Package: namer (via r-universe)

September 29, 2024

Title Easily Rename and Subset Objects by Name

Version 0.1.0

Description Contains convenience functions for naming. Select subsets
by name using matches or regular expressions. Rename objects
with regular expressions or paste.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests rlang, dplyr, vctrs

Repository https://hughjonesd.r-universe.dev

RemoteUrl https://github.com/hughjonesd/namer

RemoteRef v0.1.0

RemoteSha e8b6e7edb848d5267e44c215566d1ec14be1c3d1

Contents

namer-package *Manipulate objects by name*

Description

Contains convenience functions for naming. Select subsets by name using matches or regular expressions. Rename objects with regular expressions or paste.

Details

{namer} is a tiny r package containing convenience functions for manipulating objects by their names. Using these functions makes your code easier to read, and reduces duplication:

```
library(namer)

vec <- c(One = 1, Two = 2, Three = 3, Four = 4)

# Base R:
vec[startsWith(names(vec), "T")]
#>   Two Three
#>     2     3

# Clearer:
vec |> named_starting("T")
#>   Two Three
#>     2     3


# Base R:
some_names <- names(vec) %in% c("Two", "Three")
names(vec)[some_names] <- tolower(names(vec)[some_names])

# Clearer:
vec |> rename_in(c("Two", "Three"), tolower)
#>   One   two three  Four
#>     1     2     3     4


# Base R:
vec[sort(names(vec))]
#>  Four   One three   two
#>     4     1     3     2

# Clearer:
vec |> sort_by_name()
#>  Four   One three   two
#>     4     1     3     2
```

Functions that start with named return a subset of the original object:

```
vec <- c(One = 1, Two = 2, Three = 3, Four = 4)
vec |> named_in(c("Two", "Three", "Non-existent"))
#>   Two Three
#>     2     3
vec |> named_starting("T")
#>   Two Three
#>     2     3
vec |> named_like("[A-Z].*e$")
#>   One Three
#>     1     3
```

sort_by_name() sorts object by name:

```
sort_by_name(vec)
#>  Four   One Three   Two
#>     4     1     3     2
```

Functions that start with rename return the object with its names changed. You can use a named character vector:

```
vec |> rename_in(c("One", "Two"), c(one = "One", two = "Two"))
#>   one   two Three  Four
#>     1     2     3     4
```

Or an unnamed character vector:

```
vec |> rename_in(c("One", "Two"), c("First", "Second"))
#>  First Second  Three   Four
#>      1      2      3      4
```

Or a function:

```
vec |> rename_all(tolower)
#>   one   two three  four
#>     1     2     3     4
vec |> rename_starting("T", tolower)
#>   One   two three  Four
#>     1     2     3     4
```

Or you can use a one-sided formula, as in purrr:

```
vec |> rename_in(c("One", "Two"), ~paste(.x, 1:2, sep = "."))
#> One.1 Two.2 Three  Four
#>     1     2     3     4
```

Or use a regular expression with rename_gsub:

```
vec |> rename_gsub("[aeiou]", "e")
#>   One   Twe Three  Feer
#>     1     2     3     4
```

Or match names from old to new with `rename_lookup`:

```
df <- data.frame(
        old = c("One", "Two", "Three", "Four"),
        new = c("A", "B", "C", "D")
      )
vec |> rename_lookup(df$old, df$new)
#> A B C D
#> 1 2 3 4
```

Installation:

You can install from R-universe:

```
install.packages("namer", repos = c("https://hughjonesd.r-universe.dev",
                 "https://cloud.r-project.org"))
```

Or install the development version from GitHub:

```
# install.packages("remotes")
remotes::install_github("hughjonesd/namer")
```

Author(s)

Maintainer: David Hugh-Jones <`davidhughjones@gmail.com`>

`named_in` *Subset objects by name*

Description

Subset objects by name

Usage

```
named_in(x, y)

not_named_in(x, y)
```

Arguments

x	An object with names.
y	A vector of names.

Details

`named_in(x, y)` is similar to `x[y]` except that:

- unmatched elements of `y` do not return an `NA` element;
- elements are returned in their original order within `x`.

`not_named_in(x, y)` returns elements of `x` whose name is not an element of `y`.

Value

For `named_in`: `x[names(x) %in% y]`.

For `not_named_in`: `x[! names(x) %in% y]`.

Examples

```
vec <- c(one = 1, two = 2, three = 3, four = 4)
vec |> named_in(c("two", "one", "three", "five"))
vec |> not_named_in(c("two", "three"))
```

`named_like` *Subset objects by name using a regular expression*

Description

Subset objects by name using a regular expression

Usage

```
named_like(x, pattern, ...)

not_named_like(x, pattern, ...)
```

Arguments

`x`	An object with names.
`pattern`	A regular expression string (see regex).
`...`	Passed in to `grepl()`.

Value

For `named_like`: `x[grepl(pattern, names(x), ...)]`.

For `not_named_like`: `x[! grepl(pattern, names(x), ...)]`.

Examples

```
vec <- c(one = 1, two = 2, three = 3, four = 4)
vec |> named_like("^t")
vec |> not_named_like("e$")
```

named_starting *Subset objects by name using an initial substring*

Description

Subset objects by name using an initial substring

Usage

```
named_starting(x, prefix)
```

Arguments

x	An object with names.
prefix	A character string

Value

`x[startsWith(names(x), prefix)]`

Examples

```
vec <- c(one = 1, two = 2, three = 3, four = 4)
vec |> named_starting("t")
```

other-resources *Other useful resources for manipulating names*

Description

There are several existing functions for working with names in R.

Details

Obviously, `base::names()` gets an object's names and `names<-` sets them.

`stats::setNames()` directly returns the object after setting names.

`base::make.names()` turns a character vector into syntactically valid names. `vctrs::vec_as_names()` does the same thing, r-lib style.

`base::make.unique()` makes elements of a character vector unique by appending sequence numbers to duplicates.

`rlang::set_names()` is like `setNames()` but also takes a function to transform names.

`rlang::names2()` is like `names()` but returns a character vector of `""` rather than `NULL` if an object has no `names` attribute.

`dplyr::rename()` and friends change the names of data frames or tibbles, but not other objects.

`https://principles.tidyverse.org/names-attribute.html` is a principled framework for thinking about names in R.

`rename_all` *Rename all names*

Description

Rename all names

Usage

```
rename_all(x, f, ...)
```

Arguments

`x`	An object with names.
`f`	A function, one-sided formula, or character vector.
`...`	Passed into `f`. An error is thrown if `...` is non-empty when `f` is a character vector.

Details

- If `f` is a function it will be applied to the selected names. If it is a formula and the 'rlang' package is installed, it will be converted to a function by `rlang::as_function()`, then applied.
- If `f` is a named character vector like `c(new_name = "old_name", ...)` then `"old_name"` will become `"new_name"`, as in `dplyr::rename()`.
- If `f` is an unnamed character vector, these will be the new names in order.

Value

The renamed object.

Examples

```
vec <- c("One" = 1, "Two" = 2, "Three" = 3, "Four" = 4)
vec |> rename_all(tolower)
```

`rename_in` *Rename names in a set*

Description

Elements of `x` whose names are in `nm` will be renamed.

Usage

```
rename_in(x, nm, f, ...)
```

Arguments

`x`	An object with names.
`nm`	A character vector passed to `%in%`.
`f`	A function, one-sided formula, or character vector.
`...`	Passed into `f`. An error is thrown if `...` is non-empty when `f` is a character vector.

Details

- If `f` is a function it will be applied to the selected names. If it is a formula and the 'rlang' package is installed, it will be converted to a function by `rlang::as_function()`, then applied.
- If `f` is a named character vector like `c(new_name = "old_name", ...)` then `"old_name"` will become `"new_name"`, as in `dplyr::rename()`.
- If `f` is an unnamed character vector, these will be the new names in order.

Value

The renamed object.

Examples

```
vec <- c("One" = 1, "Two" = 2, "Three" = 3, "Four" = 4)
vec |> rename_in(c("Two", "Three"), paste0, "x")
```

`rename_like` *Rename names that match a regular expression*

Description

Rename names that match a regular expression

Usage

```
rename_like(
  x,
  pattern,
  f,
  ...,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE
)
```

Arguments

`x`	An object with names.
`pattern`	A regular expression string (see regex).
`f`	A function, one-sided formula, or character vector.
`...`	Passed into `f`. An error is thrown if `...` is non-empty when `f` is a character vector.
`ignore.case`, `perl`, `fixed`, `useBytes`	Passed into `grepl()`.

Details

- If `f` is a function it will be applied to the selected names. If it is a formula and the 'rlang' package is installed, it will be converted to a function by `rlang::as_function()`, then applied.
- If `f` is a named character vector like `c(new_name = "old_name", ...)` then `"old_name"` will become `"new_name"`, as in `dplyr::rename()`.
- If `f` is an unnamed character vector, these will be the new names in order.

Value

The renamed object.

Examples

```
vec <- c("One" = 1, "Two" = 2, "Three" = 3, "Four" = 4)
rename_like(vec, "^T", paste0, "x")
```

`rename_lookup` *Rename by looking up names in a table*

Description

This is useful when you have a vector of old names and a vector of new names, or columns in a data frame.

Usage

```
rename_lookup(x, old, new, warn = FALSE)
```

Arguments

`x`	An object with names.
`old`	Character vector. Existing names will be found using `match(names(x), old)`
`new`	Character vector. A vector of new names to replace corresponding elements in `old`.
`warn`	Logical. Warn if any names are unmatched?

Details

Unmatched names are left unchanged.

Value

x renamed according to `names(x) <- new[match(names(x), old)]`.

Examples

```
df <- data.frame(
        old = c("One", "Two", "Three"),
        new = c("New", "Newer", "Newest")
      )
vec <- c("One" = 1, "Two" = 2, "Three" = 3, "Four" = 4)
vec |> rename_lookup(df$old, df$new)
```

`rename_remove_prefix` *Remove a prefix or suffix from names*

Description

Remove a prefix or suffix from names

Usage

```
rename_remove_prefix(x, prefix)

rename_remove_suffix(x, suffix)
```

Arguments

`x`	An object with names.
`prefix, suffix`	A length 1 character vector to remove.

Details

- If `f` is a function it will be applied to the selected names. If it is a formula and the 'rlang' package is installed, it will be converted to a function by `rlang::as_function()`, then applied.
- If `f` is a named character vector like `c(new_name = "old_name", ...)` then `"old_name"` will become `"new_name"`, as in `dplyr::rename()`.
- If `f` is an unnamed character vector, these will be the new names in order.

Value

`x` with the prefix or suffix removed from `names(x)`.

Examples

```
vec <- c("a.1" = 1, "aaa.1" = 2, "other" = 3, ".1" = 4)
vec |> rename_remove_suffix(".1")

vec <- c("x.a" = 1, "x.aaa" = 2, "other" = 3, "x." = 4)
vec |> rename_remove_prefix("x.")
```

rename_starting *Rename names that start with a prefix*

Description

Rename names that start with a prefix

Usage

```
rename_starting(x, prefix, f, ...)
```

Arguments

`x`	An object with names.
`prefix`	A string.
`f`	A function, one-sided formula, or character vector.
`...`	Passed into `f`. An error is thrown if `...` is non-empty when `f` is a character vector.

Details

- If `f` is a function it will be applied to the selected names. If it is a formula and the 'rlang' package is installed, it will be converted to a function by `rlang::as_function()`, then applied.
- If `f` is a named character vector like `c(new_name = "old_name", ...)` then `"old_name"` will become `"new_name"`, as in `dplyr::rename()`.
- If `f` is an unnamed character vector, these will be the new names in order.

Value

The renamed object.

Examples

```
vec <- c("One" = 1, "Two" = 2, "Three" = 3, "Four" = 4)
vec |> rename_starting("T", \(x) gsub(x, "[aeiou]", "e"))
```

`rename_sub` *Rename using a regular expression*

Description

Rename using a regular expression

Usage

```
rename_sub(x, pattern, replacement, ...)

rename_gsub(x, pattern, replacement, ...)
```

Arguments

`x`	An object with names.
`pattern`, `replacement`, `...`	Passed into `sub()` or `gsub()`.

Details

These functions always apply to all names.

Value

The renamed object.

Examples

```
vec <- c("One" = 1, "Two" = 2, "Three" = 3, "Four" = 4)
vec |> rename_gsub("[aeiou]", "e")
vec |> rename_sub("([aeiou])", "-\\1-")
```

`rename_where` *Rename names indexed by a subset*

Description

Rename names indexed by a subset

Usage

```
rename_where(x, index, f, ...)
```

Arguments

`x`	An object with names.
`index`	A logical or numeric index.
`f`	A function, one-sided formula, or character vector.
`...`	Passed into `f`. An error is thrown if `...` is non-empty when `f` is a character vector.

Details

- If `f` is a function it will be applied to the selected names. If it is a formula and the 'rlang' package is installed, it will be converted to a function by `rlang::as_function()`, then applied.
- If `f` is a named character vector like `c(new_name = "old_name", ...)` then `"old_name"` will become `"new_name"`, as in `dplyr::rename()`.
- If `f` is an unnamed character vector, these will be the new names in order.

Value

The renamed object.

Examples

```
vec <- c("One" = 1, "Two" = 2, "Three" = 3, "Four" = 4)
rename_where(vec, 2:3, paste0, 2:3)
```

`sort_by_name` *Sort an object by its names*

Description

Sort an object by its names

Usage

```
sort_by_name(x, decreasing = FALSE)
```

Arguments

`x`	An object with names.
`decreasing`	Logical. Should sort order be increasing or decreasing?

Value

`x[sort(names(x), decreasing = decreasing)]`

Examples

```
vec <- c(one = 1, two = 2, three = 3, four = 4)
sort_by_name(vec)
sort_by_name(vec, decreasing = TRUE)
```

Index