Package 'org'

November 24, 2025

Title Organising Projects

Version 2025.11.24

Description A framework for organizing R projects with a standardized structure.
Most analyses consist of three main components: code, results, and data,
each with different requirements such as version control, sharing, and encryption.
This package provides tools to set up and manage project directories, handle
file paths consistently across operating systems, organize results using
date-based structures, source code from specified directories, create and
manage Quarto documents, and perform file operations safely.
It ensures consistency across projects while accommodating different
requirements for various types of content.

Depends R (>= 3.3.0)

License MIT + file LICENSE

URL https://www.rwhite.no/org/, https://github.com/raubreywhite/org

BugReports https://github.com/raubreywhite/org/issues

Encoding UTF-8

Imports utils

Suggests testthat, knitr, rmarkdown, rstudioapi, glue

RoxygenNote 7.3.3

VignetteBuilder knitr

Date/Publication 2025-11-24 09:30:11 UTC

NeedsCompilation no

Author Richard Aubrey White [aut, cre] (ORCID:
<https://orcid.org/0000-0002-6747-1726>)

Maintainer Richard Aubrey White <hello@rwhite.no>

Repository CRAN

Contents

`create_project_quarto_external_results`
Create a Quarto project with external results generation

Description

This function creates a new project structure that uses Quarto for documentation with results generated outside the Quarto document. This approach separates data processing from documentation, making it easier to manage large analyses.

Usage

```
create_project_quarto_external_results(home, results)
```

Arguments

`home`	Location of the main project directory
`results`	Location of the results directory

Details

The function creates:

- Basic project structure with R and Quarto directories
- .gitignore files for R and Quarto
- Run.R script for project initialization, data processing, and document rendering
- Quarto configuration (_quarto.yml)
- Example Quarto document (quarto.qmd)
- RStudio project file

Value

Nothing. Creates project structure and files.

Examples

```
## Not run:
# Create a new project
org::create_project_quarto_external_results(
  home = paste0(tempdir(), "/my_project"),
  results = paste0(tempdir(), "/results")
)

## End(Not run)
```

`create_project_quarto_internal_results` *Create a Quarto project with internal results generation*

Description

This function creates a new project structure that uses Quarto for documentation with results generated from within the Quarto document. The project includes:

- R code directory
- Quarto document setup
- Results directory with date-based organization
- Git configuration
- Project initialization code

Usage

```
create_project_quarto_internal_results(home, results)
```

Arguments

`home`	Location of the main project directory
`results`	Location of the results directory

Details

The function creates:

- Basic project structure with R and Quarto directories
- .gitignore files for R and Quarto
- Run.R script for project initialization and document rendering
- Quarto configuration (_quarto.yml)
- Example Quarto document (quarto.qmd)
- RStudio project file

Value

Nothing. Creates project structure and files.

Examples

```
## Not run:
# Create a new project
org::create_project_quarto_internal_results(
  home = paste0(tempdir(), "/my_project"),
  results = paste0(tempdir(), "/results")
)

## End(Not run)
```

`initialize_project` *Initialize project environment and structure*

Description

This function initializes a new R project by setting up folder locations and sourcing code files. It creates a standardized project structure with separate locations for code, results, and data. Results are automatically organized by date, and code can be sourced from specified directories.

Usage

```
initialize_project(
  env = new.env(),
  home = NULL,
  results = NULL,
  folders_to_be_sourced = "R",
  source_folders_absolute = FALSE,
  encode_from = "UTF-8",
  encode_to = "latin1",
  ...
)
```

Arguments

`env`	The environment that the code will be sourced into. Use `.GlobalEnv` to source code into the global environment. If a different environment is provided, all functions will be sourced into that environment.
`home`	The folder containing 'Run.R' and 'R/'. This is the main project directory.
`results`	The base folder for storing results. A subfolder with today's date will be created and accessible via `org::project$results_today`.
`folders_to_be_sourced`	Character vector of folder names inside `home` containing .R files to be sourced into the environment.

source_folders_absolute
: If TRUE, folders_to_be_sourced is treated as absolute paths. If FALSE, paths are relative to home.

encode_from
: Source encoding for file paths (only used on Windows)

encode_to
: Target encoding for file paths (only used on Windows)

...
: Additional named arguments for other project folders (e.g., data, raw, etc.)

Details

The function performs several key operations:

1. Creates necessary directories if they don't exist
2. Sets up date-based results organization
3. Sources all .R files from specified directories
4. Handles path encoding for cross-platform compatibility
5. Maintains a mirror of settings in org::project

Value

An environment containing:

- All folder locations as named elements
- $env: The environment where code was sourced
- $results_today: Path to today's results folder

Examples

```
## Not run:
# Initialize a new project
org::initialize_project(
  home = paste0(tempdir(), "/git/analyses/2019/analysis3/"),
  results = paste0(tempdir(), "/dropbox/analyses_results/2019/analysis3/"),
  raw = paste0(tempdir(), "/data/analyses/2019/analysis3/")
)

# Access project settings
org::project$results_today  # Today's results folder
org::project$raw           # Raw data folder

## End(Not run)
```

`ls_files` *List files and directories recursively*

Description

This function is equivalent to the Unix `ls` command but works across platforms. It can list files and directories matching a regular expression pattern.

Usage

```
ls_files(path = ".", regexp = NULL)
```

Arguments

`path`	A character vector of one or more paths to search
`regexp`	A regular expression pattern to filter files/directories

Details

The function:

- Handles both single and multiple paths
- Supports regular expression filtering
- Removes system-specific directories (e.g., @eaDir)
- Returns full paths

Value

A character vector of file and directory paths

Examples

```
# List all files in current directory
org::ls_files()

# List only R files
org::ls_files(regexp = "\\.R$")

# List files in multiple directories
org::ls_files(c("dir1", "dir2"))
```

`move_directory` *Move a directory and its contents*

Description

Moves a directory and all its contents to a new location. Can optionally overwrite the destination if it already exists.

Usage

```
move_directory(from, to, overwrite_to = FALSE)
```

Arguments

`from`	Source directory path.
`to`	Destination directory path.
`overwrite_to`	Whether to overwrite existing destination (default: FALSE).

Details

The function:

- Creates the destination directory if it doesn't exist
- Copies all files and subdirectories recursively
- Removes the source directory after successful copy
- Fails if source doesn't exist or destination exists (unless overwrite_to=TRUE)

Value

Nothing. Creates the destination directory and moves all contents.

Examples

```
## Not run:
# Move a directory
org::move_directory("old_dir", "new_dir")

# Move and overwrite existing directory
org::move_directory("old_dir", "new_dir", overwrite_to = TRUE)

## End(Not run)
```

package_installed *Check if a package is installed and optionally install it*

Description

This function checks whether a specified package is installed in the current R environment. Optionally, it can install the package if it is not already installed.

Usage

```
package_installed(pkg, install_if_missing = FALSE)
```

Arguments

pkg	A character string specifying the name of the package to check.
install_if_missing	A logical value indicating whether to install the package if it is not installed. Default is FALSE.

Value

A logical value: TRUE if the package is installed (or successfully installed), FALSE otherwise.

Examples

```
## Not run:
org::package_installed("data.table")
org::package_installed("ggplot2", install_if_missing = TRUE)

## End(Not run)
```

path *Construct file path from components*

Description

Joins path components using forward slashes, ensuring proper path formatting across operating systems. Handles multiple components and removes any double slashes that might occur.

Usage

```
path(...)
```

Arguments

...	Character vectors that will be concatenated with "/" as separator.

Value

A character vector containing the constructed path.

Examples

```
org::path("home", "user", "data.csv")  # Returns "home/user/data.csv"
org::path("home//user", "data.csv")    # Returns "home/user/data.csv"
```

---

project *Project folder locations*

Description

An environment that stores the locations of folders used in the project.

Usage

```
project
```

Format

An environment containing the following elements:

home The folder containing 'Run.R' and 'R/'

results_today The folder inside `results` with today's date, created by `initialize_project`

---

set_results *Set results folder after project initialization*

Description

Sets the results folder in the project environment and creates a date-based subfolder. The date-based folder is accessible via `proj$results_today` and empty date folders are automatically cleaned up when new results are added.

Usage

```
set_results(results, proj = org::project)
```

Arguments

`results`	A character vector specifying one or more possible results folder paths. The first existing path will be used.
`proj`	The project environment. Default is `org::project`.

Value

Nothing. Modifies the `proj` environment to include:

$results The base results folder path

$results_today Path to today's results folder (format: YYYY-MM-DD)

`write_text` *Write text to file*

Description

Writes text to a file, optionally including a header at the top of the file. Text and header are converted from Linux newline format to Windows newline format before writing.

Usage

```
write_text(
  txt,
  file = "",
 header = "**THIS FILE IS CONSTANTLY OVERWRITTEN -- DO NOT MANUALLY EDIT**\r\n\r\n"
)
```

Arguments

`txt`	A character string of text to be written to the file.
`file`	A character string specifying the file path. Passed through to `base::cat`. Default is an empty string, which writes to the console.
`header`	An optional character string header to be inserted at the top of the text file. Default is `**THIS FILE IS CONSTANTLY OVERWRITTEN -- DO NOT MANUALLY EDIT**\r\n\r\n`.

Value

No return value. Called for its side effect of writing to a file.

Examples

```
## Not run:
org::write_text("Sample text", "output.txt")
org::write_text("Another piece of text", "output.txt", "Custom Header\r\n\r\n")

## End(Not run)
```

Index