

Package: mark (via r-universe)

September 7, 2024

Type Package

Title Miscellaneous, Analytic R Kernels

Version 0.8.0.9005

Maintainer Jordan Mark Barbone <jmbarbone@gmail.com>

Description Miscellaneous functions and wrappers for development in other packages created, maintained by Jordan Mark Barbone.

License MIT + file LICENSE

URL <https://github.com/jmbarbone/mark>,
<https://jmbarbone.github.io/mark/>

BugReports <https://github.com/jmbarbone/mark/issues>

Encoding UTF-8

LazyData true

Depends R (>= 3.6)

Imports cli, fs (>= 1.6.2), fuj (>= 0.2.1), magrittr (>= 2.0.1),
rlang, stats (>= 3.6), tools (>= 3.6), utils (>= 3.6)

Suggests bench (>= 1.1.1), bib2df (>= 1.1.1), crayon (>= 1.3.4), covr (>= 3.5.1), desc (>= 1.3.0), dplyr (>= 1.0.6), echo (>= 0.1.0), graphics (>= 3.6), haven, knitr (>= 1.30), rcmdcheck (>= 1.3.3), stringi (>= 1.5.3), spelling (>= 2.2), testthat (>= 3.0.0), tibble (>= 3.0.4), waldo (>= 0.2.5), withr (>= 2.3.0), xopen, yaml, jsonlite, arrow (>= 16.1.0)

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/testthat/parallel true

Language en-US

Repository <https://jmbarbone.r-universe.dev>

RemoteUrl <https://github.com/jmbarbone/mark>

RemoteRef HEAD

RemoteSha 8b57ee9ca54aadb2114e35b9c7d63163a77ad4ad

Contents

add_file_timestamp	4
are_identical	5
array_extract	6
as_ordered	6
base_alpha	7
base_n	8
betwixt	9
blank_values	10
char2fact	11
checkOptions	11
chr_split	12
clipboard	13
complete_cases	14
counts	15
date_from_partial	16
depth	17
detail	18
diff_time	19
drop_levels	21
ept	21
eval_named_chunk	22
expand_by	23
fact	23
fact2char	25
fact_na	26
fact_reverse	26
fct_expand_seq	27
file_copy_md5	28
file_info	28
file_name	29
file_utils	29
fizzbuzz	31
get_dir_max_number	32
get_dir_recent_date	32
get_recent_dir	33
get_recent_file	33
get_version	34
glob	34
handlers	35
import	36
insert	37
is_dir	38
labels	38
limit	40
lines_of_r_code	40
list2df	41

list_environments	42
logic_ext	43
ls_ext	45
make_sf	45
mark	46
match_arg	46
match_param	47
md5	49
median2	49
merge_list	50
multi_grepl	51
na_assignments	52
na_cols	53
normalize	54
norm_path	55
note	56
not_available	57
omit_na	58
package_available	59
percentile_rank	59
print.mark_bib_df	60
print.pseudo_id	61
print_c	61
process_bib_dataframe	62
pseudo_id	63
quiet_stop	64
range2	64
read_bib	65
recode_by	66
reindex	67
remove_na	68
remove_null	69
round_by	69
row_bind	70
rscript	71
save_source	71
set_names0	72
simpleTimeReport	72
sort_by	73
sort_names	74
source_files	74
source_to_env	75
sourcing	75
str_extract_date	76
str_slice	77
switch-ext	77
tableNA	79
that	81

todos	81
to_boolean	83
to_row_names	84
tryn	84
t_df	85
unique_rows	86
unlist0	86
use_author	87
utils-paste	88
vap	89
vector2df	90
within_call	90
with_par	91
write_file_md5	91

Index **93**

add_file_timestamp	<i>Add file timestamp</i>
--------------------	---------------------------

Description

Adds a timestamp to a file

Usage

```
add_file_timestamp(
  x,
  ts = Sys.time(),
  format = "%Y-%m-%d %H%M%S",
  sep = " "
)
```

Arguments

x	A vector of files
ts	A single timestamp or vector of timestamps (default: Sys.time())
format	A format to be applied to the times; set to NULL to skip formatting
sep	A character vector of length 1 to separate the timestamp from the file name

Value

The full name paths with the appended time stamp

Examples

```
file1 <- tempfile(fileext = ".txt")
file2 <- tempfile()

add_file_timestamp(file1)
add_file_timestamp(file2)

file.remove(file1, file2)
```

are_identical	<i>Identical extensions</i>
---------------	-----------------------------

Description

Extensions for the use of `base::identical()`

Usage

```
are_identical(..., params = NULL)
```

Arguments

...	Vectors of values to compare, element-wise of equal length
params	Additional params (as a named list of arguments for base::identical)

Value

A logical vector of TRUE/FALSE of equal length of each ... vector

Examples

```
x <- y <- z <- 1:5
y[2] <- 3L
z[5] <- NA_integer_

identical(x, y)      # compare entire vector
are_identical(x, y)  # element-wise
are_identical(x, y, z) # 3 or more vectors
```

array_extract	<i>Array extract</i>
---------------	----------------------

Description

Extract dimensions from an array

Usage

```
array_extract(.arr, ..., default = "1")
```

Arguments

.arr	An array
...	A named list by array dimension number and the value
default	The default dimension index

Value

A value from the array arr

Examples

```
x <- array(rep(NA, 27), dim = c(3, 3, 3))
x[1, 2, 3] <- TRUE
x[1, 2, 3]
x
array_extract(x, `2` = 2, `3` = 3)
```

as_ordered	<i>Ordered</i>
------------	----------------

Description

As ordered

Usage

```
as_ordered(x)

## Default S3 method:
as_ordered(x)
```

Arguments

x	A vector of values
---	--------------------

Details

Simple implementation of ordered. If `x` is ordered it is simply returned. If `x` is a factor the ordered class is added. Otherwise, `x` is made into a factor with `fact()` and then the ordered class is added. Unlike just `fact`, `ordered` will replace the NA levels with `NA_integer_` to work appropriately with other functions.

Value

An ordered vector

See Also

[fact\(\)](#)

Other factors: [char2fact\(\)](#), [drop_levels\(\)](#), [fact\(\)](#), [fact2char\(\)](#), [fact_na\(\)](#)

Examples

```
x <- c("a", NA, "b")
x <- fact(x)
str(x) # NA is 3L

y <- x
class(y) <- c("ordered", class(y))
max(y)
max(y, na.rm = TRUE) # returns NA -- bad

# as_ordered() removes the NA level
x <- as_ordered(x)
str(x)
max(x, na.rm = TRUE) # returns b -- correct
```

base_alpha

Alpha base

Description

Base 26 conversion with letters

Usage

```
base_alpha(x, base = 26)
```

Arguments

`x` A string of letters. Non characters are removed.
`base` A numeric

Value

A vector of integers

Examples

```
base_alpha("AB")
base_alpha("XFD")
base_alpha(c("JMB", "Jordan Mark", "XKCD"))
sum(base_alpha(c("x", "k", "c", "d")))
```

base_n

Base N conversion

Description

Convert between base numbers

Usage

```
base_n(x, from = 10, to = 10)
```

Arguments

x	A vector of integers
from, to	An integer base to convert to and from; from must be an integer from 1 to 10 and to can currently only be 10.

Value

The A vector of integers converted from base from to base to

Examples

```
base_n(c(24, 22, 16), from = 7)
```

betwixt	<i>Betwixt boundaries</i>
---------	---------------------------

Description

Compare a vector betwixt (between) other values

Usage

```
between_more(x, left, right, type = c("gele", "gel", "gle", "gl"))
```

```
betwixt(x, left = NULL, right = NULL, bounds = c("[", "[", "]", "()"))
```

Arguments

x	A numeric vector of values
left, right	Boundary values. For <code>betwixt()</code> , when NULL no comparison is made for that boundary. When both are NULL, x is just returned.
type	Abbreviation for the evaluation of left on right (see details)
bounds	Boundaries for comparisons of left and right (see details)

Details

type `` , bounds“ can be one of the below:

g,(is greater than (>))

ge,[greater than or equal to (>=))

l,) less than (<))

le,[] less than or equal to (<=))

Note: `between_more()` may be deprecated in the future in favor of just `betwixt()`

Value

A logical vector

Examples

```
between_more(2:10, 2, 10, "gl")
betwixt(2:10, 2, bounds = "()")
between_more(10, 2, 10, "gle")
betwixt(2:10, bounds = "[")
betwixt(1:5, c(3, 3, 2, 2, 1), 5)
```

blank_values	<i>Blank values</i>
--------------	---------------------

Description

Detect *blank* values; select, remove columns that are entirely *blank*

Usage

```
is_blank(x, na_blank = FALSE, ws = TRUE)
select_blank_cols(x, na_blank = FALSE, ws = TRUE)
remove_blank_cols(x, na_blank = FALSE, ws = TRUE)
is_blank_cols(x, names = TRUE, na_blank = FALSE, ws = TRUE)
```

Arguments

x	An object, or data.frame for *_cols() functions
na_blank	Logical, if TRUE treats NA values as <i>blank</i>
ws	Logical, when TRUE treats elements that are entirely <i>whitespace</i> as blanks
names	Logical, if TRUE (default) will return column names as names of vector

Details

Blank values are values that do not contain any text

Value

- `is_blank()` a logical vector indicating *blank* elements in x
- `select_blank_cols()` x with only columns that are all *blank*
- `remove_blank_cols()` x without columns of only *blank*
- `is_blank_cols()` a logical vector: TRUE all rows of column are *blank*, otherwise FALSE

char2fact	<i>Character to factor</i>
-----------	----------------------------

Description

Converts characters to factors

Usage

```
char2fact(x, n = 5)

## Default S3 method:
char2fact(x, n = 5)

## S3 method for class 'character'
char2fact(x, n = 5)

## S3 method for class 'factor'
char2fact(x, n = 5)

## S3 method for class 'data.frame'
char2fact(x, n = 5)
```

Arguments

x	A vector of characters
n	The limit to the number of unique values for the factor

See Also

[fact2char\(\)](#)

Other factors: [as_ordered\(\)](#), [drop_levels\(\)](#), [fact\(\)](#), [fact2char\(\)](#), [fact_na\(\)](#)

checkOptions	<i>Check options</i>
--------------	----------------------

Description

For each name in x checks the current option value and reports if there is a difference in a message. This does not change the options

Usage

```
checkOptions(x)
```

Arguments

x A named list of new options

Details

Checks and reports on options

Value

Invisible, a list of the current options from options()

Examples

```
op <- options()

x <- list(width = -20, warning.length = 2, probably_not_a_real_option = 2)
checkOptions(x)
# pointless, but shows that no messages are given
identical(options(), checkOptions(options()))

options(op)
```

chr_split

Character split

Description

Split apart a string by each character

Usage

```
chr_split(x)
```

Arguments

x A vector of strings to split

Value

A character vector of length nchar(x)

Examples

```
chr_split("split this")
```

clipboard	<i>Write to and read from the clipboard</i>
-----------	---

Description

Wrappers for working with the clipboard

Usage

```
write_clipboard(x, ...)

## Default S3 method:
write_clipboard(x, ...)

## S3 method for class 'data.frame'
write_clipboard(x, sep = "\t", row.names = FALSE, ...)

## S3 method for class 'matrix'
write_clipboard(x, sep = "\t", ...)

## S3 method for class 'list'
write_clipboard(x, sep = "\t", ...)

read_clipboard(method = c("default", "data.frame", "tibble"), ...)
```

Arguments

x	An object
...	Additional arguments sent to methods or to <code>utils::write.table()</code>
sep	the field separator string. Values within each row of x are separated by this string.
row.names	either a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written.
method	Method switch for loading the clipboard

Details

As these functions rely on `utils::readClipboard()` and `utils::writeClipboard()` they are only available for Windows 10. For copying and pasting floats, there may be some rounding that can occur.

Value

`write_clipboard()` None, called for side effects `read_clipboard()` Either a vector, `data.frame`, or `tibble` depending on the method chosen. Unlike `utils::readClipboard()`, an empty clipboard value returns NA rather than ""

Examples

```

# Will only run on windows
if (Sys.info()[["sysname"]] == "Windows") {
  foo <- function(x) {
    write_clipboard(x)
    y <- read_clipboard()
    res <- all.equal(x, y)
    if (isTRUE(res)) return("All equal")
    print(x)
    print(y)
  }
  foo(1:4)
  foo(seq(-1, 1, .02))
  foo(Sys.Date() + 1:4)

  # May have some rounding issues
  x <- "0.316362437326461129"
  write_clipboard(x)
  res <- as.character(read_clipboard())
  all.equal(x, res)
  x; res
}

```

complete_cases

Complete cases

Description

Return completed cases of a data.frame

Usage

```
complete_cases(data, cols = NULL, invert = FALSE)
```

Arguments

data	A data.frame
cols	Colnames or numbers to remove NA values from; NULL (default) will use all columns
invert	Logical, if TRUE will return incomplete cases

Value

A data.frame

Examples

```
x <- data.frame(
  a = 1:5,
  b = c(1, NA, 3, 4, 5),
  c = c(1, NA, NA, 4, 5)
)

complete_cases(x)
complete_cases(x, invert = TRUE) # returns the incomplete rows
complete_cases(x, "a")
complete_cases(x, "b")
complete_cases(x, "c")
```

counts

Count observations by unique values

Description

Variables will be return by the order in which they appear. Even factors are shown by their order of appearance in the vector.

There are 2 methods for counting vectors. The default method uses `base::tabulate()` (the workhorse for `base::table()` with a call to `pseudo_id()` to transform all inputs into integers. The logical method counts TRUE, FALSE and NA values, which is much quicker.

Usage

```
counts(x, ...)

## S3 method for class 'data.frame'
counts(x, cols, sort = FALSE, ..., .name = "freq")

props(x, ...)

## Default S3 method:
props(x, sort = FALSE, na.rm = FALSE, ...)

## S3 method for class 'data.frame'
props(x, cols, sort = FALSE, na.rm = FALSE, ..., .name = "prop")
```

Arguments

x	A vector or data.frame
...	Arguments passed to other methods
cols	A vector of column names or indexes
sort	Logical, if TRUE will sort values (not counts) before returning. For factors this will sort by factor levels. This has no effect for logical vectors, which already return in the order of FALSE, TRUE, NA.

.name	The name of the new column
na.rm	If TRUE will remove NA values from proportions

Details

Get counts or proportions of unique observations in a vector or columns in a data.frame

Value

A named vector of integers or doubles (for counts, and props, respectively) or data.frame with columns for each column chosen and the .name chosen for the summary

Examples

```
x <- sample(1:5, 10, TRUE)
counts(x)
props(x)

x <- quick_df(list(
  a = c("a", "c", "a", "c", "d", "b"),
  b = c("a", "a", "a", "c", "c", "b"),
  c = c("a", "a", "a", "c", "b", "b")
))

counts(x, "a")
counts(x, c("a", "b", "c"))
props(x, 2)
props(x, 1:3)

props(c(1, 1, 3, NA, 4))
props(c(1, 1, 3, NA, 4), na.rm = TRUE)
```

date_from_partial *Partial dates*

Description

Derive a date vector from a partial date string

Usage

```
date_from_partial(
  x,
  format = "ymd",
  method = c("min", "max"),
  year_replacement = NA_integer_
)
```


Arguments

x	A vector of dates written as characters
format	Format order of the date (accepts only combinations of 'y', 'm', and 'd')
method	Method for reporting partial dates as either the earliest possible date ("min") or the latest possible date ("max"); dates with missing days will be adjusted accordingly to the month and, if needed, the leap year
year_replacement	(Default: NA_integer_) If set, will use this as a replacement for dates that contain missing years

Details

Takes a character as an argument and attempts to create a date object when part of the date string is missing.

Value

A vector of Dates

Examples

```
x <- c("2020-12-17", NA_character_, "", "2020-12-UN", "2020-12-UN",
      "2019-Unknown-00", "UNK-UNK-UNK", "1991-02-UN", " ",
      "2020January20")
data.frame(
  x = x,
  min = date_from_partial(x),
  max = date_from_partial(x, method = "max"),
  year = date_from_partial(x, year_replacement = 1900)
)
```

depth

Depth

Description

Functions to extract the 'depth' of an object

Usage

```
depth(x, ...)
```

Default S3 method:
depth(x, ...)

S3 method for class 'list'
depth(x, ...)

Arguments

x An object
 ... Possible additional arguments passed to methods (not in use)

Details

This function does not count an empty lists (`list()`) as a level or NULL objects.

Value

A single integer

Examples

```
a <- c(1, 2, 3)
depth(a) # Vectors are 1L

b <- list(a = 1, b = list(list(1)))
depth(b)
```

detail

Details an object

Description

Provides details about an object

Usage

```
detail(x, ...)
```

Default S3 method:
`detail(x, factor_n = 5L, ...)`

S3 method for class 'data.frame'
`detail(x, factor_n = 5L, ...)`

Arguments

x An object
 ... Additional arguments passed to methods
 factor_n An integer threshold for making factors; will convert any character vectors with factor_n or less unique values into a fact; setting as NA will ignore this

Examples

```
x <- sample(letters[1:4], 10, TRUE)
detail(x)

df <- quick_df(list(
  x = x,
  y = round(runif(10), 2),
  z = Sys.Date() + runif(10) * 100
))

detail(df)
```

diff_time

Diff time wrappers

Description

Wrappers for computing diff times

Usage

```
diff_time(
  x,
  y,
  method = c("secs", "mins", "hours", "days", "weeks", "months", "years", "dyears",
    "wyears", "myears"),
  tzx = NULL,
  tzy = tzx
)

diff_time_days(x, y, ...)
diff_time_weeks(x, y, ...)
diff_time_hours(x, y, ...)
diff_time_mins(x, y, ...)
diff_time_secs(x, y, ...)
diff_time_months(x, y, ...)
diff_time_years(x, y, ...)
diff_time_dyears(x, y, ...)
diff_time_wyears(x, y, ...)
```

```
diff_time_myyears(x, y, ...)
```

Arguments

<code>x, y</code>	Vectors of times
<code>method</code>	A method to report the difference in units of time (see Units section)
<code>tzx, tzy</code>	time zones (see Time zones section)
<code>...</code>	Additional arguments passed to <code>diff_time()</code>

Details

A few significant differences exist with these functions * The class of the object returned is no longer `diff_time` (but does print) with the `diff_time` method. This makes the exporting process easier as the data will not have to be converted back to numeric * `diff_time()` computes the difference of `time1 - time2`, but the inverse feels a bit more nature: time difference from `x` to `y` * Additional units can be used (detailed below) * Differences can be sensitive to time zones if time zones are passed to the `tz` parameter as a character vector

Value

A `diff_time` vector, object

Units

Units can be used beyond those available in `base::diff_time()`. Some of these use assumptions in how units of time should be standardized and can be changed in the corresponding options. Any of these can be calculated with `base::diff_time()` through using `units = "days"` but the `dtime` class will print out with these specifications into the console for less potential confusion.

months Months by number of days `mark.days_in_month` (defaults: 30)

years Years by number of days `mark.days_in_year` (defaults: 365)

dyears Years by number of days `mark.days_in_year` (defaults: 365) (same as years)

myears Years by number of days in a month `mark.days_in_month` (defaults: 30)

wyears Years by number of weeks in a year `mark.weeks_in_year` (defaults: 52)

Time zones

Time zones can be passed as either a numeric vector of GMT/UTC offsets (the number of seconds from GMT) or as a character vector. If the letter, these need to conform with values from `base::OlsonNames()`.

A default timezone can be set with `options(mark.default_tz = .)`. The value can either be a numeric

drop_levels	<i>Drop levels</i>
-------------	--------------------

Description

Drop unused levels of a factor

Usage

```
drop_levels(x, ...)  
  
## S3 method for class 'data.frame'  
drop_levels(x, ...)  
  
## S3 method for class 'fact'  
drop_levels(x, ...)  
  
## S3 method for class 'factor'  
drop_levels(x, ...)
```

Arguments

x	A factor or data.frame
...	Additional arguments passed to methods (not used)

See Also

[base::droplevels](#)

Other factors: [as_ordered\(\)](#), [char2fact\(\)](#), [fact\(\)](#), [fact2char\(\)](#), [fact_na\(\)](#)

ept	<i>Parse and evaluate text</i>
-----	--------------------------------

Description

A wrapper for `eval(parse(text = .))`

Usage

```
ept(x, envir = parent.frame())
```

Arguments

x	A character string to parse
envir	The environment in which to evaluate the code

Value

The evaluation of x after parsing

eval_named_chunk	<i>Evaluate a Named Chunk</i>
------------------	-------------------------------

Description

Evaluate a named chunk from an Rmd file.

Usage

```
eval_named_chunk(rmd_file, label_name)
```

Arguments

rmd_file	Absolute path to rmd file
label_name	Name of label

Value

The value from the evaluated code chunk

Examples

```
temp_rmd <- tempfile(fileext = ".rmd")

text <- '
```{r not this label}
print("that is wrong")
```

```{r hello label}
text <- "hello, world"
print(text)
print(TRUE)
```

```{r another label}
warning("wrong label")
```
'

## Not run:
writeLines(text, con = temp_rmd)

eval_named_chunk(temp_rmd, "hello label")
# [1] "hello, world"
# [1] TRUE
```

```
file.remove(temp_rmd)

## End(Not run)
```

| | |
|-----------|-------------------------|
| expand_by | <i>Expands a vector</i> |
|-----------|-------------------------|

Description

Expands vector x by y

Usage

```
expand_by(x, y, expand = c("x", "y", "intersect", "both"), sort = FALSE)
```

Arguments

| | |
|--------|--|
| x, y | Vectors |
| expand | Character switch to expand or keep only the values that intersect, all values in x or y, or retain all values found. |
| sort | Logical, if TRUE will sort by names in output |

Value

A vector with expanded

Examples

```
x <- letters[c(3:2, 5, 9)]
y <- letters[c(1:4, 8)]
expand_by(x, y, "x")
expand_by(x, y, "y")
expand_by(x, y, "intersect")
expand_by(x, y, "both")
```

| | |
|------|---------------|
| fact | <i>Factor</i> |
|------|---------------|

Description

Quickly create a factor

Usage

```
fact(x)

## Default S3 method:
fact(x)

## S3 method for class 'character'
fact(x)

## S3 method for class 'numeric'
fact(x)

## S3 method for class 'integer'
fact(x)

## S3 method for class 'Date'
fact(x)

## S3 method for class 'POSIXt'
fact(x)

## S3 method for class 'logical'
fact(x)

## S3 method for class 'factor'
fact(x)

## S3 method for class 'fact'
fact(x)

## S3 method for class 'pseudo_id'
fact(x)

## S3 method for class 'haven_labelled'
fact(x)
```

Arguments

x A vector of values

Details

`fact()` can be about 5 times quicker than `factor()` or `as.factor()` as it doesn't bother sorting the levels for non-numeric data or have other checks or features. It simply converts a vector to a factor with all unique values as levels with NAs included.

`fact.factor()` will perform several checks on a factor to include NA levels and to check if the levels should be reordered to conform with the other methods. The `fact.fact()` method simple returns `x`.

Value

A vector of equal length of `x` with class `fact` and `factor`. If `x` was ordered, that class is added in between.

level orders

The order of the levels may be adjusted to these rules depending on the class of `x`:

character The order of appearance

numeric/integer/Date/POSIXt By the numeric order

logical As TRUE, FALSE, then NA if present

factor Numeric if levels can be safely converted, otherwise as they are

See Also

[as_ordered\(\)](#)

Other factors: [as_ordered\(\)](#), [char2fact\(\)](#), [drop_levels\(\)](#), [fact2char\(\)](#), [fact_na\(\)](#)

fact2char

Factor to character

Description

Convert factor columns to characters in a `data.frame`

Usage

```
fact2char(data, threshold = 10)
```

Arguments

`data` A `data.frame`

`threshold` A threshold for the number of levels to be met/exceeded for transforming into a character

Value

The `data.frame` `data` with factors converted by the rule above

See Also

[char2fact\(\)](#)

Other factors: [as_ordered\(\)](#), [char2fact\(\)](#), [drop_levels\(\)](#), [fact\(\)](#), [fact_na\(\)](#)

| | |
|---------|---------------------|
| fact_na | fact <i>with</i> NA |
|---------|---------------------|

Description

Included NA values into `fact()`

Usage

```
fact_na(x, remove = FALSE)
```

Arguments

| | |
|--------|--|
| x | A fact or object coerced to fact |
| remove | If TRUE removes NA value from the fact levels and uniques attributes |

Details

This re-formats the x value so that NAs are found immediately within the object rather than accessed through its attributes.

Value

A fact vector

See Also

Other factors: [as_ordered\(\)](#), [char2fact\(\)](#), [drop_levels\(\)](#), [fact\(\)](#), [fact2char\(\)](#)

| | |
|--------------|----------------------------|
| fact_reverse | <i>Fact reverse levels</i> |
|--------------|----------------------------|

Description

Reverse the levels of a fact

Usage

```
fact_reverse(x)
```

Arguments

| | |
|---|--|
| x | A fact object (or passed to fact()) |
|---|--|

fct_expand_seq *Factor Expand by Sequence*

Description

Expands an ordered factor from one level to another

Usage

```
fct_expand_seq(  
  x,  
  min_lvl = min(x, na.rm = TRUE),  
  max_lvl = max(x, na.rm = TRUE),  
  by = 1L  
)
```

Arguments

| | |
|---------|-------------------------------------|
| x | An ordered factor |
| min_lvl | The start of the level sequence |
| max_lvl | The end of the level sequence |
| by | Integer, number of steps in between |

Details

Defaults for `min_lvl` and `max_lvl` are the minimum and maximum levels in the ordered vector `x`.

Value

An ordered vector

Examples

```
x <- ordered(letters[c(5:15, 2)], levels = letters)  
fct_expand_seq(x)  
fct_expand_seq(x, "g", "s", 3L) # from "g" to "s" by 3  
fct_expand_seq(x, "g", "t", 3L) # same as above  
  
# from the first inherit level to the last observed  
fct_expand_seq(x, min(levels(x)))
```

| | |
|---------------|--------------------------------------|
| file_copy_md5 | <i>File copy with md5 hash check</i> |
|---------------|--------------------------------------|

Description

File copy with md5 hash check

Usage

```
file_copy_md5(path, new_path, overwrite = NA, quiet = FALSE)
```

Arguments

| | |
|-----------|--|
| path | A character vector of one or more paths. |
| new_path | A character vector of paths to the new locations. |
| overwrite | When NA, only saves if the md5 hashes do not match. Otherwise, see fs::file_copy() . |
| quiet | When TRUE, suppresses messages from md5 checks. |

Value

The path(s) of the new file(s), invisibly. When `overwrite` is NA, the paths will be returned with two addition attributes, "changed", a logical vector indicating whether the file was changed (NA for when the file is new), and "md5sum", a list of the md5sums of the old and new md5 sums.

| | |
|-----------|-------------------------------|
| file_info | <i>File information utils</i> |
|-----------|-------------------------------|

Description

Other utility functions for dealing with files

Usage

```
newest_file(x)
newest_dir(x)
oldest_file(x)
oldest_dir(x)
largest_file(x)
smallest_file(x)
```

Arguments

x A vector of file paths

Value

A full file path

| | |
|-----------|------------------|
| file_name | <i>File name</i> |
|-----------|------------------|

Description

Basename of file without extension

Usage

```
file_name(x, compression = FALSE)
```

Arguments

x character vector giving file paths.
compression logical: should compression extension '.gz', '.bz2' or '.xz' be removed first?

Value

The file name of the path without the extension

| | |
|------------|--|
| file_utils | <i>Open a file using windows file associations</i> |
|------------|--|

Description

Opens the given files(s)

Usage

```
open_file(x)

file_open(x)

shell_exec(x)

list_files(
  x = ".",
  pattern = utils::glob2rx(glob),
```

```

    glob = NULL,
    ignore_case = FALSE,
    all = FALSE,
    negate = FALSE,
    basename = FALSE
)

list_dirs(
  x = ".",
  pattern = NULL,
  ignore_case = FALSE,
  all = FALSE,
  basename = FALSE,
  negate = FALSE
)

```

Arguments

| | |
|---------------|--|
| x | A character vector of paths |
| pattern, glob | Pattern to search for files. glob is absorbed into pattern, through <code>utils::glob2rx()</code> . |
| ignore_case | logical. Should pattern-matching be case-insensitive? |
| all | a logical value. If FALSE, only the names of visible files are returned (following Unix-style visibility, that is files whose name does not start with a dot). If TRUE, all file names will be returned. |
| negate | Logical, if TRUE will inversely select files that do not match the provided pattern |
| basename | If TRUE only searches pattern on the basename, otherwise on the entire path |

Details

`open_file` is an alternative to `shell.exec()` that can take take multiple files. `list_files` and `list_dirs` are mostly wrappers for `fs::dir_ls()` with preferred defaults and pattern searching on the full file path.

`file_open` is simply an alias.

Value

- `open_file()`, `shell_exec()`: A logical vector where TRUE successfully opened, FALSE did not and NA did not try to open (file not found)
- `list_files()`, `list_dirs()`: A vector of full paths

| | |
|----------|------------------|
| fizzbuzz | <i>Fizz Buzz</i> |
|----------|------------------|

Description

For when someone asked you to do something you've done before, you can argue that the quickest way to do it is to just take the work someone else did and utilize that. No reason to reinvent the wheel.

Usage

```
fizzbuzz(n, show_numbers = TRUE)
```

```
fizzbuzz_lazy(n)
```

```
.fizzbuzz_vector
```

Arguments

| | |
|--------------|-----------------------|
| n | The number of numbers |
| show_numbers | If TRUE shows no |

Format

An object of class character of length 1000000.

Details

Multiples of 3 are shown as "Fizz"; multiples of 5 as "Buzz"; multiple of both (i.e., 15) are "FizzBuzz". `fizzbuzz_lazy()` subsets the `.fizzbuzz_vector` object, which is a solution with default parameters up to 1e6

Value

A character vector of 1, 2, Fizz, 3, Buzz, etc

Examples

```
fizzbuzz(15)
fizzbuzz(30, show_numbers = FALSE)
cat(fizzbuzz(30), sep = "\n")

# show them how fast your solution is:
if (package_available("bench")) {
  bench::mark(fizzbuzz(1e5), fizzbuzz_lazy(1e5))
}
```

`get_dir_max_number` *Get recent directory by number name*

Description

Finds the directory where the number is the greatest. This can be useful for when folders are created as run IDs.

Usage

```
get_dir_max_number(x)
```

Arguments

`x` The directory to look in

Value

A full path to a directory

`get_dir_recent_date` *Get recent directory by date*

Description

Looks at the directories and assumes the date

Usage

```
get_dir_recent_date(x = ".", dt_pattern = NULL, dt_format = NULL, all = FALSE)
```

Arguments

`x` A directory
`dt_pattern` A pattern to be passed to filter for the directory
`dt_format` One or more formats to try
`all` Logical, if TRUE will recursively search for directories

Value

A full path to a directory

| | |
|----------------|-----------------------------|
| get_recent_dir | <i>Get recent directory</i> |
|----------------|-----------------------------|

Description

Finds the recent subdirectory in a directory.

Usage

```
get_recent_dir(x = ".", ...)
```

Arguments

| | |
|-----|--|
| x | The root directory |
| ... | Additional arguments passed to list_dirs() |

Value

The full path of the most recent directory

| | |
|-----------------|------------------------|
| get_recent_file | <i>Get recent file</i> |
|-----------------|------------------------|

Description

A function where you can detect the most recent file from a directory.

Usage

```
get_recent_file(x, exclude_temp = TRUE, ...)
```

Arguments

| | |
|--------------|---|
| x | The directory in which to search the file |
| exclude_temp | Logical, if TRUE tries to remove temp Windows files |
| ... | Additional arguments passed to list_files() |

Value

The full name of the most recent file from the stated directory

| | |
|-------------|-----------------------------|
| get_version | <i>Get and bump version</i> |
|-------------|-----------------------------|

Description

Will read the DESCRIPTION file and to get and adjust the version

bump_date_version() will not check if the version is actually a date. When the current version is the same as today's date(equal by character strings) it will append a .1.

Usage

```
get_version()

bump_version(version = NULL)

bump_date_version(version = NULL)

update_version(version = NULL, date = FALSE)
```

Arguments

| | |
|---------|---|
| version | A new version to be added; default of NULL will automatically update. |
| date | If TRUE will use a date as a version. |

Details

Get and bump package version for dates

Value

- get_version(): A package_version
- bump_version(): None, called for its side-effects
- bump_date_version(): None, called for its side-effects
- update_version(): None, called for its side-effects

| | |
|------|--------------------------|
| glob | <i>Wildcard globbing</i> |
|------|--------------------------|

Description

Helper function for globbing character vectors

Usage

```
glob(x, pattern = NULL, value = TRUE, ...)
```

Arguments

| | |
|------------|--|
| x | A vector of characters |
| pattern | Wildcard globbing pattern |
| value, ... | Additional parameters passed to grep. Note: value is by default TRUE; when NA, ... is passed to grepl. |

Examples

```
x <- c("apple", "banana", "peach", "pear", "orange")
glob(x, "*e")
glob(x, "pea*", value = FALSE)
glob(x, "*an*", value = NA)

path <- system.file("R", package = "mark")
glob(list.files(path), "r*")
```

handlers

*Handlers***Description**

Catch and report handlers

Usage

```
has_warning(x, FUN, ...)
has_error(x, FUN, ...)
has_message(x, FUN, ...)
get_warning(x, FUN, ..., .null = TRUE)
get_message(x, FUN, ..., .null = TRUE)
get_error(x, FUN, ..., .null = TRUE)
```

Arguments

| | |
|-------|--|
| x | A vector |
| FUN | A function |
| ... | Additional params passed to FUN |
| .null | Logical, if FALSE will drop NULL results (for get_*) |

Details

These functions can be used to catch whether an evaluation will return an error or warning without raising.

Value

The `has_*`() functions will return TRUE/FALSE for if the handler is found in the execution of the code. The `get_*`() functions provide the text of the message

References

Function for *catching* has been adapted from <https://stackoverflow.com/a/4952908/12126576>

Examples

```
has_warning(c(1, "no"), as.integer)
#   1   no
# FALSE TRUE

get_warning(c(1, "no"), as.integer)

# drop NULLs
get_warning(c(1, "no"), as.integer, .null = FALSE)

foo <- function(x) {
  stopifnot(x > 0)
  x
}

has_error(c(1, 0, 2), foo)
#   1   0   2
# FALSE TRUE FALSE

get_error(c(1, 0, 2), foo)

# drop NULLs
get_error(c(1, 0, 2), foo, .null = FALSE)
```

import

Import

Description

Import a single function from a package

Usage

```
import(pkg, fun, overwrite = FALSE)
```

Arguments

| | |
|-----------|--|
| pkg | String, name of the package |
| fun | String, fun name of the function |
| overwrite | Logical, if TRUE and fun is also found in the current environment, will overwrite assignment |

Value

None, called for side effects

Examples

```
# assigns `add` -- test with caution
import("magrittr", "add")
```

| | |
|--------|---------------|
| insert | <i>Insert</i> |
|--------|---------------|

Description

Insert values at a position

Usage

```
insert(x, positions, values)
```

Arguments

| | |
|-----------|--|
| x | A vector of values |
| positions | Integer of positions of x to insert values |
| values | A vector of values to insert into x |

Value

A vector with the intended values inserted

Examples

```
insert(letters[1:5], c(2, 4), c("X", "Y"))
```

| | |
|--------|--------------------------|
| is_dir | <i>Is File/Directory</i> |
|--------|--------------------------|

Description

Is the path a file/directory?

Usage

```
is_dir(x)
is_file(x)
```

Arguments

x A vector of file paths

Details

These are essentially taken from `utils::file_test()` for `op = '-d'` and `op = -f` but separated.

Value

A logical vector

| | |
|--------|-------------------------|
| labels | <i>Dataframe labels</i> |
|--------|-------------------------|

Description

Assign labels to a vector or data.frame.

Usage

```
assign_labels(x, ...)

## Default S3 method:
assign_labels(x, label, ...)

## S3 method for class 'data.frame'
assign_labels(
  x,
  ...,
  .missing = c("error", "warn", "skip"),
  .ls = rlang::list2(...)
)
```

```

get_labels(x)

## Default S3 method:
get_labels(x)

## S3 method for class 'data.frame'
get_labels(x)

view_labels(x, title)

remove_labels(x, ...)

## Default S3 method:
remove_labels(x, ...)

## S3 method for class 'data.frame'
remove_labels(x, cols, ...)

```

Arguments

| | |
|-----------------------|--|
| <code>x</code> | A vector of data.frame |
| <code>...</code> | One or more unquoted expressed separated by commas. If assigning to a data.frame, <code>...</code> can be replaced with a data.frame where the first column is the targeted colname and the second is the desired label. |
| <code>label</code> | A single length string of a label to be assigned |
| <code>.missing</code> | A control setting for dealing missing columns in a list; can be set to error to stop() the call, warn to provide a warning, or skip to silently skip those labels. |
| <code>.ls</code> | A named list of columns and labels to be set if <code>...</code> is empty |
| <code>title</code> | Title for the viewer window – if not supplemented will show as <code>paste0(as.character(substitute(x)) - " - Labels")</code> |
| <code>cols</code> | A character vector of column names; if missing will remove the label attribute across all columns |

Details

When labels are assigned to a data.frame they can make viewing the object (with `View()` inside Rstudio). The `view_labels()` has a call to `View()` inside and will retrieve the labels and show them in the viewer as a data.frame.

Value

A labelled vector or data.frame

Examples

```

labs <- assign_labels(
  iris,

```

```

  Sepal.Length = "cms",
  Sepal.Width  = "cms",
  Petal.Length = "cms",
  Petal.Width  = "cms",
  Species      = "Iris ..."
)

labs$dummy <- ""
get_labels(labs) # shows label as <NA> for dummy column

labs0 <- remove_labels(labs, c("Sepal.Length", "Sepal.Width"))
get_labels(labs0) # No labels for Sepal.Length and Sepal.Width

```

limit

Limit

Description

Limit a numeric vector by lower and upper bounds

Usage

```
limit(x, lower = min(x), upper = max(x))
```

Arguments

| | |
|-------|--|
| x | A numeric vector |
| lower | A lower limit (as $x < \text{lower}$) |
| upper | An upper limit (as $x > \text{higher}$) |

Value

The vector x with lower and upper as the minimum, maximum values

lines_of_r_code

Lines of R code

Description

Find the total number of lines of R code

Usage

```
lines_of_r_code(x = ".", skip_empty = TRUE)
```


Arguments

| | |
|------------|---|
| x | Directory to search for files |
| skip_empty | Logical, if TRUE will not count lines that are empty or only contain a bracket or quotation mark. |

Details

Tries to read each file in the directory that ends in .R or .r and sums together. Files that fail to read are not counted.

Value

An integer for the number of lines in all applicable files

Examples

```
lines_of_r_code(system.file())
lines_of_r_code(system.file(), skip_empty = FALSE)
```

| | |
|---------|---------------------------|
| list2df | <i>List to data.frame</i> |
|---------|---------------------------|

Description

Converts a list object into a data.frame

Usage

```
list2df(x, name = "name", value = "value", show_NA, warn = TRUE)
```

Arguments

| | |
|-------------|--|
| x | A (preferably) named list with any number of values |
| name, value | Names of the new key and value columns, respectively |
| show_NA | Ignored; if set will trigger a warning |
| warn | Logical; if TRUE will show a warning when |

Details

Unlike `base::list2DF()`, `list2df()` tries to format the data.frame by using the names of the list as values rather than variables. This creates a longer form list that may be more tidy.

Value

a data.frame object with columns "name" and "value" for the names of the list and the values in each

Examples

```
x <- list(a = 1, b = 2:4, c = letters[10:20], "unnamed", "unnamed2")
list2df(x, "col1", "col2", warn = FALSE)

if (getRversion() >= as.package_version('4.0')) {
# contrast with `base::list2DF()` and `base::as.data.frame()`
x <- list(a = 1:3, b = 2:4, c = letters[10:12])
list2df(x, warn = FALSE)
list2DF(x)
as.data.frame(x)
}
```

| | |
|-------------------|--|
| list_environments | <i>List all environments and objects</i> |
|-------------------|--|

Description

Functions to list out all environments and objects

Usage

```
environments()
```

```
ls_all(all.names = FALSE)
```

```
objects_all(all.names = FALSE)
```

Arguments

`all.names` a logical value. If TRUE, all object names are returned. If FALSE, names which begin with a `'.'` are omitted.

Details

`environments()` is basically a printing wrapper for `base::search()`

`ls_all()` and `objects_all()` can be used retrieved all objects from all environments in the `search()` path, which may print out a large result into the console.

Value

- `environments()`: Invisibly, a character vector of environment names
- `ls_all()`, `objects_all()`: A named list for each of the environments the `search()` path with all the objects found in that environment

`logic_ext`*Logic - Extension'*

Description

All functions take logical or logical-like (i.e., 1, 0, or NA as integer or doubles) and return logical values.

Extensions to the base logical operations to account for NA values.

`base::isTRUE()` and `base::isFALSE()` will only return single length TRUE or FALSE as it checks for valid lengths in the evaluation. When needing to check over a vector for the presence of TRUE or FALSE and not being held back by NA values, `is_true` and `is_false` will always provide a TRUE FALSE when the vector is logical or return NA if the vector `x` is not logical.

`%or%` is just a wrapper for `base::xor()`

Usage

```
is_true(x)

## Default S3 method:
is_true(x)

## S3 method for class 'logical'
is_true(x)

is_false(x)

## Default S3 method:
is_false(x)

## S3 method for class 'logical'
is_false(x)

x %xor% y

OR(..., na.rm = FALSE)

AND(..., na.rm = FALSE)

either(x, y)

is_boolean(x)

none(..., na.rm = FALSE)
```

Arguments

| | |
|--------------------|--|
| <code>x, y</code> | A vector of logical values. If NULL will generate a warning. If not a logical value, will return NA equal to the vector length |
| <code>...</code> | Vectors or a list of logical values |
| <code>na.rm</code> | Logical, if TRUE will ignore NA |

Details

Logical operations, extended

Value

- `is_true()`, `is_false()`, `either()`, `%or%`, `AND()`, `OR()`: A logical vector, equal length of `x` (or `y` or of all `...` lengths)
- `is_boolean()`: TRUE or FALSE
- `none()`: TRUE, FALSE, or NA

Examples

```
x <- c(TRUE, FALSE, NA)
y <- c(FALSE, FALSE, TRUE)
z <- c(TRUE, NA, TRUE)
isTRUE(x)
is_true(x)
isFALSE(x)
is_false(x)
x %xor% TRUE
TRUE %xor% TRUE
TRUE %xor% FALSE
NA %xor% FALSE
OR(x, y, z)
OR(x, y, z, na.rm = TRUE)
AND(x, y, z)
AND(x, y, z, na.rm = TRUE)
either(x, FALSE)
either(TRUE, FALSE)
either(FALSE, NA)
either(TRUE, NA)
none(x)
none(x & y, na.rm = TRUE)
is_boolean(x)
is_boolean(c(1L, NA_integer_, 0L))
is_boolean(c(1.01, 0, -1))
```

| | |
|--------|----------------------------------|
| ls_ext | <i>List Objects - extensions</i> |
|--------|----------------------------------|

Description

List Objects - extensions

Usage

```
ls_dataframe(pattern, all.names = FALSE, envir = parent.frame())
```

```
ls_function(pattern, all.names = FALSE, envir = parent.frame())
```

```
ls_object(pattern, all.names = FALSE, envir = parent.frame())
```

Arguments

| | |
|-----------|---|
| pattern | an optional regular expression . Only names matching pattern are returned. glob2rx can be used to convert wildcard patterns to regular expressions. |
| all.names | a logical value. If TRUE, all object names are returned. If FALSE, names which begin with a '.' are omitted. |
| envir | an alternative argument to name for specifying the environment. Mostly there for back compatibility. |

Value

A character vector of names

| | |
|---------|----------------------------------|
| make_sf | <i>Make system file function</i> |
|---------|----------------------------------|

Description

Simple wrapper for package specific function for internal packages

Usage

```
make_sf(package)
```

Arguments

| | |
|---------|-------------------------|
| package | The name of the package |
|---------|-------------------------|

| | |
|------|-------------|
| mark | <i>mark</i> |
|------|-------------|

Description

Miscellaneous, Analytic R Kernels

Author(s)

Maintainer: Jordan Mark Barbone <jmbarbone@gmail.com> (0000-0001-9788-3628) [copyright holder]

See Also

Useful links:

- <https://github.com/jmbarbone/mark>
- <https://jmbarbone.github.io/mark/>
- Report bugs at <https://github.com/jmbarbone/mark/issues>

| | |
|-----------|------------------------|
| match_arg | <i>Match arguments</i> |
|-----------|------------------------|

Description

This function is essentially a clear version of `base::match.arg()` which produces a cleaner warning message and does not restrict the `table` param to character vectors only.

Usage

```
match_arg(x, table)
```

Arguments

| | |
|-------|--------------------|
| x | An argument |
| table | A table of choices |

Details

Match arguments

Value

A single value from `x` matched on `table`

See Also[match_param\(\)](#)**Examples**

```
x <- c("apple", "banana", "orange")
match_arg("b", x)

# Produces error
try(match_arg("pear", x))

foo <- function(x, op = c(1, 2, 3)) {
  op <- match_arg(op)
  x / op
}

foo(10, 3)

# Error
try(foo(1, 0))
```

 match_param

Match params

Description

Much like [base::match.arg\(\)](#) with a few key differences:

- Will not perform partial matching
- Will not return error messages with ugly quotation marks

Usage

```
match_param(
  param,
  choices,
  null = TRUE,
  partial = getOption("mark.match_param.partial", FALSE),
  multiple = FALSE,
  simplify = TRUE
)
```

Arguments

| | |
|---------|---|
| param | The parameter |
| choices | The available choices; named lists will return the name (a character) for when matched to the value within the list element. A list of formula objects (preferred) retains the LHS of the formula as the return value when matched to the RHS of the formula. |

| | |
|----------|--|
| null | If TRUE allows NULL to be passed a param |
| partial | If TRUE allows partial matching via pmatch() |
| multiple | If TRUE allows multiple values to be returned |
| simplify | If TRUE will simplify the output to a single value |

Details

Param matching for an argument

Value

A single value from param matched on choices

See Also

[match_arg\(\)](#)

Examples

```
fruits <- function(x = c("apple", "banana", "orange")) {
  match_param(x)
}

fruits()          # apple
try(fruits("b")) # must be exact fruits("banana")

pfruits <- function(x = c("apple", "apricot", "banana")) {
  match_param(x, partial = TRUE)
}
pfruits()          # apple
try(pfruits("ap")) # matchParamMatchError
pfruits("app")     # apple

afruits <- function(x = c("apple", "banana", "orange")) {
  match_param(x, multiple = TRUE)
}

afruits() # apple, banana, orange

# can have multiple responses
how_much <- function(x = list(too_few = 0:2, ok = 3:5, too_many = 6:10)) {
  match_param(x)
}

how_much(1)
how_much(3)
how_much(9)

# use a list of formulas instead
ls <- list(1L ~ 0:1, 2L, 3L ~ 3:5)
sapply(0:5, match_param, choices = ls)
```

| | |
|-----|--|
| md5 | <i>Compute the MD5 hash of an object</i> |
|-----|--|

Description

Wrapper for calling `tools::md5sum()` on objects rather than files.

Usage

```
md5(x)
```

Arguments

x An object

Details

All x objects are [serialized](#) to a temporary file before `tools::md5sum()` is called.

Value

A md5sum object

Examples

```
md5("hello")
md5(1:10)
md5(data.frame(a = 1:10, b = letters[1:10]))
```

| | |
|---------|----------------------|
| median2 | <i>Median (Q 50)</i> |
|---------|----------------------|

Description

Median as the 50th quantile with an option to select quantile algorithm

Usage

```
median2(x, type = 7, na.rm = FALSE)
```

```
q50(x, type = 7, na.rm = FALSE)
```

Arguments

| | |
|-------|--|
| x | numeric vector whose sample quantiles are wanted, or an object of a class for which a method has been defined (see also ‘details’). NA and NaN values are not allowed in numeric vectors unless na.rm is TRUE. |
| type | an integer between 1 and 9 selecting one of the nine quantile algorithms detailed below to be used. |
| na.rm | logical; if true, any NA and NaN’s are removed from x before the quantiles are computed. |

Details

q50 is an alias for median2

Value

See stats::quantile()

See Also

[stats::quantile\(\)](#)

Examples

```
set.seed(42)
x <- rnorm(100)
median(x)          # 0.08979677
median2(x, type = 7) # 0.08979677 - default type is 7
median2(x, type = 3) # 0.08976065
```

merge_list

Merge lists

Description

Merge lists with different or intersecting names

Usage

```
merge_list(x, y, keep = c("x", "y"), null = c("ignore", "drop", "keep")[1:2])
```

Arguments

| | |
|------|--|
| x, y | Lists to merge |
| keep | When matching names are found, from which object should the values be retained; "x" retains values from x, "y" retains values from y. |
| null | Method for handling NULL values. When two values are passed, they will be applied to x and y respectively. When a single value is passed, it will be applied to both x and y. <ul style="list-style-type: none"> • "ignore": NULL values are ignored. When passes to x, the NULL values will be retained if they are not overridden by y. • "drop": NULL values are dropped from merge and will not appear in the output. • "keep": NULL values are retained in the output and can override other values. |

Examples

```
x <- list(a = 1, b = 2, c = NULL, d = NULL)
y <- list(a = 2, b = NULL, c = 3)

# compared to:
utils::modifyList(x, y)
utils::modifyList(x, y, keep.null = TRUE)

merge_list(x, y)
merge_list(x, y, keep = "y")
merge_list(x, y, null = "drop")
```

multi_grepl

Multiple searching

Description

Multiple search pattern searches

Usage

```
multi_grepl(x, patterns, ..., simplify = TRUE)

multi_grep(x, patterns, ..., simplify = TRUE)
```

Arguments

| | |
|----------|---|
| x | Passed to <code>base::grepl()</code> |
| patterns | A list or vector of patterns to search across x; if named value returned will be the name of the pattern – otherwise the position. Pattern match reported will be the first in the list that is found |
| ... | Additional arguments passed to <code>base::grepl()</code> |
| simplify | if FALSE will return a list of all matches, otherwise the first match found |

Value

The name or position of the pattern that is matched

Examples

```
x <- c("apple", "banana", "lemon")
multi_grepl(x, c("a" = "^[ab]", "b" = "lem"))
multi_grepl(x, c("a" = "^[ab]", "b" = "q"))           # lemon not matches on either
multi_grepl(x, c("a" = "^[ab]", "b" = "e"))           # apple matches "a" before "b"
multi_grepl(x, c("a" = "^[ab]", "b" = "e"), simplify = FALSE) # shows all matches
multi_grepl(x, c("^[ab]", "e"))                       # returned as positions
multi_grepl(x, c("^[ab]", "e"), simplify = FALSE)
```

| | |
|----------------|------------------------|
| na_assignments | <i>NA at positions</i> |
|----------------|------------------------|

Description

Converts select elements of a vector into NAs

This is how the end results are

- NA_at and NA_if require a suitable index value (`x[y] <- NA`)
 - NA_at expects `y` (or the result of function `y`) to be integers
 - NA_if expects `y` (or the result of function `y`) to be logical
- NA_in and NA_out expect some values to match on
 - NA_in checks `x[x %in% y] <- NA`
 - NA_out checks `x[x %out% y] <- NA` (see [match_ext](#))

Usage

```
NA_at(x, y, ...)
```

```
NA_if(x, y, ...)
```

```
NA_in(x, y, ...)
```

```
NA_out(x, y, ...)
```

Arguments

| | |
|------------------|--|
| <code>x</code> | A vector of values |
| <code>y</code> | Either a suitable value (see Details) or a function which accepts <code>x</code> as its first parameter and can return suitable values |
| <code>...</code> | Additional values passed to <code>y</code> (if <code>y</code> is a function) |

Details

Convert specific values to NA

Value

x with assigned NA values

See Also

Inspired by `dplyr::na_if()`

Examples

```
let <- ordered(letters[1:5])
NA_at(let, c(1, 3, 5)) # [1] <NA> b    <NA> d    <NA>
NA_if(let, let <= "b") # [1] <NA> <NA> c    d    e
NA_in(let, c("a", "c")) # [1] <NA> b    <NA> d    e
NA_out(let, c("a", "c")) # [1] a    <NA> c    <NA> <NA>
```

na_cols

Selecting NA columns

Description

Select or remove columns that are entirely NA

Usage

```
select_na_cols(x)
```

```
remove_na_cols(x)
```

```
is_na_cols(x, names = TRUE)
```

Arguments

x A data.frame

names Logical, if TRUE (default) will return column names as names of vector

Value

- `select_na_cols()` x with only columns that are all NA
- `remove_na_cols()` x without columns of only NA
- `is_na_cols()` a logical vector: TRUE all rows of column are NA, otherwise FALSE

| | |
|-----------|-------------------------|
| normalize | <i>Normalize values</i> |
|-----------|-------------------------|

Description

Normalizes values based on possible range and new bounds

Usage

```
normalize(x, ...)

## Default S3 method:
normalize(x, range = base::range(x, na.rm = TRUE), bounds = 0:1, ...)

## S3 method for class 'data.frame'
normalize(x, ...)
```

Arguments

| | |
|--------|--|
| x | An object that is (coercible to) double; data.frames are transformed |
| ... | Additional arguments passed to methods |
| range | The range of possible values of x. See details for more info. Defaults to the range of non-NA values |
| bounds | The new boundaries for the normalized values of x. Defaults to 0 and 1. |

Details

Parameters `range` and `bounds` are modified with `base::range()`. The largest and smallest values are then used to determine the minimum/maximum values and lower/upper bounds. This allows for a vector of more than two values to be passed.

The current implementation of `normalize.data.frame()` allows for list of parameters passed for each column. However, it is probably best suited for default values.

Value

x with transformed values where range values are transformed to bounds.

Examples

```
x <- c(0.23, 0.32, 0.12, 0.61, 0.26, 0.24, 0.23, 0.32, 0.29, 0.27)
data.frame(
  x = normalize(x),
  v = normalize(x, range = 0:2),
  b = normalize(x, bounds = 0:10),
  vb = normalize(x, range = 0:2, bounds = 0:10)
)
```

```
# maintains matrix
mat <- structure(c(0.24, 0.92, 0.05, 0.37, 0.19, 0.69, 0.43, 0.22, 0.85,
0.73, 0.89, 0.68, 0.57, 0.89, 0.61, 0.98, 0.75, 0.37, 0.24, 0.24,
0.34, 0.8, 0.25, 0.46, 0.03, 0.71, 0.79, 0.56, 0.83, 0.97), dim = c(10L, 3L))

mat
normalize(mat, bounds = -1:1)
normalize(as.data.frame(mat), bounds = -1:1)
```

| | |
|-----------|------------------------|
| norm_path | <i>Normalize paths</i> |
|-----------|------------------------|

Description

Normalize and check a vector of paths

Usage

```
norm_path(x = ".", check = FALSE, remove = check)
```

```
file_path(..., check = FALSE, remove = check)
```

```
user_file(..., check = FALSE, remove = check)
```

Arguments

| | |
|--------|---|
| x | A character vector of paths |
| check | Logical, if TRUE will check if the path exists and output a warning if it does not. |
| remove | Logical, if TRUE will remove paths that are not found |
| ... | Character vectors for creating a path |

Value

A vector of full file paths

| | |
|------|-----------------------------------|
| note | <i>Append a note to an object</i> |
|------|-----------------------------------|

Description

An alternative to the `base::comment()`.

Usage

```
note(x) <- value
set_note(x, value)
note(x)
```

Arguments

| | |
|-------|---|
| x | An object |
| value | The note to attach; if NULL will remove the note and the class noted from the object. |

Details

When the note is assigned to an object a new class will be added, `note`, so that a print function can call an S3 method. The print for this can be adjusted for its width by using the option `mark.note.width` which defaults to the option `width` when not set.

The type of object assigned to the note is not restricted, so user beware of odd prints or additional features added to the notes fun.

When assigning a note (with `note<-`, and its alias `set_note()`) the noted class is added to the object. This allows the `print.noted` class to be dispatched and for the note to be printed every time the object is called/printed and the next print method used. However, it will not be called when `not interactive()`

Value

- `note<-`, `set_note()` will return `x` (with the "note" attribute assigned)
- `note()` will retrieve the "note" attribute

Examples

```
x <- c("x", "k", "c", "d")
comment(x) <- "This is just a comment"
comment(x)

# Comment is intentionally hidden
x
note(x) <- "Just some random letters"
```



```
note(x)

# Note is now present every time
x

# Assigning `NULL` will remove note (and class)
note(x) <- NULL
note(x) # NULL
x      # No more note
```

| | |
|---------------|---------------------------|
| not_available | <i>Make not available</i> |
|---------------|---------------------------|

Description

Create NA vectors

Usage

```
not_available(type = "logical", length = 0L)
```

```
set_not_available(type, value)
```

```
NA_Date_
```

```
NA_POSIXct_
```

```
NA_POSIXlt_
```

Arguments

| | |
|--------|--------------------------------------|
| type | Type of NA (see details) |
| length | Length of the vector |
| value | A value to return in not_available() |

Format

An object of class Date of length 1.

An object of class POSIXct (inherits from POSIXt) of length 1.

An object of class POSIXlt (inherits from POSIXt) of length 1.

Details

If length is a text it will search for an appropriate match.

Value

A vector of NA values

Examples

```
x <- not_available("Date", 3)
x
class(x)
```

omit_na

Omit NA values

Description

Omit NA values

Usage

```
omit_na(x)
```

Arguments

x A vector of values

Value

x which NA values removes and two attributes of integers: na which is the position of NA values, and valid for the position of non-NA values; empty positions reported as integer(0)

Examples

```
# Like stats::na.omit but always provides
x <- letters[1:5]
omit_na(x)
x[c(3, 5)] <- NA
omit_na(x)
```

| | |
|-------------------|--------------------------------------|
| package_available | <i>Check if package is available</i> |
|-------------------|--------------------------------------|

Description

A wrapped requireNamespace

Usage

```
package_available(namespace)
```

Arguments

namespace One or more packages to to require.

Value

- require_namespace(): None, called for side effects
- package_available(): Visibly, TRUE or FALSE

| | |
|-----------------|------------------------|
| percentile_rank | <i>Percentile rank</i> |
|-----------------|------------------------|

Description

The bounds of the percentile rank are > 0 and < 1 (see Boundaries)

A percentile rank here is the proportion of scores that are less than the current score.

$$PR = (c_L + 0.5f_i)/N$$

Where

c_L is the frequency of scores less than the score of interest

f_i is the frequency of the score of interest

Usage

```
percentile_rank(x, weights = times, times)
```

Arguments

x A vector of values to rank

weights, times A vector of the number of times to repeat x

Details

Computes a percentile rank for each score in a set.

Value

The percentile rank of x between 0 and 1 (see Boundaries)

Boundaries

While the percentile rank of a score in a set must be exclusively within the boundaries of 0 and 1, this function may produce a percentile rank that is exactly 0 or 1. This may occur when the number of values are so large that the value within the boundaries is too small to be differentiated.

Additionally, when using the `weights` parameter, if the lowest or highest number has a value of 0, the number will then have a theoretical 0 or 1, as these values are not actually within the set.

Examples

```
percentile_rank(0:9)
x <- c(1, 2, 1, 7, 5, NA_integer_, 7, 10)
percentile_rank(x)

if (package_available("dplyr")) {
  dplyr::percent_rank(x)
}

# with times
percentile_rank(7:1, c(1, 0, 2, 2, 3, 1, 1))
```

```
print.mark_bib_df      Print bib data frame
```

Description

Print bib dataframe, or as list

Usage

```
## S3 method for class 'mark_bib_df'
print(x, list = FALSE, ...)
```

Arguments

| | |
|-------------------|--|
| <code>x</code> | The <code>mark_bib_df</code> object |
| <code>list</code> | If TRUE will print as a list rather than the <code>data.frame</code> |
| <code>...</code> | Additional arguments passed to methods |

Value

`x`, invisibly, called for its side effects

| | |
|-----------------|------------------------|
| print.pseudo_id | <i>Print pseudo_id</i> |
|-----------------|------------------------|

Description

Print pseudo_id

Usage

```
## S3 method for class 'pseudo_id'
print(x, ..., all = FALSE)
```

Arguments

| | |
|-----|--|
| x | An object of class pseudo_id |
| ... | Not implemented |
| all | if TRUE will print all uniques. This is not recommend for many uniques as it will crowd the console output |

Value

x, invisibly. Called for its side effects.

See Also

[pseudo_id\(\)](#)

| | |
|---------|-------------------|
| print_c | <i>Print as c</i> |
|---------|-------------------|

Description

Prints a vector to paste into an R script

Usage

```
print_c(x = read_clipboard(), sorted = TRUE, null = TRUE)
```

Arguments

| | |
|--------|--|
| x | A vector (defaults to reading the clipboard) |
| sorted | If TRUE (default) applies <code>sort()</code> to x |
| null | If TRUE (default) adds NULL at the end of the <code>c()</code> print |

Details

This sorts (if set) and provides unique values for each element in `x` and prints them as a call to `c`. This can be useful for copying data that you want to save as a vector in an R script. The result is both called in `cat()` as well as copied to the clipboard.

Value

Invisibly, as a character vector, the object printed to the console

Examples

```
print_c(1:10)
print_c(letters[1:3])
print_c(month.abb)
```

process_bib_dataframe *Process bib values*

Description

Generates a data frame of values from bibs

Usage

```
process_bib_dataframe(categories, values, fields, keys)
```

Arguments

| | |
|-------------------------|----------------------|
| <code>categories</code> | A list of categories |
| <code>values</code> | A list of values |
| <code>fields</code> | a Vector of fields |
| <code>keys</code> | a Vector of keys |

Value

A wide data.frame with explicit NAs

| | |
|-----------|----------------------------------|
| pseudo_id | <i>Create an ID for a vector</i> |
|-----------|----------------------------------|

Description

Transforms a vector into an integer of IDs.

Usage

```
pseudo_id(x, ...)  
  
## S3 method for class 'pseudo_id'  
pseudo_id(x, ...)  
  
## Default S3 method:  
pseudo_id(x, na_last = TRUE, ...)  
  
## S3 method for class 'factor'  
pseudo_id(x, ...)
```

Arguments

| | |
|---------|---|
| x | A vector of values |
| ... | Additional arguments passed to methods |
| na_last | Logical if FALSE will not place NA at the end |

Value

A pseudo_id object where the integer value of the vector correspond to the position of the unique values in the attribute "uniques".

Examples

```
set.seed(42)  
(x <- sample(letters, 10, TRUE))  
(pid <- pseudo_id(x))  
attr(pid, "uniques")[pid]
```

| | |
|------------|-------------------|
| quiet_stop | <i>Quiet stop</i> |
|------------|-------------------|

Description

Quietly calls stop

Usage

```
quiet_stop()
```

Value

None, called for side effects

| | |
|--------|----------------|
| range2 | <i>Range 2</i> |
|--------|----------------|

Description

Employs `min()` and `max()`. However, `base::range()`, there is no argument for removing Inf values.

Usage

```
range2(x, na.rm = FALSE)
```

Arguments

| | |
|--------------------|---|
| <code>x</code> | A numeric (or character) vector (see Note in <code>base::min</code>) |
| <code>na.rm</code> | Logical, if TRUE removes missing values |

Value

A numeric vector of length 2 of the minimum and maximum values, respectively

Examples

```
x <- rep(1:1e5, 100)
system.time(rep(range(x), 100))
system.time(rep(range2(x), 100))
x[sample(x, 1e5)] <- NA

system.time(rep(range(x, na.rm = TRUE), 100))
system.time(rep(range2(x, na.rm = TRUE), 100))
```

| | |
|----------|----------------------|
| read_bib | <i>Read Bib file</i> |
|----------|----------------------|

Description

Read a bib file into a data.frame

Usage

```
read_bib(file, skip = 0L, max_lines = NULL, encoding = "UTF-8")
```

Arguments

| | |
|-----------|---|
| file | File or connection |
| skip | The lines to skip |
| max_lines | The maximum number of lines to read |
| encoding | Assumed encoding of file (passed to readLines()) |

Details

Inspired and partially credited to `bib2df::bib2df()` although this has no dependencies outside of base functions and much quicker. This speed seems to come from removing stringr functions and simplifying a few *apply functions. This will also include as many categories as possible from the entry.

Value

A data.frame with each row as a bib entry and each column as a field

See Also

?bib2df::bib2df

Examples

```
file <- "https://raw.githubusercontent.com/jmbarbone/bib-references/master/references.bib"
bibdf <- read_bib(file, max_lines = 51L)

if (package_available("tibble")) {
  tibble::as_tibble(bibdf)
} else {
  head(bibdf)
}

if (package_available("bib2df") & package_available("bench")) {
  file <- system.file("extdata", "bib2df_testfile_3.bib", package = "bib2df")

  # Doesn't include the 'tidying' up
```

```

foo <- function(file) {
  bib <- ("bib2df" %colons% "bib2df_read")(file)
  ("bib2df" %colons% "bib2df_gather")(bib)
}

bench::mark(
  read_bib = read_bib(file),
  bib2df = bib2df::bib2df(file),
  foo = foo(file),
  check = FALSE
)[1:9]
}

```

recode_by

Recode by

Description

A simple implementation of recoding

Usage

```
recode_by(x, by, vals = NULL, mode = "any")
```

```
recode_only(x, by, vals = NULL)
```

Arguments

| | |
|-------------------|---|
| <code>x</code> | A vector to recode |
| <code>by</code> | A names vector (<code>new = old</code>); any non-matching values are set to the appropriate NA |
| <code>vals</code> | An optional vector of values to use in lieu of a names in the vector; this takes priority over names (<code>by</code>). This can be the same length as <code>by</code> or a single value. |
| <code>mode</code> | passed to <code>as.vector()</code> |

Details

This can be comparable to `dplyr::recode()` expect that the values are arranged as `new = old` rather than `old = new` and allows for a separate vector to be passed for `new`.

`recode_only()` will only recode the values matches in `by/val`. The mode is automatically set according to `mode(x)`. This functions more like `base::replace()` but with extra features

Value

A vector of values from `x`

See Also[dplyr::recode\(\)](#)**Examples**

```

recode_by(1:3, c(a = 1, b = 2))
recode_by(letters[1:3], c(`1` = "a", `2` = "b")) # will not guess mode
recode_by(letters[1:3], c(`1` = "a", `2` = "b"), mode = "integer") # make as integer
recode_by(letters[1:3], c("a", "b"), vals = 1:2) # or pass to vals

recode_only(letters[1:3], c("zzz" = "a"))
recode_only(letters[1:3], c(`1` = "a")) # returns as "1"
recode_only(1:3, c("a" = 1)) # coerced to NA

# Pass list for multiples
recode_only(letters[1:10], list(abc = c("a", "b", "c"), ef = c("e", "f")))

```

| | |
|---------|-----------------------------|
| reindex | <i>Reindex a data.frame</i> |
|---------|-----------------------------|

Description

Reindexes a data.frame with a reference

Usage

```

reindex(
  x,
  index = NULL,
  new_index,
  expand = c("intersect", "both"),
  sort = FALSE
)

```

Arguments

| | |
|-----------|---|
| x | A data.frame |
| index | The column name or number of an index to use; if NULL will assume the first column; a value of row.names will use row.names(x) |
| new_index | A column vector of the new index value |
| expand | Character switch to expand or keep only the values that intersect (none), all values in x or index, or retain all values found. |
| sort | Logical, if TRUE will sort the rows in output |

Value

A data.frame with rows of index

Examples

```
iris1 <- head(iris, 5)
iris1$index <- 1:5
reindex(iris1, "index", seq(2, 8, 2))
reindex(iris1, "index", seq(2, 8, 2), expand = "both")

# Using letters will show changes in rownames
iris1$index <- letters[1:5]
reindex(iris1, "index", letters[seq(2, 8, 2)])
reindex(iris1, "index", seq(2, 8, 2))
reindex(iris1, "index", seq(2, 8, 2), expand = "both")
```

remove_na

Remove NA

Description

Remove NAs from a vector

Usage

```
remove_na(x)

## Default S3 method:
remove_na(x)

## S3 method for class 'list'
remove_na(x)

## S3 method for class 'factor'
remove_na(x)

## S3 method for class 'fact'
remove_na(x)
```

Arguments

x A vector of values

Details

remove_na.factor will remove NA values as identified by the levels() or by the integer value of the level. factors are recreated with all NA values and, if present, the NA level removed.

Value

x without values where is.na(x) is TRUE For factors, a new factor (ordered if is.ordered(x))

Examples

```
remove_na(c(4, 1, 2, NA, 4, NA, 3, 2))

# removes based on levels
remove_na(factor(c("b", NA, "a", "c")))

# removes based on values
x <- as_ordered(c("b", "d", "a", "c"))
x[2:3] <- NA
str(remove_na(x))
```

| | |
|-------------|--------------------|
| remove_null | <i>Remove NULL</i> |
|-------------|--------------------|

Description

Remove NULL results from a list

Usage

```
remove_null(x)
```

Arguments

x A list

Value

The list x without NULL

Examples

```
x <- list(a = letters[1:5], b = NULL, c = complex(3))
x
remove_null(x)
```

| | |
|----------|---|
| round_by | <i>Rounding by a specific interval.</i> |
|----------|---|

Description

Rounds a number or vector of numbers by another

Usage

```
round_by(x, by = 1, method = c("round", "ceiling", "floor"), include0 = TRUE)
```

Arguments

| | |
|----------|---|
| x | A number or vector to round. |
| by | The number by which to round |
| method | An option to explicitly specify automatic rounding, ceiling, or floor |
| include0 | If FALSE replaces 0 with by |

Value

A vector of doubles of the same length of x

Examples

```
x <- seq(1, 13, by = 4/3)

cbind(
  x,
  by_1 = round_by(x, 1),
  by_2 = round_by(x, 2),
  by_3 = round_by(x, 3)
)
```

row_bind

Row bind

Description

Bind a list of data.frames

Usage

```
row_bind(...)
```

Arguments

... A list of data.frames to be attached to each other by row

Value

A data.frame combining all the rows from data.frames in ... and all the columns, as they appear. An empty data.frame with 0 columns and 0 rows is returned if ... has no length

See Also

[dplyr::bind_rows\(\)](#) [base::rbind\(\)](#)

| | |
|---------|----------------|
| rscript | <i>Rscript</i> |
|---------|----------------|

Description

Implements Rscript with system2

Usage

```
rscript(x, ops = NULL, args = NULL, ...)
```

Arguments

| | |
|------|---|
| x | An R file to run |
| ops | A character vector of options ("--" is added to each) |
| args | A character vector of other arguments to pass |
| ... | Additional arguments passed to system2 |

Value

A character vector of the result from calling Rscript via system2()

See Also

[source_to_env](#)

| | |
|-------------|--------------------|
| save_source | <i>Save source</i> |
|-------------|--------------------|

Description

Source a file and save as file

Usage

```
save_source(env = parent.frame(), file = mark_temp("Rds"), name = NULL)
```

Arguments

| | |
|------|--|
| env | The parent environment |
| file | The file to save the environment to |
| name | An optional name for the environment (mostly cosmetic) |

Value

A source_env/environment object, created from env

| | |
|------------|------------------|
| set_names0 | <i>Set names</i> |
|------------|------------------|

Description

Sets or removes names

Usage

```
set_names0(x, nm = x)
```

```
names_switch(x)
```

Arguments

| | |
|----|--------------------|
| x | A vector of values |
| nm | A vector of names |

Value

- set_names0(): x with nm values assigned to names (if x is NULL, NULL is returned)
- remove_names(): x without names
- names_switch(): character vector of equal length x where names and values are switched

| | |
|------------------|---------------------|
| simpleTimeReport | <i>Time reports</i> |
|------------------|---------------------|

Description

[Experimental] This function can be used to evaluate an expression line-by-line to capture outputs, errors, messages, and evaluation time.

Usage

```
simpleTimeReport(title = NULL, expr, envir = parent.frame())
```

Arguments

| | |
|-------|---|
| title | The title to be printed |
| expr | The expression to run |
| envir | The environment from which to evaluate the expr |

Details

Evaluate code and report on the time difference

Value

A reported_results/list object containing results, outputs, messages, warnings, and errors

Examples

```
simpleTimeReport("example", {
  print("1")
  Sys.sleep(1)
  warning("this is a warning")
  for (i in 1:5) {
    Sys.sleep(0.5)
  }
  sample(1e6, 1e6, TRUE)
})
```

 sort_by

Sort by

Description

Sort an object by another object

Usage

```
sort_by(x, by, ...)
```

Arguments

| | |
|-----|--|
| x | A vector |
| by | Another vector |
| ... | Additional arguments passed to base::order() |

Value

The values of x, resorted

Examples

```
l3 <- letters[1:3]
sort_by(l3, c(3, 2, 1))
# make a factor object with the reversed order
f <- factor(l3, levels = rev(l3))
sort_by(f, l3)
sort_by(1:3, rev(l3))
```

| | |
|------------|----------------------|
| sort_names | <i>Sort by names</i> |
|------------|----------------------|

Description

Sort a vector by it's name

Usage

```
sort_names(x, numeric = FALSE)
```

Arguments

| | |
|---------|---------------------------------------|
| x | A named vector of values |
| numeric | If TRUE will try to coerce to numeric |

Value

x sorted by its names()

| | |
|--------------|-----------------------------------|
| source_files | <i>Source file from directory</i> |
|--------------|-----------------------------------|

Description

Walk through files in a directory and output them. Files are sources in order of names

Usage

```
source_r_dir(dir, echo = FALSE, quiet = FALSE, ...)
```

```
source_r_file(path, echo = FALSE, quiet = FALSE, ...)
```

Arguments

| | |
|-------|--|
| dir | The location of your R scripts |
| echo | logical; if TRUE, each expression is printed after parsing, before evaluation. |
| quiet | Logical. Whether to print out a message for each file. |
| ... | Additional arguments passed to base::source() |
| path | The location of the R file. |

Value

None, called for side effects

| | |
|---------------|------------------------------|
| source_to_env | <i>Source to environment</i> |
|---------------|------------------------------|

Description

Source an R script to an environment

Usage

```
source_to_env(x, ops = NULL)
```

Arguments

| | |
|-----|---|
| x | An R script |
| ops | Options to be passed to rscript |

Value

Invisibly, and environment variable of the objects/results created from x

| | |
|----------|----------------------------|
| sourcing | <i>Sourcing extensions</i> |
|----------|----------------------------|

Description

Functions for extending sourcing features

Usage

```
ksource(file, ..., quiet = TRUE, cd = FALSE, env = parent.frame())
```

```
try_source(file, cd = FALSE, ...)
```

```
try_ksource(file, ...)
```

Arguments

| | |
|-------|--|
| file | An R or Rmd file. |
| ... | Additional arguments passed to base::source() |
| quiet | Logical; Determines whether to apply silence to knitr::purl() |
| cd | Logical; if TRUE, the R working directory is temporarily changed to the directory containing file for evaluating |
| env | An environment determining where the parsed expressions are evaluated |

Details

try_source() will output an error message rather than completely preventing the execution. This can be useful for when a script calls on multiple, independent files to be sourced and a single failure shouldn't prevent the entire run to fail as well.

Value

- ksource(): Invisibly, the result of calling source() on the .R file conversion of file
- try_source(), try_ksource(): attempts of source() and ksource() but converts errors to warnings

| | |
|------------------|---------------------------------|
| str_extract_date | <i>Extract date from string</i> |
|------------------|---------------------------------|

Description

Extract date from string

Usage

```
str_extract_date(x, format = "%Y-%m-%d")
str_extract_datetime(x, format = "%Y-%m-%d %H%M%S")
```

Arguments

| | |
|--------|-----------------------|
| x | A character vector |
| format | A date format to find |

Value

A Date (if found) or NA

Examples

```
str_extract_date("This is a file name 2020-02-21.csv")
str_extract_date(c("This is a file name 2020-02-21.csv",
                  "Date of 2012-06-15 here"))
str_extract_date(c("This is a file name 2020-02-21.csv", "No date"))
str_extract_date("Last saved 17 December 2019", format = "%d %B %Y")

str_extract_datetime(c("2020-02-21 235033", "2012-12-12 121212"))
str_extract_datetime("This is a file name 2020-02-21 235033.csv")
```

| | |
|-----------|---------------------|
| str_slice | <i>String Slice</i> |
|-----------|---------------------|

Description

Slice/split a string into multiple lines by the desired length of the line.

Usage

```
str_slice(x, n = 80L)
str_slice_by_word(x, n = 80L)
```

Arguments

| | |
|---|---------------------------------------|
| x | A character vector |
| n | Integer, the length of the line split |

Value

A character vector

Examples

```
if (requireNamespace("stringi")) {
  x <- stringi::stri_rand_lipsum(1)
  str_slice(x)
  str_slice_by_word(x, n = 50L)
}
```

| | |
|------------|---|
| switch-ext | <i>Switch with a list of parameters</i> |
|------------|---|

Description

switch_params() is a vectorized version of switch switch_case() uses a formula syntax to return the value to the right of the tilde (~) when x is TRUE switch_in_case() is a special case of switch_case() for match()-ing x in the values on the left to return the value on the right.

Usage

```
switch_params(x, ...)
switch_in_case(x, ..., .default = NULL, .envir = parent.frame())
switch_case(..., .default = NULL, .envir = parent.frame())
```

Arguments

| | |
|-----------------------|---|
| <code>x</code> | A vector of values |
| <code>...</code> | Case evaluations (named for <code>switch_params</code>) |
| <code>.default</code> | The default value if no matches are found in <code>...</code> (default: NULL produces an NA value derived from <code>...</code>) |
| <code>.envir</code> | The environment in which to evaluate the LHS of <code>...</code> (default: <code>parent.frame()</code>) |

Details

Switch with a list of params

Value

A named vector of values of same length `x`; or for `switch_case`, an unnamed vector of values matching the rhs of `...`

Inspired from:

- <https://stackoverflow.com/a/32835930/12126576>
- <https://github.com/tidyverse/dplyr/issues/5811>

Examples

```
# by single
switch_params(c("j", "m", "b"), j = 10, b = 2, m = 13)
```

```
# match with TRUE
switch_case(
  1:10 == 9 ~ NA_integer_,
  1:10 %% 3 == 0 ~ 1:10,
  1:10 %% 4 == 0 ~ 11:20,
  1:10 %% 5 == 0 ~ 21:30,
  1:10 %% 2 == 0 ~ 31:40,
  .default = -1L
)
```

```
# match within a vector
switch_in_case(
  c(1, 2, 12, 4, 20, 21),
  1:10 ~ 1,
  11:20 ~ 2
)
```

```
switch_in_case(
  c("a", "b", "d", "e", "g", "j"),
  letters[1:3] ~ "a",
  letters[5:6] ~ "e"
)
```

```
use_these <- c(1, 3, 2, 5)
```

```

switch_in_case(
  1:10,
  use_these ~ TRUE,
  .default = FALSE
)

ne <- new.env()
ne$use_these2 <- use_these
# error
try(switch_in_case(
  1:10,
  use_these2 ~ TRUE
))
switch_in_case(
  1:10,
  use_these2 ~ TRUE,
  .envir = ne
)

switch_in_case(
  seq.int(1, 60, 6),
  1:10 ~ "a",
  11:20 ~ "b",
  c(22, 24, 26) ~ "c",
  30:Inf ~ "d"
)

# Use functions
switch_in_case(
  1:6,
  c(1, 3, 5) ~ exp,
  c(2, 4) ~ log
)

```

tableNA

Table NA values

Description

Tables out whether data are NAs are not

Usage

```
tableNA(..., .list = FALSE)
```

Arguments

... one or more objects which can be interpreted as factors (including numbers or character strings), or a [list](#) (such as a data frame) whose components can be so interpreted. (For `as.table`, arguments passed to specific methods; for `as.data.frame`, unused.)

`.list` Logical, if TRUE and `...` is a list, will c

Details

All data are checked with `is.na()` and the resulting TRUE or FALSE is are tabulated.

Value

`table()` returns a *contingency table*, an object of class "table", an array of integer values. Note that unlike S the result is always an [array](#), a 1D array if one factor is given.

`as.table` and `is.table` coerce to and test for contingency table, respectively.

The `as.data.frame` method for objects inheriting from class "table" can be used to convert the array-based representation of a contingency table to a data frame containing the classifying factors and the corresponding entries (the latter as component named by `responseName`). This is the inverse of [xtabs](#).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

[tabulate](#) is the underlying function and allows finer control.

Use [ftable](#) for printing (and more) of multidimensional tables. [margin.table](#), [prop.table](#), [addmargins](#).

[addNA](#) for constructing factors with [NA](#) as a level.

[xtabs](#) for cross tabulation of data frames with a formula interface.

Examples

```
x <- list(
  a = c(1, 2, NA, 3),
  b = c("A", NA, "B", "C"),
  c = as.Date(c("2020-01-02", NA, NA, "2020-03-02"))
)
tableNA(x) # entire list
tableNA(x, .list = TRUE) # counts for each
tableNA(x[1], x[2])
tableNA(x[1], x[2], x[3]) # equivalent ot tableNA(x, .list = TRUE)
```

| | |
|------|-------------|
| that | <i>That</i> |
|------|-------------|

Description

Grammatical correctness

Usage

```
that(x, arr.ind = FALSE, useNames = TRUE)
```

Arguments

| | |
|----------|---|
| x | a logical vector or array. NAs are allowed and omitted (treated as if FALSE). |
| arr.ind | logical; should array indices be returned when x is an array? Anything other than a single true value is treated as false. |
| useNames | logical indicating if the value of <code>arrayInd()</code> should have (non-null) <code>dimnames</code> at all. |

Details

See `fortunes::fortune(175)`.

Value

see [base::which\(\)](#)

See Also

[base::which\(\)](#)

| | |
|-------|------------------|
| todos | <i>Get TODOs</i> |
|-------|------------------|

Description

Search for `#` `` TODO tags

Usage

```

todos(
  pattern = NULL,
  path = ".",
  force = getOption("mark.todos.force"),
  ext = getOption("mark.todos.ext"),
  ignore = NULL,
  ...
)

fixmes(
  pattern = NULL,
  path = ".",
  force = getOption("mark.todos.force"),
  ext = getOption("mark.todos.ext"),
  ignore = NULL,
  ...
)

```

Arguments

| | |
|---------|---|
| pattern | A character string containing a regular expression to filter for comments after tags; default NULL does not filter |
| path | Where to search for the todos. If this is a directory, paths matching the ext will be included. If a file, ext is ignored. |
| force | If TRUE will force searching for files in directories that do not contain an .Rproj file. This can be controlled with the option mark.todos.force |
| ext | A vector of file extensions to search for todos. Ignored when path is not a directory or when NULL. |
| ignore | A regular expression for files to ignore. Ignored if path is not a directory or when NULL. |
| ... | Additional parameters passed to grep (Except for pattern, x, and value) |

Details

Searches for TODO comments in files. Extensions with md, Rmd, and qmd specifically search for a <-- TODO * --> string, whereas everything else is found with # TODO.

Value

NULL if none are found, otherwise a data.frame with the line number, file name, and TODO comment.

Examples

```

## Not run:
file <- tempfile()
writeLines(c(

```

```

    "# TODO make x longer",
    "x <- 1:10",
    "length(x)",
    "# TODO add another example",
    "# FIXME This is a fixme"
  ), file)
todos(path = file)
todos("example", path = file)
fixmes(path = file)
file.remove(file)

## End(Not run)

```

to_boolean

To Boolean

Description

Convert a vector to boolean/logical

Usage

```

to_boolean(x, ...)

## S3 method for class 'logical'
to_boolean(x, ...)

## S3 method for class 'numeric'
to_boolean(x, true = 1L, false = 0L, ...)

## S3 method for class 'character'
to_boolean(x, true = NULL, false = NULL, ...)

## S3 method for class 'factor'
to_boolean(x, true = NULL, false = NULL, ...)

```

Arguments

| | |
|-------|--|
| x | A vector of values |
| ... | Additional arguments passed to methods |
| true | A vector of values to convert to TRUE |
| false | A vector of values to convert to FALSE |

Value

A logical vector of equal length as x

| | |
|--------------|---------------------|
| to_row_names | <i>To row names</i> |
|--------------|---------------------|

Description

Converts a column to row names

Usage

```
to_row_names(data, row_names = 1L)
```

Arguments

| | |
|-----------|-------------------------------------|
| data | A data.frame |
| row_names | The numeric position of the column. |

Value

A data.frame

Examples

```
x <- data.frame(  
  a = 1:4,  
  b = letters[1:4]  
)  
  
to_row_names(x)  
to_row_names(x, "b")
```

| | |
|------|--|
| tryn | <i>Try an expression a set number of times</i> |
|------|--|

Description

Try an expression a set number of times

Usage

```
tryn(expr, n = 10, silent = TRUE)
```

Arguments

| | |
|--------|--------------------------------|
| expr | expression to evaluate |
| n | number of attempts until error |
| silent | whether to suppress warnings |

Value

result of expr

Examples

```
foo <- function() stop("I added an error")
try(tryn(n = 10, foo()))
```

| | |
|------|-----------------------------|
| t_df | <i>Data frame transpose</i> |
|------|-----------------------------|

Description

This transposes a data.frame with `t()` but transforms back into a data.frame with column and row names cleaned up. Because the data types may be mixed and reduced to characters, this may only be useful for a visual viewing of the data.frame.

Usage

```
t_df(x, id = NULL)
```

Arguments

| | |
|----|----------------|
| x | A data.frame |
| id | No longer used |

Details

Transposes a data.frame as a data.frame

Value

A transposed data.frame with columns ("colname", "row_1", ..., for each row in x.

Examples

```
x <- data.frame(col_a = Sys.Date() + 1:5, col_b = letters[1:5], col_c = 1:5)
t_df(x)
```

| | |
|-------------|--------------------|
| unique_rows | <i>Unique rows</i> |
|-------------|--------------------|

Description

Drops duplicated rows

Usage

```
unique_rows(data, cols = NULL, from_last = FALSE, invert = FALSE)
```

Arguments

| | |
|-----------|---|
| data | A data.frame |
| cols | Columns to compare against; when NULL selects all columns |
| from_last | When TRUE returns the last row containing duplicates, rather than the first |
| invert | If TRUE returns the duplicated rows |

Value

data with duplicates removed

Examples

```
df <- quick_df1(
  i = 1:4,
  a = rep(1:2, 2L),
  b = rep("a", 4L),
)

unique_rows(df, 2:3)
unique_rows(df, c("a", "b"), from_last = TRUE, invert = TRUE)
```

| | |
|---------|--------------------------|
| unlist0 | <i>Unlist and squash</i> |
|---------|--------------------------|

Description

Unlist without unique names; combine names for unique values

Usage

```
unlist0(x)

squash_vec(x, sep = ".")
```

Arguments

x A vector of values
 sep A separation for combining names

Details

- `unlist0()` is much like `unlist()` expect that name are not made to be unique.
- `squash_vec()` works differently

Value

- `unlist0()`: a vector with the possibility of non-unique names
- `squash_vec()`: A vector of unique values and names

Examples

```
x <- list(a = 1:3, b = 2, c = 2:4)
y <- c(a = 1, b = 1, c = 1, d = 2, e = 3, f = 3)

# unlist0() doesn't force unique names
unlist(x) # names: a1 a2 a3 b c1 c2 c3
unlist0(x) # names: a a a b c c c
unlist0(y) # no change

# squash_vec() is like the inverse of unlist0() because it works on values
squash_vec(x)
squash_vec(y)
```

use_author *Add author to DESCRIPTION*

Description

Adds author to description

Usage

```
use_author(author_info = find_author())
```

Arguments

author_info Author information as a named list

Details

Only valid for a single author.

Value

None, called for side effects

 utils-paste

Paste combine

Description

Paste and combine

Usage

```
paste_c(x, y, collate = TRUE, sep = "")
paste_combine(..., collate = TRUE, sep = "")
collapse0(..., sep = "")
```

Arguments

| | |
|------------------------|--|
| <code>x, y, ...</code> | Vectors to paste and/or combine |
| <code>collate</code> | Logical; TRUE prints out combinations in order of the first vector elements then the next; otherwise reversed (see examples) |
| <code>sep</code> | A character string to separate terms |

Value

A character vector

Examples

```
x <- letters[1:5]
y <- 1:3
z <- month.abb[c(1, 12)]
paste_combine(x, y)
paste_combine(x, y, z)
paste_combine(x, y, z, sep = ".")
paste_combine(x, y, sep = "_")
paste_combine(x, y, collate = FALSE)
collapse0(list(1:3, letters[1:3]), 5:7, letters[5:7])
collapse0(1:3, letters[5:7], sep = "_")
```

| | |
|-----|--------------|
| vap | <i>Vaps!</i> |
|-----|--------------|

Description

Wrappers for vapply

Usage

```
vap_int(.x, .f, ..., .nm = FALSE)
```

```
vap_dbl(.x, .f, ..., .nm = FALSE)
```

```
vap_chr(.x, .f, ..., .nm = FALSE)
```

```
vap_lgl(.x, .f, ..., .nm = FALSE)
```

```
vap_cplx(.x, .f, ..., .nm = FALSE)
```

```
vap_date(.x, .f, ..., .nm = FALSE)
```

Arguments

| | |
|------------------|--|
| <code>.x</code> | A vector of values |
| <code>.f</code> | A function to apply to each element in vector <code>.x</code> |
| <code>...</code> | Additional arguments passed to <code>.f</code> |
| <code>.nm</code> | Logical, if TRUE returns names of <code>.x</code> (Note: If <code>.x</code> does not have any names, they will be set to the values) |

Details

These are simply wrappers for `base::vapply()` to shorten lines.

Each function is designed to use specific vector types:

vap_int integer

vap_dbl double

vap_chr character

vap_lgl logical

vap_cplx complex

vap_date Date

Value

A vector of type matching the intended value in the function name.

See Also

[base::vapply\(\)](#)

| | |
|-----------|-----------------------------|
| vector2df | <i>Vector to data.frame</i> |
|-----------|-----------------------------|

Description

Transforms a vector (named) to a data.frame

Usage

```
vector2df(x, name = "name", value = "value", show_NA)
```

Arguments

| | |
|-------------|--|
| x | A vector of values. |
| name, value | Character strings for the name and value columns |
| show_NA | Ignored; will trigger a warning if set |

Value

A data.frame with name (optional) and value columns

| | |
|-------------|------------------------|
| within_call | <i>Function within</i> |
|-------------|------------------------|

Description

Returns the function call you are within

Usage

```
within_call()
within_fun()
outer_call(n = 0)
outer_fun(n = 0)
```

Arguments

| | |
|---|--------------------------------------|
| n | The number of calls to move out from |
|---|--------------------------------------|

Value

The string of the call/function

| | |
|----------|---------------------------|
| with_par | <i>Temporary plotting</i> |
|----------|---------------------------|

Description

Reset par() after running

Usage

```
with_par(..., ops = NULL)
```

Arguments

... Code to be evaluated
ops A named list to be passed to `graphics::par()`

Value

Invisibly, the result of ...

Examples

```
with_par(  
  plot(lm(Sepal.Length ~ Sepal.Width, data = iris)),  
  plot(lm(Petal.Length ~ Petal.Width, data = iris)),  
  ops = list(mfrow = c(2, 4))  
)
```

| | |
|----------------|---------------------------------------|
| write_file_md5 | <i>Write file with md5 hash check</i> |
|----------------|---------------------------------------|

Description

Write file with md5 hash check

Usage

```
write_file_md5(  
  x,  
  path = NULL,  
  method = mark_write_methods(),  
  overwrite = NA,  
  quiet = FALSE,  
  encoding = "UTF-8",  
  compression = getOption("mark.compress.method", mark_compress_methods()),  
  ...
```

```
)
mark_write_methods()
mark_compress_methods()
```

Arguments

| | |
|-------------|--|
| x | An object to write to file |
| path | The file or connection to write to (dependent on part by method) |
| method | The method of saving the file. When default, the method is determined by file extension of path, if present, otherwise by the type of object of x. |
| overwrite | When NA, only saves if the md5 hashes do not match. Otherwise, see fs::file_copy() . |
| quiet | When TRUE, suppresses messages from md5 checks. |
| encoding | The encoding to use when writing the file. |
| compression | The compression method to use when writing the file. |
| ... | Additional arguments passed to the write function. |

Value

- [write_file_md5\(\)](#): x, invisibly. When path is not the stdout(), x is returned with the attribute "path" set to the result of [file_copy_md5\(\)](#).
- [mark_write_methods\(\)](#): A list of applicable methods and their aliases
- [mark_compress_methods\(\)](#): A character vector of applicable compression methods

options()

- `mark.compress.method`: compression method to use when writing files
- `mark.list.hook`: when a data.frame contains a list column, this function is applied to each element of the list. The default "auto" uses `toJSON()` if the package `jsonlite` is available, otherwise

Examples

```
# just writes to stdout()
df <- data.frame(a = 1, b = 2)
write_file_md5(df)

temp <- tempfile()
write_file_md5(df, temp) # new
write_file_md5(df, temp) # same
df$c <- 3
write_file_md5(df, temp) # changes
fs::file_delete(temp)
```

Index

- * **datasets**
 - fizzbuzz, 31
 - not_available, 57
- * **factors**
 - as_ordered, 6
 - char2fact, 11
 - drop_levels, 21
 - fact, 23
 - fact2char, 25
 - fact_na, 26
- .fizzbuzz_vector (fizzbuzz), 31
- %xor%(logic_ext), 43

- add_file_timestamp, 4
- addmargins, 80
- addNA, 80
- AND (logic_ext), 43
- are_identical, 5
- array, 80
- array_extract, 6
- as_ordered, 6, 11, 21, 25, 26
- as_ordered(), 25
- assign_labels (labels), 38

- base::comment(), 56
- base::droplevels, 21
- base::grepl(), 51
- base::identical, 5
- base::isFALSE(), 43
- base::isTRUE(), 43
- base::match.arg(), 46, 47
- base::min, 64
- base::range(), 54, 64
- base::rbind(), 70
- base::replace(), 66
- base::source(), 74, 75
- base::vapply(), 89, 90
- base::which(), 81
- base::xor(), 43
- base_alpha, 7

- base_n, 8
- betwee_more (betwixt), 9
- between (betwixt), 9
- between_more (betwixt), 9
- between_more(), 9
- betwixt, 9
- betwixt(), 9
- blank_values, 10
- bump_date_version (get_version), 34
- bump_version (get_version), 34

- char2fact, 7, 11, 21, 25, 26
- char2fact(), 25
- checkOptions, 11
- chr_split, 12
- clipboard, 13
- collapse0 (utils-paste), 88
- complete_cases, 14
- counts, 15

- date_from_partial, 16
- depth, 17
- detail, 18
- diff_time, 19
- diff_time_days (diff_time), 19
- diff_time_dyears (diff_time), 19
- diff_time_hours (diff_time), 19
- diff_time_mins (diff_time), 19
- diff_time_months (diff_time), 19
- diff_time_myyears (diff_time), 19
- diff_time_secs (diff_time), 19
- diff_time_weeks (diff_time), 19
- diff_time_wyears (diff_time), 19
- diff_time_years (diff_time), 19
- dplyr::bind_rows(), 70
- dplyr::na_if(), 53
- dplyr::recode(), 66, 67
- drop_levels, 7, 11, 21, 25, 26

- either (logic_ext), 43

- environments (list_environments), 42
- ept, 21
- eval_named_chunk, 22
- expand_by, 23
- fact, 7, 11, 21, 23, 25, 26
- fact(), 7, 26
- fact2char, 7, 11, 21, 25, 25, 26
- fact2char(), 11
- fact_na, 7, 11, 21, 25, 26
- fact_reverse, 26
- fct_expand_seq, 27
- file_copy_md5, 28
- file_copy_md5(), 92
- file_info, 28
- file_name, 29
- file_open (file_utils), 29
- file_path (norm_path), 55
- file_utils, 29
- fixmes (todos), 81
- fizzbuzz, 31
- fizzbuzz_lazy (fizzbuzz), 31
- fs::dir_ls(), 30
- fs::file_copy(), 28, 92
- ftable, 80
- get_dir_max_number, 32
- get_dir_recent_date, 32
- get_error (handlers), 35
- get_labels (labels), 38
- get_message (handlers), 35
- get_recent_dir, 33
- get_recent_file, 33
- get_version, 34
- get_warning (handlers), 35
- glob, 34
- glob2rx, 45
- graphics::par(), 91
- handlers, 35
- has_error (handlers), 35
- has_message (handlers), 35
- has_warning (handlers), 35
- import, 36
- insert, 37
- is_blank (blank_values), 10
- is_blank_cols (blank_values), 10
- is_boolean (logic_ext), 43
- is_dir, 38
- is_false (logic_ext), 43
- is_file (is_dir), 38
- is_na_cols (na_cols), 53
- is_true (logic_ext), 43
- knitr::purl(), 75
- ksource (sourcing), 75
- labels, 38
- largest_file (file_info), 28
- limit, 40
- lines_of_r_code, 40
- list, 79
- list2df, 41
- list_dirs (file_utils), 29
- list_dirs(), 33
- list_environments, 42
- list_files (file_utils), 29
- list_files(), 33
- logic_ext, 43
- logical, 81
- ls_all (list_environments), 42
- ls_dataframe (ls_ext), 45
- ls_ext, 45
- ls_function (ls_ext), 45
- ls_object (ls_ext), 45
- make_sf, 45
- margin.table, 80
- mark, 46
- mark-package (mark), 46
- mark_compress_methods (write_file_md5), 91
- mark_compress_methods(), 92
- mark_write_methods (write_file_md5), 91
- mark_write_methods(), 92
- match_arg, 46
- match_arg(), 48
- match_ext, 52
- match_param, 47
- match_param(), 47
- md5, 49
- median2, 49
- merge_list, 50
- multi_grep (multi_grepl), 51
- multi_grepl, 51
- NA, 50, 80, 81

- na_assignments, 52
- NA_at (na_assignments), 52
- na_cols, 53
- NA_Date_ (not_available), 57
- NA_if (na_assignments), 52
- NA_in (na_assignments), 52
- NA_out (na_assignments), 52
- NA_POSIXct_ (not_available), 57
- NA_POSIXlt_ (not_available), 57
- names_switch (set_names0), 72
- newest_dir (file_info), 28
- newest_file (file_info), 28
- none (logic_ext), 43
- norm_path, 55
- normalize, 54
- not_available, 57
- note, 56
- note<- (note), 56

- objects_all (list_environments), 42
- oldest_dir (file_info), 28
- oldest_file (file_info), 28
- omit_na, 58
- open_file (file_utils), 29
- OR (logic_ext), 43
- outer_call (within_call), 90
- outer_fun (within_call), 90

- package_available, 59
- paste_c (utils-paste), 88
- paste_combine (utils-paste), 88
- percentile_rank, 59
- pmatch(), 48
- print.mark_bib_df, 60
- print.pseudo_id, 61
- print_c, 61
- process_bib_dataframe, 62
- prop.table, 80
- props (counts), 15
- pseudo_id, 61, 63
- pseudo_id(), 61

- q50 (median2), 49
- quiet_stop, 64

- range2, 64
- read_bib, 65
- read_clipboard (clipboard), 13
- readLines(), 65

- recode_by, 66
- recode_only (recode_by), 66
- regular expression, 45
- reindex, 67
- remove_blank_cols (blank_values), 10
- remove_labels (labels), 38
- remove_na, 68
- remove_na_cols (na_cols), 53
- remove_null, 69
- round_by, 69
- row_bind, 70
- rscript, 71, 75

- save_source, 71
- select_blank_cols (blank_values), 10
- select_na_cols (na_cols), 53
- serialized, 49
- set_names0, 72
- set_not_available (not_available), 57
- set_note (note), 56
- shell_exec (file_utils), 29
- simpleTimeReport, 72
- smallest_file (file_info), 28
- sort_by, 73
- sort_names, 74
- source_files, 74
- source_r_dir (source_files), 74
- source_r_file (source_files), 74
- source_to_env, 71, 75
- sourcing, 75
- squash_vec (unlist0), 86
- squash_vec(), 87
- stats::quantile(), 50
- str_extract_date, 76
- str_extract_datetime
 (str_extract_date), 76
- str_slice, 77
- str_slice_by_word (str_slice), 77
- switch-ext, 77
- switch_case (switch-ext), 77
- switch_in_case (switch-ext), 77
- switch_params (switch-ext), 77

- t_df, 85
- tableNA, 79
- tabulate, 80
- that, 81
- to_boolean, 83
- to_row_names, 84

todos, [81](#)
tools::md5sum(), [49](#)
try_ksource(sourcing), [75](#)
try_source(sourcing), [75](#)
tryn, [84](#)

unique_rows, [86](#)
unlist(), [87](#)
unlist0, [86](#)
unlist0(), [87](#)
update_version(get_version), [34](#)
use_author, [87](#)
user_file(norm_path), [55](#)
utils-paste, [88](#)
utils::file_test(), [38](#)
utils::glob2rx(), [30](#)
utils::readClipboard(), [13](#)
utils::write.table(), [13](#)
utils::writeClipboard(), [13](#)

vap, [89](#)
vap_chr(vap), [89](#)
vap_cplx(vap), [89](#)
vap_date(vap), [89](#)
vap_dbl(vap), [89](#)
vap_int(vap), [89](#)
vap_lgl(vap), [89](#)
vector2df, [90](#)
view_labels(labels), [38](#)

with_par, [91](#)
within_call, [90](#)
within_fun(within_call), [90](#)
write_clipboard(clipboard), [13](#)
write_file_md5, [91](#)
write_file_md5(), [92](#)

xtabs, [80](#)