Package 'helfRlein'

June 9, 2018

Type Package

Date 2018-06-09

Title R-Helper functions

Version 0.0.0.9000

Description A usefull collection of R helper functions.

Depends R (>= 3.3.3)

Imports data.table (>= 1.9), igraph (>= 1.1.2), data.tree (>= 0.7.0), stringr (>= 1.2.0), RCurl

License MIT + file LICENSE

LazyData TRUE

RoxygenNote 6.0.1

Suggests testthat

R topics documented:

`helfRlein-package` *This package contains some R helper functions.*

Description

There is no other purpose to the whole package but to collect and distribute some of our helper functions which are used over and over again.

Author(s)

Jakob Gepp <jakob.gepp@statworx.com>

`burglr` *burglg - Stealing code from the web*

Description

With `burglr` you can source r-scripts from the web.

Usage

```
burglr(urls)
```

Arguments

`urls`	a character vector with web urls indicating the location of an r-script.

Details

Well, it is as easy as it sounds. The function takes web URL(s) as input and evaluates the input in your current R session.

Value

Visual Feedback if the sourcing was successful

Note

Version 1.0 - f871288d8eae25bcd1dd2bf506f5c47a7cbbc6e4

Author(s)

Andre Bleier

See Also

For more information and updates see here.

Examples

```
burglr(urls = "https://github.com/andrebleier/Xy/master/Xy.R")
```

clean_gc *Using gc multiple times*

Description

Cleans the memory by using `gc()` numerous times until.

Usage

```
clean_gc(num.gc = 100, threshold = 0.01, verbose = FALSE)
```

Arguments

`num.gc`	a numeric that indicates the maximum number of iterations.
`threshold`	a numeric with the percentage difference. If the change in memory size falls below the threshold, the function stops.
`verbose`	a boolean. If `TRUE` information about the run are printed.

Details

The function calls `gc()` until the difference in memory size falls below the `threshold`.

Author(s)

Jakob Gepp

Examples

```
clean_gc(verbose = TRUE)
```

dive *a debugging function to dive in*

Description

A function which simplifies debugging attempts of self-written R functions. The main objective of this function is to get default and pre-specified arguments of a function into a user-specified output. This output can either be the global environment, a list or a console print.

Usage

```
dive(x, return = "cons")
```

Arguments

`x`	a string with a function or apply call
`return`	a string either of type `"cons"`, `"env"` or `"list"`. Further information in the return section.

Details

A word of caution: If you have string arguments in functions like in the `paste`, you have to escape the quotation marks. Suppose we want to debug the `paste` function with `dive`. Then specifying the argument `x` won't be work with `x = "paste("Hello World")"`, since there will be unexpected symbols in `"paste("Hello World")"`. Thus the correct specification is: `x = "paste(\"Hello World\")"`.

Value

The function has several return options:

- `"cons"` prints the arguments into the console
- `"env"` evaluates the arguments in the global
- `"list"` returns a list with the arguments.

When using *apply debugging only the return `"cons"` and `"list"` are available, since there is not guess for the iterator.

Note

Version 1.0 - 09adcdad5a26fa0f8739434b36c02d106e340ed2

Author(s)

Andre Bleier

See Also

For more information and updates see here.

Examples

```
# Define a function
foo <- function(x = 3, y = 1, z = 1, type = "add") {
 if (type == "add") {
 OUT <- z+y+x
 } else if (type == "vec") {
 OUT <- c(z,y,x)
 } else {
 OUT <- list(z,y,x)
 }
return(OUT)
}
# Save the debug option into a string
my.debug <- "foo(x = 2, y = 3, z = 2)"
# Get the arguments with dive
dive(my.debug, return = "cons")

# Try dive with an apply function
my.apply.debug <- "lapply(c(1,3,5), FUN = foo, y = 6, z = 2)"
# Get the arguments with dive
dive(my.apply.debug, return = "cons")

# Try dive with a character argument
# Escape strings
```

```
my.string.debug <- "lapply(c(1,3,5), FUN = foo, y = 2, z = 1, type = \"vec\")"
dive(my.string.debug, return = "cons")
```

`get_network` *flowchart of R projects*

Description

With this function a network plot of the connections of the functions in a given path can be created.

Usage

```
get_network(dir, variations = c(" <- function", "<- function", "<-function"),
  pattern = "\\.R")
```

Arguments

`dir`	a path that includes the functions
`variations`	a character vector with the function's definition string. The default is c(" <- function", "<- function", "<-function").
`pattern`	a string with the file suffix - default is "\.R".

Value

Returns an object with the adjacency matrix `$matrix` and and igraph object `$igraph`.

Note

TODO: list with exclude files and comments ' ' in one line

TODO: maybe return plot

Author(s)

Jakob Gepp

See Also

For more information see our blog.

Examples

```
## Not run:
net <- get_network(dir = "R/")
g1 <- net$igraph
plot(g1)

## End(Not run)
```

get_sequence *get start and end indices of sequnces of patterns*

Description

Given a vector `x` and a `pattern`, the functions returns the start and end indices of the sequences with at least `minsize` repetitions of the `pattern`.

Usage

```
get_sequence(x, pattern, minsize = 2L)
```

Arguments

`x`	a vector
`pattern`	the pattern to look for
`minsize`	the minimum length of the repeating pattern

Value

Returns a matrix with the range of the sequence. Each row representes a sequence.

Author(s)

Jakob Gepp

Examples

```
get_sequence(x = c(0,1,1,0,0,0,0,3,0,0,1,0,423,0,0,0,0,0,1,0),
             pattern = 0,
             minsize = 4)

#      min max
# [1,]   4   7
# [2,]  14  18
```

intersect2 *intersect for multiple input vectors*

Description

Function to check the intersect within multiple vectors.

Usage

```
intersect2(...)
```

Arguments

`...`	vectors to check for intersect

Value

Returns the intersect of all given input vectors.

Author(s)

Jakob Gepp

Examples

```
intersect2(c(1:3), c(1:4), c(1:2))
# [1] 1 2
```

`multiplot` *combine multiple ggplots*

Description

Functions that allows to combine diffrent ggplots into one plot.

Usage

```
multiplot(..., plotlist = NULL, cols = 1, layout = NULL)
```

Arguments

`...`	multiple ggplots.
`plotlist`	a list with ggplots.
`cols`	numeric. Number of columns in the output plot.
`layout`	a matrix with the layout of the plots.

Details

ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects) - cols: Number of columns in layout - layout: A matrix specifying the layout. If present, 'cols' is ignored.

If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE), then plot 1 will go in the upper left, 2 will go in the upper right, and 3 will go all the way across the bottom.

Author(s)

Cookbook for R

See Also

This is copied from the Cookbook for R

na_omitlist *remove NA from list*

Description

This functions removes NAs from a list. With `recursive == TRUE` NAs within each list's elements are removed as well.

Usage

```
na_omitlist(y = list(), recursive = FALSE)
```

Arguments

y	a list
recursive	logical. If TRUE, NAs within the list's elements aree removed too.

Value

Returns the list without NAs.

Author(s)

Jakob Gepp

Examples

```
y <- list(c(1:3), letters[1:4], NA, c(1, NA), list(c(5:6, NA), NA, "A"))
na_omitlist(y, recursive = TRUE)

# [[1]]
# [1] 1 2 3
#
# [[2]]
# [1] "a" "b" "c" "d"
#
# [[3]]
# [1] 1
#
# [[4]]
# [[4]][[1]]
# [1] 5 6
#
# [[4]][[2]]
# [1] "A"
```

print_fs *print file structure*

Description

Prints the directories and files of the given path.

Usage

```
print_fs(path = ".", depth = 2L)
```

Arguments

path	a folder path.
depth	a positive integer with the depth of the folder structure.

Value

Either the file structure gets printed or returned.

Author(s)

Jakob Gepp

Examples

```
print_fs(path = ".")
```

strsplit *improved strsplit function*

Description

This functions uses strsplit and adds the possibility to split and keep the delimiter after or before the given split.

Usage

```
strsplit(x, split, type = "remove", perl = FALSE, ...)
```

Arguments

x	character vector, each element of which is to be split. Other inputs, including a factor, will give an error.
split	character vector (or object which can be coerced to such) containing regular expression(s) (unless fixed = TRUE) to use for splitting. If empty matches occur, in particular if split has length 0, x is split into single characters. If split has length greater than 1, it is re-cycled along x.
type	a charachter. Either to "remove" or keep the delimiter "before" or "remove" the split.
perl	logical. Should Perl-compatible regexps be used? Is TRUE for "before" and "remove".
...	other inouts for base::strsplit

Value

A list of the same length as x, the i-th element of which contains the vector of splits of x[i].

Note

TODO see issues for further advancements

Author(s)

Jakob Gepp

See Also

strsplit or stackoverflow for more details.

Examples

```
x <- c("3D/MON&SUN")
strsplit(x, "[/&]")
# [[1]]
# [1] "3D"  "MON" "SUN"

strsplit(x, "[/&]", type = "before")
# [[1]]
# [1] "3D"  "/MON" "&SUN"

strsplit(x, "[/&]", type = "after")
# [[1]]
# [1] "3D/"  "MON&" "SUN"
```

to_na *replace NaN and Inf with NA*

Description

Takes out NaN and Inf and replaces them with NA

Usage

```
to_na(x)
```

Arguments

x	vector

Value

Returns vector with with replaced NAvalues.

Note

— Idea for improvement

Add args to flexible select which scenarios should be set NA

- nan, infinite, other defined values

Author(s)

Daniel Luettgau

Examples

```
test <- list(a = c("a", "b", NA),
             b = c(NaN, 1,2, -Inf),
             c = c(TRUE, FALSE, NaN, Inf))

lapply(test, to_na)

## Output
# $a
# [1] "a" "b" NA

# $b
# [1] NA  1  2 NA

# $c
# [1]  1  0 NA NA
```

Index