

Package: org (via r-universe)

September 26, 2026

Title Organising Projects

Version 2026.8.20

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, 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

Config/Needs/website raubreywhite/rwtemplate

Imports utils

Suggests testthat, knitr, rmarkdown, glue, pak

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

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

Date/Publication 2026-08-26 18:44:36 UTC

RemoteUrl <https://github.com/raubreywhite/org>

RemoteRef HEAD

RemoteSha a11b2eed498d1e11b782c7021f7a4adfa333d8be

Contents

initialize_project	2
loc_per_file	4
ls_files	5
move_directory	6
path	7
project	8
set_results	9
write_text	10
Index	12

initialize_project	<i>Initialize project environment and structure</i>
--------------------	---

Description

initialize_project() initializes a new R project. It sets up folder locations and sources code files. It creates a standardized project structure with separate locations for code, results, and data. It organizes results by date automatically, and it can source code from the directories you specify.

Usage

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

Arguments

- | | |
|-----------------------|--|
| env | The environment that initialize_project() sources the code into. Use .GlobalEnv to source code into the global environment. If you provide a different environment, all functions are 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. initialize_project() creates a subfolder with today's date. You reach it at org::project\$results_today. |
| folders_to_be_sourced | Character vector of folder names inside home. These folders hold the .R files to source into the environment. |

max_loc_per_file	The maximum number of code lines a single .R file in <code>folders_to_be_sourced</code> may hold. <code>initialize_project()</code> stops with an error naming every file above the limit, before it sources any of them. A code line is a physical line that is neither blank nor entirely a comment, as counted by <code>loc_per_file()</code> . Default is Inf, which checks nothing.
install_missing_packages	If TRUE, <code>initialize_project()</code> scans <code>folders_to_be_sourced</code> for package dependencies. It looks for <code>library()</code> , <code>require()</code> , and <code>pkg::</code> usage. It then installs any missing package with <code>pak</code> before it sources the code. If <code>pak</code> is not available, it falls back to <code>install.packages()</code> . Default is FALSE.
source_folders_absolute	If TRUE, <code>initialize_project()</code> treats <code>folders_to_be_sourced</code> as absolute paths. If FALSE, the 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., <code>data</code> , <code>raw</code> , etc.).

Details

`initialize_project()` performs the seven operations below, in this order.

1. Creates necessary directories if they do not exist.
2. Sets up date-based results organization.
3. Handles path encoding for cross-platform compatibility.
4. Stops if any file holds more code lines than `max_loc_per_file`.
5. Stops if a package the code needs is missing, or installs it when `install_missing_packages` is TRUE.
6. Sources all .R files from the specified directories.
7. Maintains a mirror of settings in `org::project`.

Value

An environment containing:

- All folder locations as named elements.
- `$env`: The environment that the code was sourced into.
- `$results_today`: Path to today's results folder.

See Also

`vignette("org")` for the recommended project layout. Its "Team collaboration" section shows how to give one folder several possible paths.

Other project setup: `project`, `set_results()`

Examples

```
# A minimal project: a home folder holding an R/ folder of functions
home <- file.path(tempdir(), "org_init_example", "analysis3")
dir.create(file.path(home, "R"), recursive = TRUE, showWarnings = FALSE)
writeLines("greet <- function() 'hello'", file.path(home, "R", "greet.R"))

proj <- org::initialize_project(
  home = home,
  results = file.path(tempdir(), "org_init_example", "results"),
  raw = file.path(tempdir(), "org_init_example", "raw")
)

# Folder locations, both on the returned environment and on org::project
proj$results_today # Today's results folder
proj$raw # Raw data folder
org::project$results_today

# Everything in home/R/ has been sourced into `env`
proj$env$greet()

unlink(file.path(tempdir(), "org_init_example"), recursive = TRUE)
```

loc_per_file

Count the code lines in R files

Description

loc_per_file() counts the code lines in each file. A code line is a physical line that is neither blank nor entirely a comment. The R parser identifies the comments, so a # inside a string does not hide a line, and a line inside a multi-line string still counts.

Usage

```
loc_per_file(paths)
```

Arguments

paths A character vector of paths to R files.

Details

Three rules decide whether a line counts:

- A blank line does not count.
- A line holding only a comment does not count. Roxygen lines are comments.
- Every other line counts, including a line inside a multi-line string, and a line of code that ends with a comment.

A regular expression alone cannot apply the second rule, because it cannot tell a comment from a # inside a string. loc_per_file() reads the comment positions from the parser instead, and it errors on a file that R cannot parse.

Value

A named integer vector, one element per path. Each name is the path, and each value is the number of code lines in that file.

See Also

`initialize_project()`, whose `max_loc_per_file` argument uses this function to stop a project that holds a file above the limit.

Other file utilities: `ls_files()`, `move_directory()`, `path()`, `write_text()`

Examples

```
f <- file.path(tempdir(), "org_loc_example.R")
writeLines(c(
  "# a comment, which does not count",
  "",
  "x <- 1 # a trailing comment, so the line counts",
  "y <- 2"
), f)

org::loc_per_file(f) # 2

unlink(f)
```

ls_files

List the files and directories directly inside a folder

Description

`ls_files()` is equivalent to the Unix `ls` command, and it works across platforms. It can list files and directories that match a regular expression pattern.

Usage

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

Arguments

<code>path</code>	A character vector of one or more paths to search.
<code>regexp</code>	A regular expression pattern to filter files and directories.

Details

`ls_files()`:

- 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.

See Also

`vignette("org")`, whose "Troubleshooting" section uses this function to check which files a project will source.

Other file utilities: `loc_per_file()`, `move_directory()`, `path()`, `write_text()`

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	<i>Move a directory and its contents</i>
----------------	--

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

`move_directory()`:

- Creates the destination directory if it does not exist.
- Copies all files and subdirectories recursively.
- Removes the source directory after a successful copy.
- Fails if the source does not exist, or if the destination exists and `overwrite_to` is FALSE.

Value

Nothing. Creates the destination directory and moves all contents.

See Also

vignette("org"), whose "Function reference" section lists this alongside the other file operations.

Other file utilities: [loc_per_file\(\)](#), [ls_files\(\)](#), [path\(\)](#), [write_text\(\)](#)

Examples

```
from <- file.path(tempdir(), "org_move_from")
to <- file.path(tempdir(), "org_move_to")
dir.create(from, showWarnings = FALSE)
writeLines("first", file.path(from, "a.txt"))

# Move a directory
org::move_directory(from, to)
dir.exists(from) # FALSE, the source is gone
list.files(to) # "a.txt"

# Move and overwrite existing directory. The destination is replaced,
# not merged, so "a.txt" does not survive.
dir.create(from, showWarnings = FALSE)
writeLines("second", file.path(from, "b.txt"))
org::move_directory(from, to, overwrite_to = TRUE)
list.files(to) # "b.txt"

unlink(to, recursive = TRUE)
```

path

Construct file path from components

Description

Joins path components with forward slashes, so the path format is the same across operating systems. Handles multiple components and removes any double slashes that might occur.

Usage

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

Arguments

... Character vectors to join with "/" as the separator. `path()` recycles them against each other, the same way `base::file.path()` does, so one component of length 3 and one of length 1 give three paths.

Value

A character vector containing the constructed path. It is `character(0)` if any component has length zero.

See Also

`vignette("org")`, whose "Path construction and cross-platform compatibility" section shows how this function is used in an analysis.

Other file utilities: `loc_per_file()`, `ls_files()`, `move_directory()`, `write_text()`

Examples

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

# Components are recycled, so one call builds several paths
org::path("home", c("a.csv", "b.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. `initialize_project()` creates it.

See Also

`vignette("org")`, whose "Project structure" section describes the standard folders and the additional-folder convention. `org` stores folders passed through ... here under their own names. This help page does not describe them individually.

Other project setup: `initialize_project()`, `set_results()`

Examples

```
home <- file.path(tempdir(), "org_project_example")
dir.create(file.path(home, "R"), recursive = TRUE, showWarnings = FALSE)

proj <- org::initialize_project(
  home = home,
  results = file.path(home, "results"),
  data_raw = file.path(home, "data_raw")
)

# Every folder is now available from anywhere in the analysis
org::project$home
org::project$results
org::project$results_today
org::project$data_raw

unlink(home, recursive = TRUE)
```

set_results	<i>Set results folder after project initialization</i>
-------------	--

Description

Sets the results folder in the project environment. Creates a date-based subfolder. You reach the date-based folder at `proj$results_today`. `set_results()` automatically cleans up empty date folders 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. <code>set_results()</code> uses the first existing path.
proj	The project environment. Default is <code>org::project</code> .

Value

Nothing. Modifies the `proj` environment to include:

\$results The base results folder path.

\$results_today Path to today's results folder. The format is YYYY-MM-DD.

See Also

`vignette("org")`, whose "Common workflows" section shows when to change the results folder after a project is already initialized.

Other project setup: [initialize_project\(\)](#), [project](#)

Examples

```
home <- file.path(tempdir(), "org_set_results_example")
dir.create(file.path(home, "R"), recursive = TRUE, showWarnings = FALSE)

proj <- org::initialize_project(
  home = home,
  results = file.path(home, "results_a")
)
org::project$results_today

# Point the project at a different results folder
org::set_results(file.path(home, "results_b"))
org::project$results_today
dir.exists(org::project$results_today)

unlink(home, recursive = TRUE)
```

write_text

Write text to file

Description

Writes text to a file. It can also insert a header at the top of the file. `write_text()` converts the text and the header from Linux newline format to Windows newline format before it writes.

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 <code>base::cat</code> . 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 <code>**THIS FILE IS CONSTANTLY OVERWRITTEN -- DO NOT MANUALLY EDIT**\r\n\r\n</code> .

Value

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

See Also

`vignette("org")`, whose "Example project structure" section uses this function to keep a change log beside the results.

Other file utilities: [loc_per_file\(\)](#), [ls_files\(\)](#), [move_directory\(\)](#), [path\(\)](#)

Examples

```
f <- file.path(tempdir(), "output.txt")

org::write_text("Sample text", f)
readLines(f, warn = FALSE)

# A custom header replaces the default one. The file is overwritten.
org::write_text("Another piece of text", f, "Custom Header\r\n\r\n")
readLines(f, warn = FALSE)

unlink(f)
```

Index

* file utilities

- loc_per_file, 4
- ls_files, 5
- move_directory, 6
- path, 7
- write_text, 10

* project setup

- initialize_project, 2
- project, 8
- set_results, 9

base::file.path(), 7

initialize_project, 2
initialize_project(), 5, 8, 9

loc_per_file, 4
loc_per_file(), 3, 6–8, 11
ls_files, 5
ls_files(), 5, 7, 8, 11

move_directory, 6
move_directory(), 5, 6, 8, 11

path, 7
path(), 5–7, 11
project, 3, 8, 9

set_results, 9
set_results(), 3, 8

write_text, 10
write_text(), 5–8