Package: reuseme (via r-universe)

October 26, 2024

Title Collections of Utility Functions to Work Across Projects

Version 0.0.2.9007

Description Allows you to browse current projects, rename files safely, add screenshots to project on Windows. It is also my personal library and contains wrapper around common functions, from dplyr and readxl. It takes advantage of cli hyperlinks. Finally, it provides a custom print method for tibbles, inspired by janitor, and readr.

License MIT + file LICENSE

URL https://olivroy.github.io/reuseme/, https://github.com/olivroy/reuseme

BugReports https://github.com/olivroy/reuseme/issues

Depends R (>= 4.1)

Imports cli, dplyr (>= 1.1.1), fs, glue, kit, lifecycle, purrr, rlang (>= 1.1.0), rprojroot, rstudioapi, scales, stringr, tibble, usethis (>= 2.2.0), vctrs

Suggests clipr, curl, gert, gt, magick, pillar, testthat (>= 3.2.1), withr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Language en-CA

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

RemoteUrl https://github.com/olivroy/reuseme

RemoteRef main

RemoteSha 530269e02a60bccb708be6f7ffc974a7ac0c826a

Contents

`active_rs_doc_copy` *Copy the active document to the same location*

Description

The goal is to provide things that RStudio or usethis doesn't provide natively.

Usage

```
active_rs_doc_copy(new = NULL, ..., old = NULL)
```

Arguments

`new`	The new file name, that will be copied in the same directory as the active document For `active_rs_doc_move()`, a directory.
`...`	These dots are for future extensions and must be empty.
`old`	The old name, defaults to the active document.

Details

For example, `active_rs_doc_rename()` will not happen, because it is already easy to do so via the RStudio IDE.

Value

The new file name

See Also

`rename_files2()`

Other document manipulation helpers: `active_rs_doc_delete()`

`active_rs_doc_delete` *Delete the active RStudio document safely*

Description

[Experimental]

Gathers informative summary about the document you are about to delete.

Will delete more easily if file name starts with `temp-`, if file is untracked and recent.

Usage

```
active_rs_doc_delete()
```

Value

Called for side-effects. The document content invisibly if deleting and reason.

See Also

Other document manipulation helpers: `active_rs_doc_copy()`

Examples

```
active_rs_doc_delete()
```

`active_rs_doc_move` *Move the active document to another directory*

Description

Wrapper around `rename_files2()`, but shortcut to allow renaming the active file.

Usage

```
active_rs_doc_move(new = NULL, old = NULL, ...)
```

Arguments

`new`	A new directory
`old`	The old file (defaults to the active RStudio document.)

`...`	Arguments passed on to `rename_files2`
	`overwrite` whether to overwrite `new` if it already exists. Be careful.
	`force` **[Deprecated]** Use `warn_conflicts` instead of `force = TRUE`
	`action` One of `"rename"` or `"test"`
	`warn_conflicts` One of
	• `"default"`: will be check more thoroughly depending on the situation. If only moving directory, and `"all"` otherwise.
	• `"all"` (larger scope: if `old = "data/my-streets.csv|my_streets"` will check for objects named `my_streets`, other files like `my-streets.R`, etc.),
	• `"exact"` will only search for `"data/my-streets.csv"` in documents `"none"` will not search for references in documents and will rename.

Value

`new` if renaming succeeded. Mostly called for its side-effects

`active_rs_doc_nav` *Open Files Pane at current document location*

Description

Easily navigate to active file document.

Usage

```
active_rs_doc_nav(path = active_rs_doc())
```

Arguments

`path`	A path to file to navigate to (default active document).

Details

Wrapper around executeCommand("activateFiles") + `rstudioapi::filesPaneNavigate()` + `rstudioapi::getActiveDoc`

Value

NULL, called for its side effects.

`browse_pkg` *Browse pkgdown site if it exists*

Description

A wrapper around `usethis::browse_package()` that aims at identifying the package website. It looks up for a link in DESCRIPTION.

Usage

```
browse_pkg(
  package = NULL,
  open = FALSE,
  news_only = FALSE,
  ref_only = FALSE,
  vignettes_show = TRUE
)
```

Arguments

`package`	Name of package. If `NULL`, the active project is targeted, regardless of whether it's an R package or not. If `"<org>/<repo>"` is supplied, will jump to GitHub homepage.
`open`	Whether to open the pkgdown site in the browser.
`news_only`	Should only the news link be shown?
`ref_only`	Should only the reference index be show?
`vignettes_show`	Should the vignette information be displayed on console?

Value

The package website URL, invisibly. (If `ref_only`, or `news_only`, the reference index URL or Changelog URL).

Examples

```
browse_pkg("reuseme")
browse_pkg()
```

case_if_any *case-when, but checks for all matches, returns a character*

Description

Each case is evaluated for **all** cases and a character vector match for each element determines the corresponding value in the output vector. If no cases match, the `.default` is used. The function allows you to assign multiple values to a character value, which can be very handy for EDA.

Usage

```
case_if_any(..., .default = "", .sep = ";", .drop_empty = TRUE)
```

Arguments

`...`	<`dynamic-dots`> A sequence of two-sided formulas. The left hand side (LHS) determines which values match this case. The right hand side (RHS) provides the replacement value. The LHS inputs must evaluate to logical vectors. The RHS inputs will be coerced to their common type. All inputs will be recycled to their common size. That said, we encourage all LHS inputs to be the same size. Recycling is mainly useful for RHS inputs, where you might supply a size 1 input that will be recycled to the size of the LHS inputs. `NULL` inputs are ignored.
`.default`	The value used when all of the LHS inputs return either `FALSE` or `NA`. `.default` must be size 1 or the same size as the common size computed from `...`. `.default` participates in the computation of the common type with the RHS inputs. `NA` values in the LHS conditions are treated like `FALSE`, meaning that the result at those locations will be assigned the `.default` value. To handle missing values in the conditions differently, you must explicitly catch them with another condition before they fall through to the `.default`. This typically involves some variation of `is.na(x) ~ value` tailored to your usage of `case_when()`. If `NULL`, the default, a missing value will be used.
`.sep`	the separator between answers. (default is `;`), can't be a substring of any of the text
`.drop_empty`	drop if no match is returned. (Defaults to `TRUE` for legibility), but if `FALSE`, can be used more easily with `tidyr::separate_wider/longer_delim()`

Value

A vector with the same size as the common size computed from the inputs in `...` and the same type as the common type of the RHS inputs in `...`.

Examples

```
case_if_any(
  mtcars$vs == 1 ~ "vs = 1",
  mtcars$mpg > 150 ~ "I have mpg > 150"
)
case_if_any(
  mtcars$vs == 1 ~ "Woww",
  mtcars$mpg > 15 ~ "QW",
  mtcars$qsec > 18 ~ "ooh lalal",
  .sep = ";",
  .default = NA
)
```

`count_pct` *Count observations by group and compute percentage*

Description

`count_pct()` lets you quickly count the unique values of one or more variables: `df |> count_pct(a, b)` It calculates the percentage by group afterwards

Usage

```
count_pct(
  .data,
  ...,
  label = FALSE,
  accuracy = NULL,
  name = NULL,
  sort = FALSE
)
```

Arguments

`.data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`...`	<`data-masking`> Variables to group by.
`label`	A logical, If `TRUE`, will return the vector as character
`accuracy`	A number to round to. Use (e.g.) `0.01` to show 2 decimal places of precision. If `NULL`, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.

`name`	The name of the new column in the output. If omitted, it will default to `n`. If there's already a column called `n`, it will use `nn`. If there's a column called `n` and `nn`, it'll use `nnn`, and so on, adding `n`s until it gets a new name.
`sort`	If `TRUE`, will show the largest groups at the top.

Details

Wrapper function around `dplyr::count()`

Value

An object of the same type as `.data`. `count()` and `add_count()` group transiently, so the output has the same groups as the input.

See Also

Other dplyr extensions: `slice_min_max()`

Examples

```
count_pct(mtcars, cyl)
mtcars |>
  dplyr::group_by(vs) |>
  count_pct(cyl)

mtcars |>
  dplyr::group_by(vs) |>
  count_pct(cyl, label = TRUE)

mtcars |>
  dplyr::group_by(vs) |>
  count_pct(cyl, label = TRUE, accuracy = 0.1)
```

`eda-identity` *Helpers that return the same value*

Description

They all share the `*_identity` suffix, they are silent in non-interactive sessions. They are very handy to create clickable hyperlinks that do not modify the current state of the analysis.

They are inspired by `pillar::glimpse()`, `tibble::view()`.

Look at the original functions for the other parameters.

Usage

```
names_identity(x, nrows = NULL, extra_msg = NULL)

count_identity(
  x,
  ...,
  sort = TRUE,
  name = NULL,
  nrows = NULL,
  extra_msg = NULL
)

mutate_identity(
  x,
  ...,
  .keep = NULL,
  .before = NULL,
  nrows = NULL,
  extra_msg = NULL
)

slice_identity(
  x,
  ...,
  .by = NULL,
  .preserve = FALSE,
  nrows = NULL,
  extra_msg = NULL
)

slice_min_identity(
  x,
  order_by,
  ...,
  n,
  prop,
  by = NULL,
  with_ties = TRUE,
  na_rm = TRUE,
  nrows = NULL,
  extra_msg = NULL
)

slice_max_identity(
  x,
  order_by,
  ...,
  n,
  prop,
  by = NULL,
  with_ties = TRUE,
  na_rm = TRUE,
  nrows = NULL,
  extra_msg = NULL
)

arrange_identity(x, ..., .by_group = FALSE, nrows = NULL, extra_msg = NULL)

distinct_identity(
  x,
  ...,
  .keep_all = FALSE,
  .arrange = FALSE,
  nrows = NULL,
  extra_msg = NULL
)

filter_identity(x, ..., .by = NULL, nrows = NULL, extra_msg = NULL)

slice_sample_identity(
  x,
  ...,
  n,
  prop,
  by = NULL,
  weight_by = NULL,
  replace = FALSE,
  nrows = NULL,
  extra_msg = NULL
)

filter_if_any_identity(
  x,
  ...,
  .by = NULL,
  .keep_new_var = FALSE,
  nrows = NULL,
  extra_msg = NULL
)

slice_min_max_identity(
  x,
  order_by,
  ...,
  n,
  prop,
  by = NULL,
  with_ties = TRUE,
  na_rm = FALSE,
  each = TRUE,
  ascending = TRUE,
  nrows = NULL,
  extra_msg = NULL
)

slice_group_sample_identity(
  x,
  group_var = NULL,
  n_groups = 1,
  nrows = NULL,
  extra_msg = NULL
)
```

Arguments

`x`	The main object (a data.frame, but some functions accept a vector.) (aka `.data` in some `dplyr` functions, but naming it `x` throughout.)
`nrows`	Number of rows to print.
`extra_msg`	A character vector of observations that will print to console, notes taken related to the transformation.
`name`, `sort`, `.keep_all`, `.by`, `by`, `n_groups`, `group_var`, `...`, `n`, `prop`, `with_ties`, `order_by`, `.keep`, `.before`, `each`, `na_rm`, `weight_by`, `replace`, `.by_group`, `.keep_new_var`, `.preserve`, `ascending`	Check original functions.
`.arrange`	Should arrange output?

Value

`x`, the original input is (invisibly) returned. (allowing the `*_identity()` functions to be used in a pipeline) will print `extra_msg` to the console in interactive sessions.

Use cases / advantages

- Like many other reuseme functions, they are most useful in interactive sessions
- print the result in interactive sessions (quiet in non-interactive.)
- Create runnable hyperlinks (In August 2023, RStudio forbids runnable hyperlinks of base functions, or non-package functions. (i.e. that don't have `::`))
- Use in pipelines to explore the data
- Use `rlang::is_interactive()` over `base::interactive()` as it's easier to control and test with `options(rlang_interactive)`
- Use the original functions for your final results.
- `count_identity()` also prints percentages.
- `slice_identity()` can be useful to resolve many-to-many warnings from dplyr join functions.

Caution

- Don't put those at the end of a pipeline
- Don't name the first argument, to avoid conflicts in case a column in the data is named `x`.
- Some functions have small tweaks
  - `mutate_identity()` only prints the distinct values, and uses `.keep = "used"`, `.before = 0`, unless specified to improve the display.
  - `count_identity()` is a wrapper of `count_pct()` (itself a wrapper of `dplyr::count()`),
  - `count_identity()` may fail if there is already a variable named `n`.
  - `slice_min/max_identity()` relocates the target column at the beginning.
  - `filter_identity()` prints a short message if no rows are returned.

See Also

- `dplyr::distinct()`
- `dplyr::filter()`
- `dplyr::slice()`
- `dplyr::mutate()`
- `dplyr::arrange()`
- `count_pct()`
- `slice_min_max()`
- `slice_group_sample()`

Examples

```
withr::local_options(rlang_interactive = TRUE)
# Workflow to explore mtcars
mtcars |>
  filter_identity(mpg > 21, extra_msg = c("Wow, these rows are very interesting.")) |>
  count_identity(
    vs,
    extra_msg = c(
      "Woo, there are 14 obs with vs = 1, 18 obs with vs = 0",
      "The split is 56%-43%"
    )
  ) |>
  dplyr::filter(disp > 150) # after all, I need only disp > 150
```

`extract_cell_value` *Elegant wrapper around filter and pull*

Description

It can be very useful when trying to extract a value from somewhere, and you have one col that represents the unique id.

Usage

```
extract_cell_value(
  data,
  var,
  filter,
  name = NULL,
  length = NULL,
  unique = FALSE
)
```

Arguments

`data`	A data.frame
`var`	A variable specified as: • a literal variable name • a positive integer, giving the position counting from the left • a negative integer, giving the position counting from the right. The default returns the last column (on the assumption that's the column you've created most recently). This argument is taken by expression and supports quasiquotation (you can unquote column names and column locations).
`filter`	the filter
`name`	The variable for the name (by default, will look for `rownames`), can be quoted (safer).
`length`	A fixed length to check for the output
`unique`	A logical. Should return unique values?

Value

A (named) character vector (if name is specified)

Examples

```
# extract the skin_color for C-3PO
extract_cell_value(
  data = dplyr::starwars,
  var = skin_color,
  filter = name == "C-3PO",
  length = 1 # ensure the length will be 1.
)
# will return a named vector of mpg (as mtcars has rownames.)
mtcars |>
  extract_cell_value(
    var = mpg,
    filter = vs == 0
  )

# Extract hair color for all people
extract_cell_value(
  data = dplyr::starwars,
  var = skin_color,
  filter = TRUE,
  name = "name" # ensure it is a named vector that corresponds to their unique ID
)
```

`file_move_temp_auto` *Move temporary file automatically from the R console*

Description

It works well when you have no API to download a file, but still want a fast R implementation.

Usage

```
file_move_temp_auto(destdir)
```

Arguments

`destdir`	The desired directory to send this to

See Also

`file_rename_auto()`

`file_rename_auto` *Move file automatically between folders*

Description

- `file_rename_auto()` automatically renames your file to a better name while keeping the same folder structure
- `file_move_dir_auto()` automatically moves your file while keeping the same file name

Usage

```
file_rename_auto(new_name, old_file = .Last.value)

file_move_dir_auto(new_dir, old_file = .Last.value)
```

Arguments

`new_name, new_dir`	New directory or file name (without extension)
`old_file`	The old file name

Value

The new full path name, invisibly, allowing you to call the functions another time.

Advantages

Instead of calling `fs::file_move("path/to/dir/file.R", "path/to/dir/new-file.R")`, you can just call `file_rename_auto("new-file", "path/to/dir/file.R")`

Instead of calling `fs::file_move("path/to/dir/file.R", "path/to/new-dir/file.R")`, you can just call `file_move_auto("new-dir", "path/to/dir/file.R")`

If the functions are used in conjunction with `file_move_temp_auto()`,

`filter_detect` *Filter rows by pattern*

Description

Shortcut for `dplyr::filter()` and `stringr::str_detect()`

Usage

```
filter_detect(.data, pattern, .cols = dplyr::everything(), ignore.case = TRUE)
```

Arguments

`.data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.
`pattern`	character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector. Coerced by `as.character` to a character string if possible. If a character vector of length 2 or more is supplied, the first element is used with a warning. Missing values are allowed except for `regexpr`, `gregexpr` and `regexec`.
`.cols`	<`tidy-select`> Columns to transform. You can't select grouping columns because they are already automatically handled by the verb (i.e. `summarise()` or `mutate()`).
`ignore.case`	if `FALSE`, the pattern matching is *case sensitive* and if `TRUE`, case is ignored during matching.

Value

A data frame with relocated columns at first

Examples

```
# don't specify column
dplyr::band_members |>
  filter_detect("Beatles")
# specify columns
dplyr::band_members |>
  filter_detect("Beatles", band)
```

`filter_if_any` *Keep rows that match one of the conditions*

Description

The `filter_if_any()` function is used to subset a data frame, retaining all rows that satisfy **at least one of** your conditions. To be retained, the row must produce a value of `TRUE` for **one of the conditions**. Note that when a condition evaluates to `NA` the row will be dropped, (hence this function) unlike base subsetting with `[`.

Usage

```
filter_if_any(.data, ..., .by = NULL, .keep_new_var = FALSE)
```

Arguments

`.data`	A data frame
`...`	<`data-masking`> Expressions that return a logical value, and are defined in terms of the variables in `.data`. If multiple expressions are included, they are combined with the `|` operator. Only rows for which **one of the conditions** evaluate to `TRUE` are kept.
`.by`	See dplyr::dplyr_by.
`.keep_new_var`	If `TRUE`, will remove newly created variables.

Details

The reason to be of this function is to simplify a call like

```
# with dplyr::filter
dat |> dplyr::filter(vs == 1 | is.na(vs))
data |>
  dplyr::mutate(cond1 = vs == 1, cond2 = is.na(vs)) |>
  dplyr::filter(dplyr::if_any(starts_with("cond")))
dat |> filter_if_any(vs == 1, is.na(vs))
```

Basically, this is just a shortcut to `mutate(.data, new_lgl_vars)` + `filter(if_any(new_lgl_vars))` + `select(-new_lgl_vars)`. It allows mutate_like syntax in `filter(if_any(...))`.

Caution: still doesn't work with `dplyr::across()`, use the regular `filter(if_any())` syntax.

Value

An object of the same type as `.data`. The output has the following properties:

- Rows are a subset of the input, but appear in the same order.
- Columns are not modified (if `.keep_new_var` = `FALSE`.
- Data frame attributes are preserved.

Examples

```
mtcars |> dplyr::filter(cyl > 5 | mpg == 2)
mtcars |> dplyr::filter(!(!cyl > 5 & !mpg == 2))
mtcars |> filter_if_any(cyl > 5, vs == 0)
```

named-base *Helpers that can return a named vector*

Description

Base R keeps names in various places, but drops them elsewhere These functions are some that I use frequently, like `max`, or `unique`

Usage

```
min_named(x, na.rm = FALSE, all_matches = FALSE)

max_named(x, na.rm = FALSE, all_matches = FALSE)

unique_named(x)
```

Arguments

`x`	A vector
`na.rm`	Should remove `NA`?
`all_matches`	If `FALSE` (default), will only return the first match, similar to `which.min()` / `which.max()` If `TRUE`, will return a named vector of all values corresponding to the max value.

Value

- `min/max_named(all_matches = FALSE)`, a named vector of length 1.
- Otherwise, a named vector.

Examples

```
max_named(c("this guy" = 2, "that guy" = 3))

unique_named(c("this guy" = 2, "that guy" = 3, "this guy" = 2))
# returns the same as base R for unnamed input
unique_named(c(1, 2, 3, 3))
# returns all values
min_named(c("x" = 1, "y" = 1, "ww" = 2), all_matches = FALSE)

# TODO is usable with `extract_cell_value()`
```

na_if2 *Transform to NA any of the condition*

Description

This function is similar to `dplyr::na_if()`, but it has 2 differences. the values of `y` are never recycled. There are two ways to provide the condition. As values or as a logical vector.

Usage

```
na_if2(x, values, expr)
```

Arguments

`x`	A vector.
`values`	A vector of values. If the length of values = 1, it is actually the preferable to use `dplyr::na_if()` for clarity.
`expr`	A logical vector same length as x

Value

`x` with `NA` values when required.

Examples

```
vec <- c(0, 1, 1, 2)
vec2 <- c("Here", "not", NA, "Here")
# NA all 2s
# You can actually use dplyr::na_if() in this case
dplyr::na_if(vec, 2)
# NA all 1 and 2
na_if2(vec, c(1, 2))
na_if2(vec, expr = vec2 == "Here")
```

open_rs_doc *Open a Document in RStudio*

Description

Wrapper around `rstudioapi::documentOpen()`, but with `fs paths`, for consistency. If the file could not be opened, a clickable hyperlink is displayed.

Usage

```
open_rs_doc(path, line = -1L, col = -1L, move_cursor = TRUE)

active_rs_doc()
```

Arguments

`path`	The path to the document.
`line`	The line in the document to navigate to.
`col`	The column in the document to navigate to.
`move_cursor`	Boolean; move the cursor to the requested location after opening the document?

Details

- `active_rs_doc()` is a wrapper around `rstudioapi::documentPath()` that handles unsaved files gracefully

Value

Invisibly returns the document id

Examples

```
if (FALSE) {
  # open the fictious file.R at line 5
  open_rs_doc("file.R", line = 5)
}
```

`outdated_pkgs` *Looks for outdated packages*

Description

Only checks for binaries, which has the advantage of not notifying you when packages are newly available on CRAN, but without the binaries available. Installing packages from source can be longer, hence the advantage of waiting a few days for binaries.

It takes advantage of pak's capacities to allow you to install packages on Windows without restarting session.

Usage

```
outdated_pkgs(type = c("binary", "source"))
```

Arguments

`type`	Type of package to look for (`"binary"` or `"source"`)

Value

A list of packages that can be updated, with links to news, the pkgdown site.

Examples

```
outdated_pkgs()
```

`outline` *Print interactive outline of file sections*

Description

The outline functions return a data frame that contains details of file location.

It also includes a print method that will provide a console output that will include clickable hyperlinks in RStudio (or if your terminal supports it). It works with both (qR)md and R files.

Outline elements include

- Any code section
- function definition (not shown in console by default)
- `TODO` items
- Parse cli hyperlinks
- Plot or table titles
- Figures caption in Quarto documents (limited support for multiline caption currently)
- Test names
- Indicator of recent modification
- Colored output for
- URL and gh issue detection and link creation.

By default

- `file_outline()` prints the outline the active document if in RStudio
- `proj_outline()` prints the outline of the active project if in RStudio
- `dir_outline()` prints the outline of the active working directory by default or

Usage

```
file_outline(
  path = active_rs_doc(),
  pattern = NULL,
  alpha = FALSE,
  print_todo = TRUE,
  recent_only = FALSE
)

proj_outline(
  path = active_rs_proj(),
  pattern = NULL,
  dir_tree = FALSE,
  alpha = FALSE,
  recent_only = FALSE
)

dir_outline(
  path = ".",
  pattern = NULL,
  dir_tree = FALSE,
  alpha = FALSE,
  recent_only = FALSE,
  recurse = FALSE
)
```

Arguments

`path`	A character vector of file paths, a project. Defaults to the active file, project or directory.
`pattern`	A string or regex to search for in the outline. If specified, will search only for elements matching this regular expression. The print method will show the document title for context. Previously `regex_outline`
`alpha`	Whether to show in alphabetical order
`print_todo`	Should include TODOs in the file outline? If `FALSE`, will print a less verbose output with sections.
`recent_only`	Show outline for recent files
`dir_tree`	If `TRUE`, will print the `fs::dir_tree()` or non-R files in the directory
`recurse`	If `TRUE` recurse fully, if a positive number the number of levels to recurse.

Details

`proj_outline()` and `dir_outline()` are wrapper of `file_outline()`.

In `proj_outline()`, `path` accepts project names, see `proj_list()` for how to set up reuseme to recognize your projects' locations.

The parser is very opinionated and is not very robust as it is based on regexps. For a better file parser, explore other options, like lightparser for Quarto, `{roxygen2}`

Will show TODO items and will offer a link to mark them as complete.

Note that `proj_outline()` strips some test files from the outline, as example test files (like in usethis repo) don't help understand a project's outline. Use `dir_outline(recurse = TRUE)` to make sure these are included in your outline.

Value

A `outline_report` object that contains the information. Inherits `tbl_df`.

A symbol will show for recently modified files.

Examples

```
file <- fs::path_package("reuseme", "example-file", "outline-script.R")
file_outline(file)

# Remove todo items
file_outline(file, print_todo = FALSE, alpha = TRUE)

# interact with data frame
file_outline(file) |> dplyr::as_tibble()


# These all work on the active file / project or directory.

file_outline()
proj_outline()
dir_outline()
# Like proj_switch(), proj_outline() accepts a project
```

`proj-reuseme` *Interact with different RStudio projects*

Description

The package offers many ways to interact with different local RStudio projects.

Setup

To take advantage of this functionality, you first have to set `options(reuseme.reposdir)` in your .Rprofile file. Access it with `usethis::edit_r_profile()`.

I would recommend you add the following. It works better if you store your RStudio projects in common directories.

Inspired by usethis options

```
if (interactive()) {
  options(reuseme.reposdir = c("~/rrr/", "~/packages", "~/rrr-work/"))

}
```

Capabilities.

Assumes that you have a project named `"learning"` A project outline

```
proj_outline("learning)
```

Add a TODO item to the `learning` project

```
use_todo("learning::Learn this")
```

Get file outline of the `file.R` in "learning"

```
proj_file("file", "learning")
```

Move to a new project in the same session

```
proj_switch("learning")
```

A lot of these features are already present in RStudio and with usethis. However, when managing many projects, the recent projects list can be more difficult to handle. Passing the full project name to `usethis::proj_activate()` was too long.

See Also

Other project management helpers: `proj_file()`, `proj_list()`, `proj_switch()`

---

proj_file — *Access the file outline within other project*

Description

It can be used as `file_outline()` + `proj`.

Usage

```
proj_file(file = NULL, path = active_rs_proj(), pattern = NULL)
```

Arguments

`file`	A filename or regexp to a file inside `proj`
`path`	a project path `proj_list()`. If `NULL`, will return active project.
`pattern`	A regular expression to look for

Value

The file outline if multiple matches are found

See Also

Other project management helpers: `proj-reuseme`, `proj_list()`, `proj_switch()`

Examples

```
try(proj_file("A non-existent file"))
```

proj_list — *Specify* `proj` *in functions*

Description

Two main ways to specify proj:

- Set `options(reuseme.reposdir)` in `.Rprofile` that contains a character vector of paths to your projects. (i.e. `~/rrr` that contains all your projects)
- Specify the full path to `proj`. (like you would for usethis function)

Usage

```
proj_list(proj = NA, dirs = getOption("reuseme.reposdir"))
```

Arguments

`proj`	A project path or name to match. If `NA`, returns all projects. If `NULL`, returns the active project.
`dirs`	The directories in which we want to list projects.

Value

A named character vector with the project name as name, and path as value. If `proj` is supplied

See Also

Other project management helpers: `proj-reuseme`, `proj_file()`, `proj_switch()`

proj_switch — *Opens a RStudio project in a new session*

Description

If not specified, will generate hyperlinks that call `usethis::proj_activate()`. `proj_switch()` looks at `options(reuseme.reposdir)`.

Usage

```
proj_switch(proj = NA, new_session = TRUE)
```

Arguments

`proj`	the name of a project located in the default locations or `NA`
`new_session`	Should open in a new session?

Value

Single logical value indicating if current session is modified.

See Also

usethis::proj_activate()

Other project management helpers: proj-reuseme, proj_file(), proj_list()

---

quarto_help *Show links to Quarto documentation of interest*

Description

Very opinionated of links I need to access periodically. Easily accessible from R console.

Usage

```
quarto_help(subject = NULL)
```

Arguments

subject	A character vector (optional)

Value

All possible links for help invisibly

Examples

```
gt_help <- quarto_help() |>
  tibble::enframe() |>
  gt::gt() |>
  gt::fmt_url(value)


gt_help
```

`rename_files2` *Rename an output or a data file and watch for references*

Description

[Experimental]

This function can improve your workflow. It is inspired by `usethis::rename_files()`, but its scope is more oriented towards analysis script.

Usage

```
rename_files2(
  old,
  new,
  warn_conflicts = c("default", "all", "exact", "none"),
  overwrite = FALSE,
  action = c("rename", "test"),
  force = deprecated()
)
```

Arguments

`old, new`	Old and new file names (with or without `.R` extensions).
`warn_conflicts`	One of • `"default"`: will be check more thoroughly depending on the situation. If only moving directory, and `"all"` otherwise. • `"all"` (larger scope: if `old = "data/my-streets.csv|my_streets"` will check for objects named `my_streets`, other files like `my-streets.R`, etc.), • `"exact"` will only search for `"data/my-streets.csv"` in documents `"none"` will not search for references in documents and will rename.
`overwrite`	whether to overwrite `new` if it already exists. Be careful.
`action`	One of `"rename"` or `"test"`
`force`	**[Deprecated]** Use `warn_conflicts` instead of `force = TRUE`

Value

`new` if renaming succeeded. Mostly called for its side-effects

Use case

Let's say you have an analysis and work on a certain subject. You want to rename a figure for clarity. For example, you had an input file named `data/my-streets.csv` and you now want to rename it to

Here is what `rename_files2()` does for you, before it renames files.

1. Look for potential name conflict
2. Look for data frame name conflicts
3. Sends information to clipboard

Will work well for you if you tend to name your objects using snake case and naming objects with snake case or kebab-case.

The philosophy is to inform you of manual steps required before actually performing file renaming.

A way to be less strict is to us

`screenshot` *Save the current image in clipboard to png in your active directory*

Description

The screenshot will be saved as `.png` to a directory following these rules

1. In a regular RStudio project (or a Quarto book), it will be saved to a `images/` directory
2. In a package project, it will be saved in a `man/figures` directory
3. In a Quarto Blog project, it will save in the current post's folder.
4. You can always override these defaults by setting `dir`

After using the shortcut Win + Shift + S, you can call this function!

Usage

```
screenshot(file = NULL, proj = proj_get(), dir = NULL)
```

Arguments

`file`	A file name, ideally `-` (kebab-case). (extension ignored) (optional, default is `image.png`)
`proj`	A project name
`dir`	A directory (optional), to override the directory rules mentioned in the description. inside `proj`.

Details

If no file name is supplied, a file named `image0*.png` will be created. The function then prompts you to rename the file with a more expressive name. It will continue the numbering if a file named image exists.

Still have to validate if it works on macOS, as it is not clear whether the image goes to the clipboard by default

The maximum number of images in a folder is 99. (only padding 2), should be enough.

You should not be able to overwrite a screenshot with a generic name, only a named one as it is possible you may require to retake your screenshot.

Value

The full image path, invisibly.

Examples

```
if (FALSE) {
  # Add an image to the clipboard
  # Run the following
  screenshot(file = "my-new-image")
}
```

`slice_group_sample` *Explore all rows in a random group*

Description

Compared to `slice_sample()` `slice_group_sample` will return all rows corresponding to a group.

Usage

```
slice_group_sample(data, group_var = NULL, n_groups = 1)
```

Arguments

`data`	A `data.frame`
`group_var`	if the data is grouped, will be ignored.
`n_groups`	Number of groups to sample. (passed to `sample(size = n_groups)`)

Value

A data frame with a sample group.

Examples

```
set.seed(10)
slice_group_sample(mtcars, group_var = vs)
mtcars |>
  dplyr::group_by(vs) |>
  slice_group_sample()
```

slice_min_max — *Subset rows using their positions*

Description

A wrapper around `dplyr::bind_rows()`, `dplyr::slice_min()`, `dplyr::slice_max()`

Usage

```
slice_min_max(
  .data,
  order_by,
  ...,
  n,
  prop,
  by = NULL,
  with_ties = TRUE,
  na_rm = FALSE,
  each = TRUE,
  ascending = TRUE
)
```

Arguments

`.data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.
`order_by`	<`data-masking`> Variable or function of variables to order by. To order by multiple variables, wrap them in a data frame or tibble.
`...`	Arguments are passed on to methods.
`n, prop`	Provide either `n`, the number of rows, or `prop`, the proportion of rows to select. If neither are supplied, `n = 1` will be used. If `n` is greater than the number of rows in the group (or `prop > 1`), the result will be silently truncated to the group size. `prop` will be rounded towards zero to generate an integer number of rows. A negative value of `n` or `prop` will be subtracted from the group size. For example, `n = -2` with a group of 5 rows will select 5 - 2 = 3 rows; `prop = -0.25` with 8 rows will select 8 * (1 - 0.25) = 6 rows.
`by`	**[Experimental]** <`tidy-select`> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see ?dplyr_by.
`with_ties`	Should ties be kept together? The default, `TRUE`, may return more rows than you request. Use `FALSE` to ignore ties, and return the first `n` rows.
`na_rm`	Should missing values in `order_by` be removed from the result? If `FALSE`, `NA` values are sorted to the end (like in `arrange()`), so they will only be included if there are insufficient non-missing values to reach `n`/`prop`.

`each`	If `FALSE`, `n` and `prop` passed to `dplyr::slice_min()` and `dplyr::slice_max()` will be divided by 2. (will use `ceiling()` if n is)
`ascending`	Return the output in ascending order. (min on top)

Value

An object of the same type as `.data`. The output has the following properties:

- Each row may appear 0, 1, or many times in the output.
- A `minmax` column is added to show which is min, which is max.
- Groups are not modified.
- Data frame attributes are preserved.

See Also

Other dplyr extensions: `count_pct()`

Examples

```
# in the presence of ties.
mtcars |> dplyr::slice_min(cyl, n = 1)
# Use with_ties = FALSE to return exactly n matches
mtcars |> dplyr::slice_min(cyl, n = 1, with_ties = FALSE)
# Use each = FALSE to have n divided in each place
mtcars |> slice_min_max(cyl, n = 2)
# Using each = TRUE (to retun n = 2, for min, n = 2 for max)
mtcars |> slice_min_max(cyl, each = TRUE, n = 2)
```

`summarise_with_total` *Compute a summary for groups with the total included.*

Description

This function is useful to create end tables, apply the same formula to a group and to its overall. You can specify a personalized `Total` value with the `.label` argument. You You should only use the output from `summarise_with_total()` with `tidyr::pivot_wider()`, write data to a spreadsheet, `gt::gt()` after that. Don't try to do more computing afterwards. It can also be used for plotting Changes the `.by` variable to a factor.

Usage

```
summarise_with_total(.data, ..., .by = NULL, .label = "Total", .first = TRUE)
```

Arguments

`.data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.
`...`	<`data-masking`> Name-value pairs of summary functions. The name will be the name of the variable in the result. The value can be: • A vector of length 1, e.g. `min(x)`, `n()`, or `sum(is.na(y))`. • A data frame, to add multiple columns from a single expression. **[Deprecated]** Returning values with size 0 or >1 was deprecated as of 1.1.0. Please use `reframe()` for this instead.
`.by`	**[Experimental]** <`tidy-select`> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see ?dplyr_by.
`.label`	Label of the total value
`.first`	Should the total be on top

Value

An ungrouped data frame with the total included in the first or last row.

Examples

```
# works with `.by`

mtcars |>
  summarise_with_total(
    x = mean(mpg),
    .by = vs,
    .label = "All vs"
  )

# works with `group_by()`
mtcars |>
  dplyr::group_by(vs) |>
  summarise_with_total(
    x = mean(mpg),
    .label = "All vs"
  )
```

`use_todo` *Add a TODO list by project to a TODO.R file in the base directory*

Description

Creates or edits a `TODO.R` file to store your TODOs. By default it will write in the current RStudio project.

Usage

```
use_todo(todo, proj = proj_get2(), code = FALSE)
```

Arguments

`todo`	A character vector of lines to add to the TODO file. See details for special handling.
`proj`	By default, the active project, an arbitrary directory, or a RStudio project name in the following directories `options(reuseme.destdir)`, uses `proj_list()` in this case. If a file, will write there.
`code`	If `TRUE`, will render code output (default is text).

Details

If you use `use_todo()` with a version-control repository, you may want to use `usethis::use_git_ignore("TODO.R")` if you don't want your `TODO.R` file

to be included in git. If using in a package directory, use `usethis::use_build_ignore("TODO.R")` to prevent a note in `R CMD CHECK`.

If you want to write to a global TODO, use

`options(reuseme.global_todo = fs::path("Documents"))` to write there.

Value

A `TODO.R` file appended with the `todo` string.

See Also

`usethis::write_union()`

Examples

```
if (FALSE) {
  use_todo("I need to do that")
  use_todo(c("I need to do that again", "youppi"))
  use_todo("c(x, y)", code = TRUE)
  use_todo("Here", proj = "my-analysis")
  use_todo(c("my-analysis::Here", "I am"))
  # add to a global todo.
  use_todo(c("all::Here", "I am"))
}
```

Index