Package: Vmisc (via r-universe)

November 12, 2024

Title Various functions for personal use

Version 0.1.10

Description Various functions for personal use.

License MIT + file LICENSE

URL https://github.com/venpopov/Vmisc

BugReports https://github.com/venpopov/Vmisc/issues

Imports withr (>= 2.2.0), stringr, tools, utils, xfun, rlang, pkgload, remotes, methods, glue

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Config/pak/sysreqs git make libicu-dev

Repository https://popov-lab.r-universe.dev

RemoteUrl https://github.com/venpopov/Vmisc

RemoteRef master

RemoteSha 06f70902e3db2ea769e14ef9fc4502616e00971f

Contents

arg2string 2
attr 3
available_packages 4
collapse 4
component_size 5
extract_pkg_fun_calls 5
is_dir_empty 6
nlist 7
op-null-default 7

packageOptions 8
parse_pkg_version 9
pkg_switch_default 9
pkg_vavailable 10
pkg_vload 11
require_pkg 12
return_on_exit 13
stop2 14
stopifnot2 15
strip_attributes 16
str_extract_nested_balanced 16
%+% 17

Index **19**

`arg2string` *Convert Function Arguments to Strings*

Description

This function takes any number of R language objects and converts their names into strings. This is particularly useful for programming where variable names or symbols need to be used as strings without evaluating them. It leverages `rlang`'s tidy evaluation framework.

Usage

```
arg2string(...)
```

Arguments

`...`	Arbitrary arguments representing R language objects or symbols.

Value

A character vector where each element is the string representation of the corresponding argument passed to the function. The order of the strings in the output matches the order of the arguments.

Examples

```
# returns the arguments as strings even though functions bmm() and brms() are not defined
arg2string(bmm('0.4.0'), brms('2.20.4'))
```

attr *Get attribute of an object*

Description

An infix operator to get the attribute of an object (equivalent to attr(x, "name", exact = TRUE))

Usage

```
x%a%which

x%a%which <- value

x%A%which

x%A%which <- value
```

Arguments

x	an object whose attributes are to be accessed.
which	a non-empty character string specifying which attribute is to be accessed.
value	an object, the new value of the attribute, or NULL to remove the attribute.

Details

These functions provide access to a single attribute of an object. The replacement form causes the named attribute to take the value specified (or create a new attribute with the value given).

Value

For the extractor, the value of the attribute matched, or NULL if no exact match is found and no or more than one partial match is found.

Examples

```
x <- 1
attr(x, "name") <- "John"
x%a%name
x%A%name

x%a%name <- "Alice"
attr(x, "name")

x <- 1:10
x%A%dim <- c(2,5)
x
```

available_packages *A character vector of available packages in the library*

Description

A character vector of available packages in the library

Usage

```
available_packages(path = .libPaths())
```

Arguments

path	A character vector of paths to search for the package. Default is the default library paths.

Value

A character vector of available package names in the library. If any package versions were installed via pkg_vload(), the version will be shown as "pkg-version"

collapse *Concatenate and collapse to a single string*

Description

Wrapper around `paste` with `collapse` argument set to `""`. This results in a single string.

Usage

```
collapse(..., sep = "")
```

Arguments

...	one or more R objects, to be converted to character vectors and concatenated.
sep	a character string to separate the terms.

Value

A single string.

Examples

```
x <- 1:10
res <- collapse(x)
identical(res, '12345678910')
```

component_size *Report Space Allocated for the Components of a List*

Description

Report Space Allocated for the Components of a List

Usage

```
component_size(x, units = "auto")
```

Arguments

x	A list object
units	A string specifying the units to use for the size. Default is "auto".

Value

A list with the same names as the input list, where each element is the size

Examples

```
x <- list(a = 1:10, b = 1:100)
component_size(x)
```

extract_pkg_fun_calls *Extract all calls to a function from package code*

Description

Given a function name and a package, this function will extract all calls to the said function from the package. The function can be any function name, either imported or defined in the package.

Usage

```
extract_pkg_fun_calls(pkg, fun)
```

Arguments

pkg	Name of the package from which to extract function calls
fun	Name of the function to extract calls to

Value

A character vector with the extracted function calls. Each value is of the form "fun(arg1=value1, arg2=value2, ...)"

Examples

```
extract_pkg_fun_calls("utils", "getOption")
```

`is_dir_empty` *Check if Directories are Empty*

Description

This function checks whether one or more directories are empty. An empty directory is one that contains no files or subdirectories. If the directory does not exist, it is considered as empty.

Usage

```
is_dir_empty(paths)
```

Arguments

`paths`	A character vector containing one or more file paths. Each path is checked to determine if the corresponding directory is empty.

Value

A logical vector where each element corresponds to a directory specified in `paths`. `TRUE` indicates that the directory is empty, and `FALSE` indicates that it is not.

Examples

```
## Not run:
# Create two temporary directories one of which is empty
library(fs)
dir1 <- tempfile()
dir2 <- tempfile()
dir_create(c(dir1, dir2))
dir_create(file.path(dir1, "subdir"))

# Check if the directories are empty (should return FALSE, TRUE)
is_dir_empty(c(dir1, dir2))

# Clean up
dir_delete(c(dir1, dir2))

## End(Not run)
```

nlist *Create Named List from Arguments*

Description

This function creates a named list from its arguments. If the arguments are named, those names are used in the resulting list. If some arguments are unnamed, the variable names themselves are used as names in the list. This can be useful for creating lists where the names are important for later indexing or manipulation, and ensures all elements in the list have names.

Usage

```
nlist(...)
```

Arguments

`...`	Arbitrary arguments to be included in the list. These can be named or unnamed. Unnamed arguments will be named based on their variable names.

Value

A list where each element corresponds to an argument passed to the function. Elements of the list are named based on either their original names or the names of the variables passed as arguments.

Examples

```
var1 <- 1
var2 <- 1:10
# This will return a list with names: c("a", "b", "var1", "var2")
nlist(a = 1, b = 2, var1, var2)
```

op-null-default *Default value for* NULL

Description

This infix function makes it easy to replace NULLs with a default value. It's inspired by the way that Ruby's or operation (||) works.

Usage

```
x %||% y
```

Arguments

x, y	If x is NULL, will return y; otherwise returns x.

Examples

```
1 %||% 2
NULL %||% 2
```

`packageOptions` *View the current or default global options for a package*

Description

`packageOptions()` scrapes the source code of a package to find all calls to `getOption()` and returns a tidied list of the current values of the options. Initial running time might be slow if a package contains a large amount of code. Repeated calls to the function will be significantly faster.

Usage

```
packageOptions(pkg, own_only = FALSE, max_length = 50, show_defaults = FALSE)
```

Arguments

`pkg`	Name of the package to extract options from
`own_only`	Logical. If `TRUE`, only options defined by the package itself will be returned. These are defined as options whose name starts with the package name, followed by a dot. E.g. `getOption("knitr.device.fallback")`. If `FALSE`, all options will be returned. Default is `FALSE`.
`max_length`	Integer. Controls the maximum length of individual the default values to be printed. You will rarely need to change this. However, if your results include strange output with very long strings defining default values via internal functions, you can decrease this value to suppress them. Default is 50.
`show_defaults`	Logical. If `TRUE`, the default values of the options will be printed as well. Default is `FALSE`.

Value

A named list of the current values of the options. This list can be saved to a variable and used with option() to restore the options to these values at a later time should you change them.

Examples

```
packageOptions("utils")
```

parse_pkg_version *Parse package name and version from a pkg('verions') call*

Description

Parse package name and version from a pkg('verions') call

Usage

```
parse_pkg_version(...)
```

Arguments

...	a number of calls to objects of type pkg('version') where pkg is the package name and version is the version number

Value

A list with two elements: names and versions. The names are the package names and the versions are of class 'package_version'. If the version is not specified, the version will be NA.

Examples

```
parse_pkg_version(brms("2.20.4"), bmm("0.4-0"), utils)
```

pkg_switch_default *Switch the default version of a package*

Description

`pkg_switch_default()` renames the folder of the default version of a package to pkg-version and renames the folder of the new default version to pkg. This allows the user to switch between different versions of the same package that are loaded by the default call to `library()`.

Usage

```
pkg_switch_default(pkg, new_default_version, path = .libPaths())
```

Arguments

pkg	The name of the package
new_default_version	The version of the package to be set as the default
path	A character vector of paths to search for the package. Default is the default library paths.

Value

Invisible TRUE if the default version was successfully switched, FALSE otherwise.

Examples

```
## Not run:
# install two versions of the xfun package
pkg_vload(stringr, stringr('1.0.0'))
# switch the default version of xfun to 0.2.0
pkg_switch_default("stringr", "1.0.0")

## End(Not run)
```

---

pkg_vavailable *Check if a specific package version is available in the library*

Description

pkg_vavailable() is an alternative to xfun::pkg_available() that checks for a specific version of the package rather than a minimal version. If the version is not specified, the function will check for the default version of the package.

Usage

```
pkg_vavailable(..., path = .libPaths(), exact = TRUE)
```

Arguments

...	One or more calls to the package name with version (if desired) specified in parantheses. E.g. brms("2.14.4") or brms or "brms"
path	A character vector of paths to search for the package. Default is the default library paths.
exact	Logical. If TRUE, the function will only return TRUE if the exact version is available. If FALSE, the function will return TRUE if the version is available or if a higher version is available. Default is TRUE.

Details

To check for a specific version, the function assumes that this version was installed using pkg_load(pkg(version)), which has created a folder named "pkg-version" in the library.

Value

a named list with the following elements:

- available: A logical vector indicating whether the package is available
- pkg_name: The name of the package
- pkg_version: The version of the package in the library
- pkg_version_specified: The version of the package specified in the call to pkg('version')
- pkg_folder: The folder name of the package in the library

Examples

```
## Not run:
pkg_vavailable(utils)
pkg_vavailable(xfun("0.1.0"))
pkg_vavailable(utils, brms("2.14.4"), xfun("0.1.0"))

# compare with xfun::pkg_available()
xfun::pkg_available("xfun", "0.1.0") # returns TRUE

## End(Not run)
```

---

pkg_vload *Load and/or install packages with specific versions*

Description

pkg_vload() attempts to load a package and, if it is not available, installs it. It can also install a specific version of a package. If the package is already installed, it will check if the version is the same as the one specified in the call. If the version is different, it will attempt to unload the package and install the specified version in a separate library, allowing the user to have multiple versions of the same package installed at the same time.

Usage

```
pkg_vload(
  ...,
  reload = FALSE,
  path = .libPaths(),
  repos = getOption("repos"),
  install_args = NULL
)
```

Arguments

...	One or more calls to the package name with version (if desired). The calls should be of the form `pkg('version')` where `pkg` is the package name and `version` is the version number. If the version is not specified, the function will check for the default version of the package.
reload	Logical. If `TRUE`, the function will attempt to unload the package and load it again, regardless of whether the version is the same as the one specified in the call. Default is `FALSE`. If the package is already loaded, it will be reloaded even if reload is `FALSE`, if the specified version is different from the one currently loaded.
path	A character vector of paths to search for the package. Default is the default library paths.
repos	A character vector of repository URLs to use for installing the package. Default is the value of `getOption("repos")`.
install_args	A list of additional arguments to be passed to `install.packages()` or `remotes::install_version()`. Default is `NULL`.

Value

This function does not return a value. Instead, it will stop the execution and display a message if the requirements are not met.

Examples

```
## Not run:
# Load the 'brms' package and install version 2.0.0 if it is not available
pkg_vload(brms("2.0.0"))

# Load multiple packages and install specific versions if they are not available
pkg_vload(brms("2.0.0"), utils)

## End(Not run)
```

---

require_pkg *Check Required Packages and Their Versions*

Description

This function checks if the required R packages are available and if their versions meet the specified minimum requirements. It will stop the execution and display a message if any required package is missing or does not meet the version requirement.

Usage

```
require_pkg(..., message_prefix = "Please install the following packages:")
```

Arguments

...	Variable arguments representing required package names and, optionally, their minimum versions. The versions should be specified immediately after the package names, in the format packageName(version).
message_prefix	A character string to be displayed before the message if the requirements are not met.

Value

This function does not return a value. Instead, it will stop the execution and display a message if the requirements are not met.

Examples

```
## Not run:
# Check if 'dplyr' and 'ggplot2' are installed (any versions):
require_pkg(dplyr, ggplot2)

# Check if 'dplyr' (version 1.0.0 or higher) and 'ggplot2' (version 8.3.0 or higher) are installed:
require_pkg(dplyr('1.0.0'), ggplot2('8.3.0'))

## End(Not run)
```

---

return_on_exit *Evaluate a function and return the result on exit in the calling environment*

Description

return_on_exit() can be called from within a parent function to set a return value when the parent function exits for whatever reason. This is useful when you want to specify a conditional return value even if a function exits abruptly.

Usage

```
return_on_exit(fun, ..., env = parent.frame())
```

Arguments

fun	Function to evaluate upon exit and return the result from the calling function
...	Arguments to pass to the function
env	Environment to evaluate the function in and to return from. The function will have access to the variables in this environment and their state at the time of the exit (not the time of the function call). If you want to pass variable values at the time of the function call, use ...

Value

The result of the function call

Examples

```
## Not run:
# function to evaluate on exit
f <- function(y, ...) {
  dots <- list(...)
  if ("x" %in% names(dots)) {
    x <- dots$x
  }
  x + y
}

# calling function
g <- function(...) {
 x <- 1    # current value of x; will not be used if it changes before g exists
 y <- 10   # value of y to pass to f, will be used as is

 # setup conditional return
 return_on_exit(f, y, ...)

 # do some work
 for (i in 1:1000000) {
   x <- i
   if (i == 100) stop("Error, but I will return something!")
 }


 # this will not be executed
 cat("This will not be printed")
 return("This will not be returned")
}

# calling g() will return 110
g()

# calling g() with x as an argument will return 30
# because x is passed to f() via the dots argument
g(x = 20)

## End(Not run)
```

---

stop2 *Wrappers around* stop *and* warning *that do not print the call stack*

Description

In addition they allow to use glue to create the error message

Usage

```
stop2(..., env.frame = -1)

warning2(..., env.frame = -1)
```

Arguments

...	zero or more objects which can be coerced to character (and which are pasted together with no separator) or a single condition object
env.frame	the environment to use for the error message if you use glue syntax. Default is -1, which is the calling environment

Value

Stops execution and prints the error message

Examples

```
## Not run:
stop2("This is an error message")
x <- 1
stop2("This is an error message with a variable: {x}")

## End(Not run)
```

---

stopifnot2 *Wrapper around stopifnot allowing for a custom error message*

Description

Wrapper around stopifnot allowing for a custom error message

Usage

```
stopifnot2(..., msg = NULL)

stopif(..., msg)

warnif(..., msg)
```

Arguments

...	zero or more expressions to be evaluated
msg	a custom error message to be printed if the expressions are not all true

strip_attributes *Remove all attributes of an object except those specified as protected*

Description

Remove all attributes of an object except those specified as protected

Usage

```
strip_attributes(x, protect = c("names", "row.names", "class"))
```

Arguments

x	an R object
protect	a character vector of attribute names to keep. Default is `c("names", "row.names", "class")`, which are the attributes that a data.frame has by default.

Value

An R object with all attributes removed except those specified in `protect`.

Examples

```
x <- data.frame(a = 1:10, b = 11:20)
attr(x, "remove_me") <- "I want to be removed"
attributes(x)
x <- strip_attributes(x, protect = c("names", "row.names", "class"))
attributes(x)
```

str_extract_nested_balanced *Extract a substring defined by a prefix, and a matched opening and closing character*

Description

Used to, for example, extract a function call from code, ignoring paranthesis within function arguments

Usage

```
str_extract_nested_balanced(string, prefix, opening = "(", closing = ")")
```

Arguments

string	A character vector
prefix	A string to match at the beginning of the substring
opening	A character to match as the opening character
closing	A character to match as the closing character

Details

There will be a match only if the function is not used as text in quotes immediately before the function call. For example, in the string "something myFunction(x = 1, y = mean(x)) otherFunction()", the function call to `myFunction` will be matched, but the function call in the string "var = \"myFunction(x = 1, y = mean(x))\"" or print("myFunction(x = 1, y = mean(x))") will not. This ensures that we do not retrieve examples or instructions about usage

Value

A character vector with the extracted substring

Examples

```
x <- "something myFunction(x = 1, y = mean(x)) otherFunction()"
str_extract_nested_balanced(x, "myFunction", "(", ")")
```

%+% *Concatenate strings python style*

Description

Equivalent to python's string concatenation via + operator

Usage

```
a %+% b
```

Arguments

a	an R object to be converted to a character vector.
b	an R object to be converted to a character vector.

Value

String concatenation of a and b

Examples

```
# adding a string and a vector
"name" %+% 1:10

# adding vector variables and strings
names <- c("John", "Sarah")
ages <- c(34,23)
res <- names %+% " is " %+% ages %+% " years old"
identical(res, c("John is 34 years old", "Sarah is 23 years old"))
```

Index

%A% *(attr)*, 3
%A%<- *(attr)*, 3
%a% *(attr)*, 3
%a%<- *(attr)*, 3
%+%, 17

arg2string, 2
attr, 3
available_packages, 4

collapse, 4
component_size, 5

extract_pkg_fun_calls, 5

is_dir_empty, 6

nlist, 7

op-null-default, 7

packageOptions, 8
packageOptions(), *8*
parse_pkg_version, 9
pkg_switch_default, 9
pkg_switch_default(), *9*
pkg_vavailable, 10
pkg_vavailable(), *10*
pkg_vload, 11

require_pkg, 12
return_on_exit, 13

stop2, 14
stopif *(stopifnot2)*, 15
stopifnot2, 15
str_extract_nested_balanced, 16
strip_attributes, 16

warnif *(stopifnot2)*, 15
warning2 *(stop2)*, 14

xfun::pkg_available(), *10*