is why we often need to set global options in a separate chunk.

## Usage

```
ggedit_opts

ggedit_opts_current
```

## Format

An object of class `list` of length 5.

An object of class `list` of length 5.

## Note

`ggedit_opts_current` is read-only in the sense that it does nothing if you call `ggedit_opts_current$set()`; you can only query the options via `ggedit_opts_current$get()`.

## Examples

```
ggedit_opts$get('themeDefaultClass')
```

---

`gggsave` *gggsave*

---

## Description

Wrapper of ggsave that saves ggplot or list of ggplot objects to image or pdf.

## Usage

```
gggsave(filename = "Rplot.pdf", plot = last_plot(), ...)
```

## Arguments

| | |
|---|---|
| `filename` | a character string giving the name of the file. If it is of the form "\|cmd", the output is piped to the command given by cmd. If it is NULL, then no external file is created (effectively, no drawing occurs), but the device may still be queried (e.g., for size of text). For use with onefile = FALSE give a C integer format such as "Rplot%03d.pdf" (the default in that case) |
| `plot` | ggplot or list of ggplots to save, defaults to last plot displayed |
| `...` | other arguments passed on to graphics device |

## Details

default output is to create one pdf regardless of size of list of plots inputted

## Value

nothing

### Examples

```
## Not run: gggsave(pList)
```

---

`gg_session` *retreive all functions that create ggroto layers or stats*

---

### Description

Retreive all functions that create ggroto layers or stats

### Usage

```
gg_session(gg_pkg = NULL)
```

### Arguments

| | |
|---|---|
| `gg_pkg` | character, package names to search in if NULL an internal search will look in loaded packages, Default: NULL |

### Value

data.frame

### Examples

```
gg_session('ggplot2')
```

---

`gg_vetting` *Backcheck what functions created the layers in a ggplot2 plot object*

---

### Description

Validate geoms with their unique attributes

### Usage

```
gg_vetting(p, obj = ggedit_opts$get("session_geoms"))
```

### Arguments

| | |
|---|---|
| `p` | gg, compiled ggplot object |
| `obj` | data.frame, contains the mapping of layer functions as created in gg_session(), Default: ggedit_opts$get('session_geoms') |

## Value

data.frame

## Examples

```
gg_vetting(pList$boxplotWrap)
lapply(pList,gg_vetting)
```

---

`is.ggedit` *Is the object of class ggedit*

---

## Description

Is the object of class ggedit. Very basic for many functions in the package.

## Usage

```
is.ggedit(x)
```

## Arguments

| | |
|---|---|
| x | an object |

## Value

logical - is the object of class ggedit

---

`is.gglist` *gglist*

---

## Description

Checks if object is a gglist object

## Usage

```
is.gglist(x)
```

## Arguments

| | |
|---|---|
| x | object |

## Value

boolean

## layersList *layersList*

### Description

Runs the ggplot_build function on the input and converts the output data objects into a nested list with the unique values of each of the aesthetic columns.

### Usage

```
layersList(obj)
```

### Arguments

| | |
|---|---|
| obj | ggplot2 plot object or list of plot objects |

### Value

list of aesthetics and their values for each layer in a plot

### Examples

```
p=ggplot2::ggplot(iris,ggplot2::aes(x=Sepal.Length,y=Sepal.Width))
p=p+ggplot2::geom_point(ggplot2::aes(colour=Species))+ggplot2::geom_line()
p
p.list=layersList(p)
p.list
```

## pList *List of plots for ggedit examples*

### Description

List of ggplot objects for examples in ggedit.

### Usage

```
pList
```

### Format

An object of class "list"

### Details

list includes a geom_point, geom_point+facet_wrap, geom_boxplot+facet_wrap, geom_point+geom_line, geom_point+geom_smooth, geom_point+geom_line+facet_wrap, geom_point+geom_line+facet_grid

## print.ggedit *Print ggedit objects*

### Description

Plots lists of ggplot2 plot objects layout functionality.

### Usage

```
## S3 method for class 'ggedit'
print(x, layout = NULL, byrow = FALSE, ...)
```

### Arguments

| | |
|---|---|
| x | list of ggplot2 plot objects |
| layout | matrix, layout of plots like in layout, if NULL then a default layout is used, Default: NULL |
| byrow | boolean, argument passed to default layout (when layout is NULL), used to transpose the output. |
| ... | not used |

### Examples

```
p <- as.gglist(pList[1:2])
p

p1 <- p+geom_hline(aes(yintercept=3))
p1

print(p1,byrow=TRUE)

print(p1,layout = matrix(c(2,2,NA,1),ncol=2))
```

## proto_features *ggplot2 layer proto extraction*

### Description

Extract geom, stat and position protos from a ggplot2 layer

### Usage

```
proto_features(l)
```

### Arguments

| | |
|---|---|
| l | ggproto |

### Value

data.frame

### Examples

```
proto_features(ggplot2::geom_smooth())
proto_features(ggplot2::annotation_logticks())
```

---

## remove_geom *Remove a layer from a compiled ggplot2 object.*

---

### Description

Removes specified layers from a ggplot object.

### Usage

```
remove_geom(p, geom, idx = NULL)
```

### Arguments

| | |
|---|---|
| p | ggplot2 plot object |
| geom | character string of the name of the layer to remove |
| idx | numeric of which index of geom to remove, Default: 1 |

### Examples

```
p <- ggplot2::ggplot(iris,ggplot2::aes(x =Sepal.Length,y=Sepal.Width))
p <- p+ggplot2::geom_point(ggplot2::aes(colour=Species))+ggplot2::geom_line()
p
pnew <- p%>%remove_geom('point',1)
pnew
```

## rgg *Remove and replace ggplot2 layers.*

### Description

Removes specified layers from a ggplot object and gives the option to replace them with a new layer. This layer can be either a geom object created from regular ggplot functions or an output from the ggedit gadget. In the latter case the layers are found in the updatedLayers object in the ggedit output.

### Usage

```
rgg(p, oldGeom, oldGeomIdx = 1, newLayer = NULL)
```

### Arguments

| | |
|---|---|
| p | ggplot2 plot object |
| oldGeom | character string of the name of the layer to remove |
| oldGeomIdx | numeric of which index of OldGeom to remove (default is 1) |
| newLayer | ggplot layer or list of layers |

### Examples

```
p <- ggplot2::ggplot(iris,ggplot2::aes(x =Sepal.Length,y=Sepal.Width))

p <- p+ggplot2::geom_point(ggplot2::aes(colour=Species))+ggplot2::geom_line()

p

p%>%rgg('point',1)

if( interactive() ){
x <- ggedit(p)
pnew <- p%>%rgg('point',1,x$updatedLayers[[1]])
pnew
}
```

## summary.ggedit *Print to console verbose outputs of objects of class ggedit*

### Description

function to tidy the ggedit output to single script calls for each plot

### Usage

```
## S3 method for class 'ggedit'
summary(object, ...)
```

### Arguments

| | |
|---|---|
| object | ggedit object |
| ... | used to pass show.structure |

### Details

use show.structure (boolean) to control if the output shows the full data.frame structure or just a placeholder [data.frame], Default: FALSE

---

## %>% *magrittr forward-pipe operator*

---

### Description

Pipe an object forward into a function or call expression.

### Usage

```
lhs %>% rhs
```

### Arguments

| | |
|---|---|
| lhs | A value or the magrittr placeholder. |
| rhs | A function call using the magrittr semantics. |

### Details

**Using %>% with unary function calls**
When functions require only one argument, `x %>% f` is equivalent to `f(x)` (not exactly equivalent; see technical note below.)

**Placing lhs as the first argument in rhs call**
The default behavior of %>% when multiple arguments are required in the rhs call, is to place lhs as the first argument, i.e. `x %>% f(y)` is equivalent to `f(x, y)`.

**Placing lhs elsewhere in rhs call**
Often you will want lhs to the rhs call at another position than the first. For this purpose you can use the dot (`.`) as placeholder. For example, `y %>% f(x, .)` is equivalent to `f(x, y)` and `z %>% f(x, y, arg = .)` is equivalent to `f(x, y, arg = z)`.

**Using the dot for secondary purposes**
Often, some attribute or property of lhs is desired in the rhs call in addition to the value of lhs

itself, e.g. the number of rows or columns. It is perfectly valid to use the dot placeholder several times in the `rhs` call, but by design the behavior is slightly different when using it inside nested function calls. In particular, if the placeholder is only used in a nested function call, `lhs` will also be placed as the first argument! The reason for this is that in most use-cases this produces the most readable code. For example, `iris %>% subset(1:nrow(.) %% 2 == 0)` is equivalent to `iris %>% subset(., 1:nrow(.) %% 2 == 0)` but slightly more compact. It is possible to overrule this behavior by enclosing the `rhs` in braces. For example, `1:10 %>% {c(min(.), max(.))}` is equivalent to `c(min(1:10), max(1:10))`.

**Using %>% with call- or function-producing `rhs`**
It is possible to force evaluation of `rhs` before the piping of `lhs` takes place. This is useful when `rhs` produces the relevant call or function. To evaluate `rhs` first, enclose it in parentheses, i.e. `a %>% (function(x) x^2)`, and `1:10 %>% (call("sum"))`. Another example where this is relevant is for reference class methods which are accessed using the `$` operator, where one would do `x %>% (rc$f)`, and not `x %>% rc$f`.

**Using lambda expressions with `%>%`**
Each `rhs` is essentially a one-expression body of a unary function. Therefore defining lambdas in magrittr is very natural, and as the definitions of regular functions: if more than a single expression is needed one encloses the body in a pair of braces, `{ rhs }`. However, note that within braces there are no "first-argument rule": it will be exactly like writing a unary function where the argument name is `"."` (the dot).

**Using the dot-place holder as `lhs`**
When the dot is used as `lhs`, the result will be a functional sequence, i.e. a function which applies the entire chain of right-hand sides in turn to its input. See the examples.

## Technical notes

The magrittr pipe operators use non-standard evaluation. They capture their inputs and examines them to figure out how to proceed. First a function is produced from all of the individual right-hand side expressions, and then the result is obtained by applying this function to the left-hand side. For most purposes, one can disregard the subtle aspects of magrittr's evaluation, but some functions may capture their calling environment, and thus using the operators will not be exactly equivalent to the "standard call" without pipe-operators.

Another note is that special attention is advised when using non-magrittr operators in a pipe-chain (`+`, `-`, `$`, etc.), as operator precedence will impact how the chain is evaluated. In general it is advised to use the aliases provided by magrittr.

## See Also

%<>%, %T>%, %$%

## Examples

```
# Basic use:
iris %>% head
```

```
# Use with lhs as first argument
iris %>% head(10)

# Using the dot place-holder
"Ceci n'est pas une pipe" %>% gsub("une", "un", .)

# When dot is nested, lhs is still placed first:
sample(1:10) %>% paste0(LETTERS[.])

# This can be avoided:
rnorm(100) %>% {c(min(.), mean(.), max(.))} %>% floor

# Lambda expressions:
iris %>%
{
  size <- sample(1:10, size = 1)
  rbind(head(., size), tail(., size))
}

# renaming in lambdas:
iris %>%
{
  my_data <- .
  size <- sample(1:10, size = 1)
  rbind(head(my_data, size), tail(my_data, size))
}

# Building unary functions with %>%
trig_fest <- . %>% tan %>% cos %>% sin

1:10 %>% trig_fest
trig_fest(1:10)
```

# Index