Package 'rpkgkit'

July 21, 2026

Title Create and Maintain R Packages

Version 0.1.7

Description Utilities for R package development including NEWS.md management, standalone file creation, and code formatting. Supports popular development workflows and integrates with 'usethis' and 'RStudio'. Includes helper functions for renaming functions and detecting common coding errors.

License MIT + file LICENSE

URL https://github.com/WangLabCSU/rpkgkit

BugReports https://github.com/WangLabCSU/rpkgkit/issues

Depends R (>= 4.1.0)

Imports cli, lifecycle, rlang (>= 1.0.0), rstudioapi

Suggests desc, devtools, dplyr, flir, gh, grDevices, httr2, pkgload, rmarkdown, stats, testthat (>= 3.0.0), usethis, withr, yaml

Config/testthat/edition 3

Config/usethis/last-upkeep 2026-06-29

Copyright Jacob Scott, Christopher T. Kenny, and Sebastian Lammers (for the pedant code included in R/vendor-pedant.R); Diego Hernangómez (for the pkgdev code included in R/vendor-pkgdev.R); and (for the pedant code included in R/vendor-pedant.R); Diego Hernangómez (for the pkgdev code included in R/vendor-pkgdev.R)

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Yuxi Yang [aut, cre],
Jacob Scott [aut, cph] (Author of the included pedant code (https://github.com/wurli/pedant)),
Christopher T. Kenny [ctb, cph] (Contributor to the included pedant code),
Sebastian Lammers [ctb, cph] (Contributor to the included pedant code),

Diego Hernangómez [aut, cph] (Author of the included pkgdev code
(https://github.com/dieghernan/pkgdev))

Maintainer Yuxi Yang <15364051195@163.com>

Repository CRAN

Date/Publication 2026-07-21 09:40:02 UTC

Contents

`add_changelog_in_standalone` *Add or update a changelog entry in a standalone R file*

Description

Appends a new changelog entry at the top of the `Changelog` section in a standalone file. Also updates the `last-updated` field in the YAML header to today's date. If no `Changelog` section exists, one is created after the YAML header.

Usage

```
add_changelog_in_standalone(path = NULL, description = NULL, date = NULL)
```

Arguments

path	Character. Path to a standalone file, a directory, or NULL. If NULL and rstudioapi is available, uses the currently active document. If a directory, searches for standalone files matching the pattern.
description	Character. Description of the changelog entry. If NULL, an interactive prompt is used (not yet implemented).
date	Character. Date string in YYYY-MM-DD format. Defaults to today.

Value

Invisibly returns the path to the updated file.

Examples

```
add_changelog_in_standalone(tempdir(), "Added new feature")
```

---

add_double_colons *Make function calls explicit*

Description

This function takes a block of code and seeks to make all function calls 'explicit' through the use of the double-colon operator `::`. This function is bound to the RStudio addin "Make function calls explicit". See examples for usage.

Usage

```
add_double_colons(
  code = NULL,
  use_packages = current_packages(),
  ignore_functions = imported_functions()
)
```

Arguments

code	Code to transform. Either a character vector or NULL, in which case any highlighted code (in RStudio) will be used.
use_packages	A character vector of package names. The order is important here - see examples for details.
ignore_functions	Functions to ignore when applying the transformation

Details

This function behaves differently depending on the context.

- **Not package development**: If the current context is not package development, then it will make function calls explicit using the currently attached packages (i.e. the ones attached by calls to `library()`).
- **Package development**: If it detects that the current context is package development it will make function calls explicit using packages in the 'Imports' field of the package `DESCRIPTION`. If the package being developed imports any packages in their entirety (i.e if `Import pkg` appears in the `NAMESPACE` file), calls to functions from these packages will be left unchanged. See `current_packages()` for more information.

Value

The transformed code as a character string

Examples

```
code <- "
  cars <- as_tibble(mtcars)
  cars %>%
    filter(mpg > 20) %>%
    summarise(across(everything(), n_distinct))
"

# Code will be transformed to use the double-colon operator, but notice
# that `n_distinct` is not transformed as it is not followed by `()`
cat(add_double_colons(code, "dplyr"))

# You can specify functions that shouldn't be transformed:
cat(add_double_colons(code, "dplyr", ignore_functions = "across"))

# Beware namespace conflicts! The following are not the same, mimicking
# the effects of reordering calls to `library()`:
cat(add_double_colons(code, c("dplyr", "stats")))

cat(add_double_colons(code, c("stats", "dplyr")))
```

---

add_global_gitgnore *Add* `.gitignore` *to package*

Description

Adds a global `.gitgnore` file to the package. File based on default `.gitignore` provided by GitHub.

Usage

```
add_global_gitgnore(pkg = NULL)
```

Arguments

pkg	The package to add the .gitignore file to. Defaults to the current working directory.

Value

Invisible, and writes a global .gitignore file

See Also

usethis::use_git_ignore(), usethis::git_vaccinate()

Examples

```
## Not run:
add_global_gitgnore()

## End(Not run)
```

---

add_global_rbuildignore

Add global .Rbuildignore patterns

Description

Appends a curated set of common files and directories to .Rbuildignore to exclude them from R package builds. This supplements the auto-generated .Rbuildignore created by usethis::create_package().

Already-present patterns are silently skipped. The curated set covers: R build artifacts, Git files, IDE/dev tool directories, CI/CD config, documentation build artifacts, CRAN/development files, and other common files that should not be shipped with a package.

Usage

```
add_global_rbuildignore(..., path = NULL)
```

Arguments

...	Additional regex patterns (character strings) to add beyond the curated defaults. Each must already be in .Rbuildignore regex format (e.g. "^\\.myfile$").
path	Character. Path to the package root directory. If NULL (the default), uses the current working directory.

Value

Invisibly returns the path to .Rbuildignore.

Examples

```
## Not run:
add_global_rbuildignore()

# With additional custom patterns
add_global_rbuildignore("^\\.myconfig$", "^data-raw$")

## End(Not run)
```

`air_format` *Format R code using air*

Description

Format R code using air

Usage

```
air_format(path = NULL, ...)
```

Arguments

`path`	Path to the R file to format. If NULL, attempts to use the active document in RStudio (requires `rstudioapi` package).
`...`	Additional arguments passed to `system2()`.

Details

Install air:

Linux: `curl -LsSf https://github.com/posit-dev/air/releases/latest/download/air-installer.sh | sh`
Windows: `powershell -ExecutionPolicy Bypass -c "irm https://github.com/posit-dev/air/releases/latest/`
uv: `uv tool install air-formatter` brew (MacOS): `brew install air`

Value

The exit status of the `air format` command (invisibly).

Examples

```
## Not run:
air_format(system.file("R_template/zzz_template.R", package = "rpkgkit"))

## End(Not run)
```

badge_translated_by_ai *Generate AI Translation Disclaimer Badge*

Description

Prints a shields.io badge and a blockquote note for each specified language that can be copied into a translated README file to indicate the content was AI-translated and has not been reviewed.

Each language entry consists of:

- A badge: `[![AI|<LANG>](https://img.shields.io/badge/AI-<LANG>-yellow)]()`
- A blockquote with the full disclaimer text in that language

Usage

```
badge_translated_by_ai(lang = "en", color = "yellow")
```

Arguments

`lang`	Character vector of language codes (e.g. `"zh-cn"`, `"ja"`). Defaults to `NULL`, which outputs disclaimers for **all** 19 supported languages. Pass a single code to get one entry only.
`color`	Color of badge. Defaults to `"yellow"`

Value

Invisibly returns a named list of character vectors, where each element contains the badge line and blockquote note for one language.

Examples

```
## Not run:
# All 19 languages
badge_translated_by_ai()

# Just one language
badge_translated_by_ai("de")

# A few languages
badge_translated_by_ai(c("ja", "ko"))

## End(Not run)
```

`browse_standalone` *Browse Standalone R Files Across GitHub*

Description

Search GitHub for all `standalone-*.R` files and filter to only include repositories that are R packages (contain a `DESCRIPTION` file in their root directory). Results are returned as a tibble with file and repository metadata.

Usage

```
browse_standalone(per_page = 100L, limit = 200L, ...)
```

Arguments

`per_page`	Number of items to return per page. If omitted, will be substituted by max(.limit, 100) if .limit is set, otherwise determined by the API (never greater than 100).
`limit`	Number of records to return. This can be used instead of manual pagination. By default it is NULL, which means that the defaults of the GitHub API are used. You can set it to a number to request more (or less) records, and also to Inf to request all records. Note, that if you request many records, then multiple GitHub API calls are used to get them, and this can take a potentially long time.
`...`	pass to `gh::gh()`

Details

This function uses the GitHub Code Search API (`GET /search/code`) to find all files matching the pattern `standalone-*.R` across all public repositories. It then checks each unique repository for the presence of a `DESCRIPTION` file in the repository root, confirming it is an R package. Only files from R package repositories are returned.

GitHub Search API returns at most 1,000 results, which covers the vast majority of standalone files on GitHub.

Value

A tibble with columns:

repo Character. Repository identifier in "owner/repo" format.

name Character. File name (e.g. "standalone-utils.R").

path Character. File path within the repository (e.g. "R/standalone-utils.R").

sha Character. File SHA.

url Character. GitHub API URL for the file.

html_url Character. GitHub HTML URL for the file.

git_url Character. Git blob URL for the file.

repo_url Character. Repository URL on GitHub.

repo_description Character. Repository description from GitHub.

Note

Requires GitHub API authentication via the gh package. Run `gh::gh_whoami()` to check your current authentication status. Unauthenticated requests are subject to strict rate limits (60 requests/hour).

Examples

```
## Not run:
  browse_standalone()

## End(Not run)
```

`check_pkgdown_reference` *Check that all exported functions are listed in pkgdown reference*

Description

Parses the `NAMESPACE` and `_pkgdown.yml` (if present) and reports any exported functions that are missing from the pkgdown reference index.

Usage

```
check_pkgdown_reference(pkg = ".")
```

Arguments

`pkg`	Character. Path to the package root directory. Defaults to the current RStudio project or `"."`.

Value

Invisibly returns a character vector of missing function names, or `NULL` if `_pkgdown.yml` does not exist. Prints a summary via `cli`.

Examples

```
## Not run:
dir <- tempdir()
usethis::create_package(dir)
usethis::proj_set(dir)
usethis::use_pkgdown()
check_pkgdown_reference(dir)

## End(Not run)
```

convert_func_syntax *Switch Between Explicit* function() *and Implicit* \() *Syntax*

Description

Converts all function definitions in an R file between the explicit function() syntax and the R 4.1+ concise lambda \() syntax. Handles strings and comments correctly, and supports nested function definitions via iterative passes.

Usage

```
convert_func_syntax(
  path = NULL,
  direction = c("to_lambda", "to_explicit"),
  ...
)
```

Arguments

path	A character string specifying the path to the R file to modify. If NULL and RStudio is available, the currently active document path is used.
direction	Conversion direction. One of: • "to_lambda": convert function(...) to \(...) • "to_explicit": convert \(...) to function(...)
...	Additional arguments. Currently unused and must be empty.

Details

The conversion correctly handles strings, comments, and nested parentheses in default argument values (e.g., function(x = foo(y))).

Nested function definitions (e.g., function(x, f = function(y) ...)) are converted in multiple iterative passes, so all nesting levels are reached.

Value

Invisibly returns the path to the modified file.

Examples

```
temp <- tempfile(fileext = ".R")
writeLines("add_one <- function(x) x + 1", temp)
convert_func_syntax(temp, direction = "to_lambda")
readLines(temp)
# "add_one <- \(x) x + 1"
convert_func_syntax(temp, direction = "to_explicit")
readLines(temp)
# "add_one <- function(x) x + 1"
```

`create_standalone` *Creates a new standalone R script with YAML metadata header. If* `path` *is an R package, the file is created in the* `R/` *subdirectory.*

Description

Creates a new standalone R script with YAML metadata header. If `path` is an R package, the file is created in the `R/` subdirectory.

Usage

```
create_standalone(
  standalone_name = NULL,
  path = NULL,
 standalone_head = list(license = "https://unlicense.org", imports = NULL, dependency =
    NULL, description = "This file provides..."),
  open = rlang::is_interactive(),
  ...
)
```

Arguments

`standalone_name`
: Character. The name suffix for the standalone file (e.g., "my_utils" creates "standalone-my_utils.R").

`path`
: Character. Directory path where to create the file. Defaults to the current working directory (`"."`).

`standalone_head`
: List. Metadata for the file header with elements:

- `license`: Character. License URL or identifier. Defaults to "https://unlicense.org".
- `imports`: Character vector. Package dependencies to import. Defaults to NULL.
- `dependency`: Character vector. Hard dependencies for the standalone file. Defaults to NULL.
- `description`: Character. A short description of the standalone file. Defaults to "To be filled.".

`open`
: Logical. Whether to open the file in RStudio editor. Defaults to TRUE.

`...`
: Additional arguments (must be empty).

Value

Invisibly returns the path to the created file.

Examples

```
create_standalone("my_utils", path = tempdir())
```

`flir_fix` *Fix R code or package using flir*

Description

Automatically detect the type of path (file, package, or directory) and apply the appropriate flir fix function.

Usage

```
flir_fix(path = NULL, ...)
```

Arguments

`path`	A file path, package directory path, or NULL. If NULL and running in RStudio, uses the active document path.
`...`	Additional arguments passed to the underlying flir fix function.

Details

The function determines the fix strategy based on the path type:

- If `path` points to an existing file, calls `flir::fix()`
- If `path` is a package directory (contains DESCRIPTION), calls `flir::fix_package()`
- If `path` is a directory, calls `flir::fix_dir()`

Value

Invisibly returns the result from the called flir function.

Examples

```
tmp <- tempfile(fileext = ".R")
writeLines("a<-1+1", tmp)
flir_fix(tmp)
cat(readLines(tmp, warn = FALSE), sep = "\n")
```

`inquire_standalone` *Inquire Standalone Files from a GitHub Repository*

Description

Retrieve information about all standalone R files in the R/ directory of a GitHub repository. Standalone files are identified by the "standalone-" prefix in their filename.

Usage

```
inquire_standalone(owner, repo, ...)
```

Arguments

`owner`	A character string specifying the repository owner's username.
`repo`	A character string specifying the repository name.
`...`	No arguments.

Details

This function queries the GitHub API to list files in the R/ directory, filters for files starting with "standalone-", parses each file's YAML metadata (delimited by "# —") and roxygen tags to extract descriptions, and generates usage code for importing each standalone file.

Value

A tibble with three columns:

`owner/repo` Character string, the repository identifier in "owner/repo" format.

description Character string, the description extracted from the file's YAML header or roxygen documentation.

usage Character string, the code to import the standalone file using `usethis::use_standalone()`.

Examples

```
## Not run:
inquire_standalone("r-lib", "rlang")

## End(Not run)
```

`is_dev_context` *Get packages from the current context*

Description

These functions find the packages/functions to use when running `add_double_colons()`.

Usage

```
is_dev_context(dir = ".")

imported_functions(dir = ".")

current_packages(
  dir = ".",
  base_packages = getOption("defaultPackages"),
  include_types = "Imports"
)
```

Arguments

`dir`	The current working directory
`base_packages`	Default packages to include
`include_types`	The types of package imports to return if the current context is package development. Should be a subset of `c("Imports", "Depends", "Suggests", "Enhances", "LinkingTo")`

Details

- `current_packages()` first checks if the current context is package development. If it is, then it returns the packages which are listed in the package `DESCRIPTION` as dependencies, but will not return any packages also listed as imports in the package `NAMESPACE`. If the current context is not package development, the currently attached packages (as given by `search()`) are used. Note that if `{pkgload}` is not installed then the latter option is always used.
- `imported_functions()` looks for a package `NAMESPACE` file and returns the names of all imported functions. If a `NAMESPACE` file is not found, or if `{pkgload}` is not loaded, `NULL` is returned.

Value

`TRUE` if the current context is package development, `FALSE` otherwise.

A character vector of imported function names, or `NULL` if no NAMESPACE file is found or `{pkgload}` is not installed.

A character vector of package names.

news_md *Manage NEWS.md file for R packages*

Description

Functions to help manage the NEWS.md file in R packages according to CRAN guidelines. Includes functions to add new entries and check the format.

Validates the NEWS.md file against CRAN guidelines and common best practices. Reports issues found and provides suggestions for fixes.

Reads and displays the NEWS.md file content with optional filtering.

Adds a new entry to the NEWS.md file following CRAN guidelines. Can create a new version section if needed or add to an existing one.

Usage

```
news_md_check(path = NULL, strict = FALSE, verbose = TRUE)

news_md_show(path = NULL, version = NULL, max_versions = NULL)

news_md_add_entry(
  entry,
  version = NULL,
  category = "NEW FEATURES",
  contributor = NULL,
  path = NULL,
  date = NULL,
  open_section = TRUE
)
```

Arguments

path	Path to the package root. If NULL (the default), uses the current working directory.
strict	If TRUE, treats warnings as errors. Default is FALSE.
verbose	If TRUE, prints detailed information about checks performed.
version	Package version number. If NULL, uses the version from DESCRIPTION.
max_versions	Maximum number of versions to display. NULL shows all.
entry	Text of the news entry (without leading "* ").
category	Category of the change (e.g., "NEW FEATURES", "BUG FIXES", "MINOR IMPROVEMENTS", "DOCUMENTATION", "DEPRECATED", "DEFUNCT", "BREAKING CHANGES", "PERFORMANCE", "TESTING", "INTERNAL CHANGES"). Default is "NEW FEATURES".
contributor	GitHub username or name for attribution (optional).
date	Date of the release. If NULL, uses today's date (YYYY-MM-DD format).

`open_section` If TRUE and the version section exists but isn't open (has content after it), creates a new section. If FALSE, adds to existing section.

Details

The NEWS.md format follows CRAN recommendations:

- Version headers use # followed by "package version (date)"
- Major changes use ## with category names (e.g., "NEW FEATURES", "BUG FIXES")
- Individual items use bullet points starting with `*`
- Contributor acknowledgments use `(@username)` at the end of items
- Dates should be in YYYY-MM-DD format

Value

A list with components:

valid Logical indicating if NEWS.md passes all checks.

errors Character vector of error messages.

warnings Character vector of warning messages.

suggestions Character vector of improvement suggestions.

Invisibly returns the content as a character vector.

Invisibly returns the path to the NEWS.md file.

Examples

```
## Not run:
temp <- tempdir()
usethis::create_package(temp)
file.create(file.path(temp, "NEWS.md"))
result <- news_md_check(temp)
if (!result$valid) {
  print(result$errors)
  print(result$warnings)
}

## End(Not run)
## Not run:
temp <- tempdir()
usethis::create_package(temp)
file.create(file.path(temp, "NEWS.md"))
news_md_add_entry("Added `foo()`", path = temp)

# Show latest version news
news_md_show(version = "latest", path = temp)

# Show last 3 versions
news_md_show(max_versions = 3, path = temp)
```

```
## End(Not run)
## Not run:
temp <- tempdir()
usethis::create_package(temp)
file.create(file.path(temp, "NEWS.md"))
# Add a bug fix entry
news_md_add_entry("Fixed issue with parsing large files.",
                  category = "BUG FIXES",
                  contributor = "johndoe",
                  path = temp)

# Add a new feature with specific version
news_md_add_entry("Added new function for data validation.",
                  version = "1.2.0",
                  category = "NEW FEATURES",
                  path = temp)

## End(Not run)
```

`package_func_arg_explicit` *Make Function Arguments Explicit*

Description

Transform function calls in R source code so that all arguments are passed with explicit parameter names. Uses a recursive tree-walking approach: every function call node is inspected, the called function's formals are retrieved, and positional arguments are given their formal parameter name.

Transformation preserves all content outside the expression boundaries (roxygen docs, section comments, blank lines). Inline comments on the last line of a transformed expression are re-attached to the output.

If the function has a `...` formal, unmatched positional and named arguments are left in place as-is (they are captured by `...`).

Operators (`+`, `-`, `*`, `/`, etc.), subset operators (`[`, `[[`, `$`), assignment (`<-`, `=`, `<<-`), and special syntax (`if`, `for`, `while`, `repeat`, `{`, `(`, `function`) are not transformed.

Usage

```
package_func_arg_explicit(path = NULL, skip_functions = NULL, ...)

make_func_arg_explicit(path = NULL, skip_functions = NULL, ...)
```

Arguments

`path`	Path to an R file to modify. If `NULL` and RStudio is available, the active document path is used.

`skip_functions` Optional character vector of function or operator names to skip during transformation (e.g. `c("my_special_fn")`). In addition to user-provided names, all built-in operators (`+`, `-`, etc.), special syntax forms (`if`, `for`, `{`, etc.), and all `%...%` infix operators (`%in%`, `%>%`, `%||%`, etc.) are always skipped automatically.

`...` Additional arguments. Currently unused and must be empty.

Value

Invisible `NULL`, called for its side effect of writing the transformed code back to the file.

Functions

- `package_func_arg_explicit()`: Processes all `.R` files in a package's `R/` directory, making function arguments explicit.

Single-file operation

Operates on one R file. When `path` is `NULL` and RStudio is available, the currently active document is used automatically.

Examples

```
tf <- tempfile(fileext = ".R")
writeLines("vapply(1:9, function(x) x*2, numeric(1))", tf)
make_func_arg_explicit(tf)
cat(readLines(tf), sep = "\n")
# vapply(X = 1:9, FUN = function(x) x*2, FUN.VALUE = numeric(1))
```

---

package_func_call_explicit

Make Function Calls Explicit

Description

Add double colons (`::`) to function calls from specified packages to make package dependencies explicit in R code.

This function uses code adapted from the pedant package by Jacob Scott et al., licensed under MIT.

Usage

```
package_func_call_explicit(
  path = NULL,
  use_packages = current_packages(),
  ignore_functions = imported_functions(),
  ...
)
```

```
make_func_call_explicit(
  path = NULL,
  use_packages = current_packages(),
  ignore_functions = imported_functions(),
  ...
)
```

Arguments

path	A character string specifying the path to the R file to modify. If NULL and RStudio is available, the currently active document path is used.
use_packages	A character vector of package names to process. Defaults to current_packages().
ignore_functions	A character vector of function names to ignore. Defaults to imported_functions().
...	Additional arguments. Currently unused and must be empty.

Details

This function reads the specified R file, identifies function calls from the specified packages, and adds explicit namespace qualifiers (::) to those calls. The modified code is written back to the original file.

Value

Invisible NULL. This function is called for its side effect of modifying the specified file in place.

Functions

- package_func_call_explicit(): Processes all .R files in a package's R/ directory, adding explicit namespace qualifiers.

Examples

```
file <- tempfile(fileext = ".R")
writeLines("
starwars |>
 mutate(name, bmi = mass / ((height / 100)^2)) |>
 select(name:mass, bmi)
", file)
make_func_call_explicit(
  path = file,
  use_packages = c("dplyr"),
  ignore_functions = c("library", "require")
)
readLines(file) |> message()
```

package_lost_glue_brace *Detect Lost Glue Brace in* glue *and* cli *Expressions*

Description

Check whether { and } are balanced in all glue() / glue_data() and cli_*() string arguments within an R file. The file is parsed into an AST, then each string literal that is an argument to a target function is checked with a stack-based brace matcher. Any mismatches are reported with line number and a visual caret (^^^^) marker under the problematic region.

Usage

```
package_lost_glue_brace(path = NULL, test_included = TRUE, ...)

detect_lost_glue_brace(path = NULL, ...)
```

Arguments

path	A character string specifying the path to the R file to inspect. If NULL and RStudio is available, the currently active document path is used.
test_included	Whether to include test (test/testthat/*) files in the check.
...	unused

Value

Invisibly returns TRUE if all expressions are balanced, FALSE otherwise. Side-effect messages are emitted via cli::cli.

Functions

- package_lost_glue_brace(): Scans all .R files in an R package (and optionally tests/testthat/), aggregated with per-file reporting.

Examples

```
file <- tempfile()
writeLines("glue(\"{a\")", file)
detect_lost_glue_brace(file)
```

package_print_and_cat *Detect* print() *and* cat() *Calls (CRAN-Unsafe)*

Description

Check whether R source files contain direct calls to print() or cat(), which are generally not permitted by CRAN policies. Output should use message instead.

These functions parse R source code into an AST and identify every SYMBOL_FUNCTION_CALL token whose text is "print" or "cat". Each match is reported with the line number, the full source line, and a caret marker pointing at the offending call.

When fix = TRUE, the function performs a simple text replacement of print(and cat(with message(on the affected lines. The replacement uses word-boundary matching to avoid false positives inside other identifiers (e.g. sprintf or print.myclass).

Usage

```
package_print_and_cat(path = NULL, test_included = TRUE, fix = FALSE, ...)

detect_print_and_cat(path = NULL, fix = FALSE, ...)
```

Arguments

path	For detect_print_and_cat(): path to an R file. If NULL and RStudio is available, the active document path is used. For package_print_and_cat(): path to the root directory of an R package. If NULL, the function walks up from the active document to find the package root.
test_included	Logical, used only by package_print_and_cat(). If TRUE (the default), .R files under tests/testthat/ are also scanned.
fix	Logical. If TRUE, replace print(/cat(with message(directly in the source file(s). Default is FALSE.
...	Additional arguments passed to utils::methods (currently unused).

Value

Invisibly returns TRUE if no calls were found, FALSE otherwise. Side-effect messages and caret markers are emitted via **cli** and message.

Functions

- package_print_and_cat(): Scans all .R files in an R package (and optionally tests/testthat/), aggregated with per-file reporting.

Single file vs package scope

`detect_print_and_cat()` Operates on one R file. When `path` is `NULL` and RStudio is available, the currently active document is used automatically.

`package_print_and_cat()` Scans all `.R` files in a package's `R/` directory, plus `tests/testthat/` when `test_included = TRUE`. Results are aggregated into a single report showing per-file summaries.

Examples

```
# --- Single file ---
tmp <- tempfile(fileext = ".R")
writeLines('print("hello")', tmp)
detect_print_and_cat(tmp)

# --- With auto-fix ---
detect_print_and_cat(tmp, fix = TRUE)

# --- Entire package ---
pkg <- tempfile()
dir.create(file.path(pkg, "R"), recursive = TRUE)
writeLines('cat("debug\\n")', file.path(pkg, "R", "example.R"))
writeLines(c("Package: example", "Version: 0.0.1"),
           file.path(pkg, "DESCRIPTION"))
package_print_and_cat(pkg)
```

`rename_func` *Rename Function Definitions in an R File*

Description

Renames function definitions in an R file to follow a consistent naming convention. Supports `"snake_case"`, `"camelCase"`, `"PascalCase"`, and `"google"` (dot-separated) naming styles. All references to the renamed functions within the file are also updated.

Usage

```
rename_func(
  path = NULL,
  style = c("snake_case", "camelCase", "PascalCase", "google"),
  ...
)
```

Arguments

path	A character string specifying the path to the R file to modify. If NULL and RStudio is available, the currently active document path is used.
style	Naming convention to apply. One of: • "snake_case": all lowercase with underscores (e.g., my_function) • "camelCase": lower camel case (e.g., myFunction) • "PascalCase": upper camel case (e.g., MyFunction) • "google": dot-separated lowercase (e.g., my.function)
...	Additional arguments. Currently unused and must be empty.

Details

Function definitions are identified using the pattern `name <- function(`, `name = function(`, or the R 4.1+ shorthand `name <- \\(` / `name = \\(`. Both the definition site and all call sites / references within the file are updated. The conversion handles mixed existing styles (snake_case, camelCase, PascalCase, dot.separated) and normalizes function names to the target style.

Value

Invisibly returns the path to the modified file.

Examples

```
temp <- tempfile(fileext = ".R")
writeLines("foo_bar <- function(){message('foo_bar')}", temp)
rename_func(temp, style = "camelCase")
readLines(temp)
rename_func(temp, style = "snake_case")
readLines(temp)
```

---

render_rmd *Render an R Markdown or R document to Markdown format*

Description

Render an R Markdown or R document to Markdown format

Usage

```
render_rmd(path = NULL, output_format = "md_document", ...)
```

Arguments

path	Path to the input file. If NULL and rstudioapi is available, uses the currently active document in RStudio.
output_format	Output format to render to. Defaults to "md_document".
...	Additional arguments passed to rmarkdown::render.

Value

The output file path from rmarkdown::render.

Examples

```
rlang::check_installed("rmarkdown")
tmp <- tempfile(fileext = ".Rmd")
writeLines(c("---", "title: Test", "---", "", "Hello, world!"), tmp)
render_rmd(path = tmp)
```

---

update_time_in_standalone *Update the last-updated field in standalone R files*

Description

Updates the `last-updated` field in the YAML header of standalone files to today's date. Supports single file or batch update for all standalone files in a directory.

Usage

```
update_time_in_standalone(path = NULL)
```

Arguments

path	Character. Path to a standalone file or a directory. If `NULL` and `rstudioapi` is available, uses the currently active document. Defaults to NULL.

Value

Invisibly returns a character vector of updated file paths.

Examples

```
update_time_in_standalone(tempdir())
```

use_hexsticker *Use Hex Sticker in README*

Description

[Deprecated] Adds a hex sticker image (optionally wrapped in a hyperlink) at the end of the top-level heading (the line starting with #) in README.md. This is a common pattern for R package README files.

Usage

```
use_hexsticker(
  img_path,
  url = NULL,
  alt_text = "package logo",
  height = 139L,
  align = "right",
  path = NULL,
  ...
)
```

Arguments

img_path	Character. Path to the image file (e.g., "man/figures/logo.png").
url	Character. URL the image should link to. If NULL (default), no hyperlink is added.
alt_text	Character. Alt text for the image. Defaults to "package logo".
height	Numeric or character. Image height in pixels. Defaults to 139.
align	Character. Image alignment attribute. Defaults to "right".
path	Character. Path to the package root directory. If NULL (the default), uses the current working directory.
...	Additional HTML attributes to include in the <img> tag as named arguments.

Value

Invisibly returns TRUE on success.

Examples

```
## Not run:
temp <- tempdir()
usethis::create_package(path = temp)
writeLines("# Package Name", file.path(temp, "README.md"))
file.create(file.path(temp, "logo.png"))
use_hexsticker(file.path(temp, "logo.png"), url = "https://my-pkg-website.com", path = temp)

## End(Not run)
```

use_multilanguage_readme *Create Multilingual README Files*

Description

Creates translated README files under `inst/` for a target R package and prints badges that can be pasted into the main `README.md` to link to each translation.

The five non-English United Nations official languages are used by default: Chinese (`zh-cn`), Spanish (`es`), French (`fr`), Arabic (`ar`), and Russian (`ru`). Any language code (including non-UN languages) can be supplied via the `lang` argument.

Usage

```
use_multilanguage_readme(
  lang = c("zh-cn", "es", "fr", "ar", "ru"),
  color = "blue",
  ...,
  path = NULL,
  overwrite = FALSE
)
```

Arguments

`lang`	Character vector of language codes (e.g. `"zh-cn"`, `"ja"`). Defaults to the five non-English UN official languages. Codes must be supported by the internal name mapping (see **Language Codes** below) or they will be used as-is for badge labels.
`color`	Color of badge. Defaults to `"blue"`
`...`	Not used.
`path`	Character. Path to the package root directory. If `NULL` (the default), uses the current working directory.
`overwrite`	Logical. If `TRUE`, overwrite existing README translation files. Defaults to `FALSE`.

Value

Invisibly returns a character vector of paths to the created files.

Language Codes

The following codes have built-in display name mappings:

Code	Display Name
en	English

zh-cn	Chinese (Simplified)
zh-tw	Chinese (Traditional)
es	Spanish
fr	French
de	German
pt	Portuguese
ja	Japanese
ko	Korean
ar	Arabic
ru	Russian
it	Italian
nl	Dutch
pl	Polish
tr	Turkish
vi	Vietnamese
th	Thai
id	Indonesian
hi	Hindi

Unrecognised codes are used verbatim as badge labels.

Examples

```
## Not run:
dir <- tempdir()
usethis::create_package(dir)
use_multilanguage_readme(path = dir)

# Custom languages
use_multilanguage_readme(c("de", "ja", "ko"), path = dir)

## End(Not run)
```

`use_r_v4.1.0` *Add a minimum* R *version dependency to a package*

Description

Adds `R (>= 4.1.0)` to the `Depends` field of a package's `DESCRIPTION` file. This sets a minimum R version requirement for your package. Then you can use `\()` and `|>` syntax in your package.

- Requires that the target directory is an R package root (contains a `DESCRIPTION` file).
- Calls `usethis::use_package()` to add the dependency.

Usage

```
use_r_v4.1.0(path = NULL, ...)
```

Arguments

path	Path to the package root. If NULL (the default), the current working directory is used.
...	Must be empty. Reserved for future arguments.

Value

Invisibly returns NULL, called for side effects.

Examples

```
tmpdir <- tempdir()
usethis::create_package(path = tmpdir)
use_r_v4.1.0(path = tmpdir)
```

---

use_vendor *Use a Vendor Package*

Description

Reference a permissively-licensed R package from GitHub for inclusion in your own R package. This function:

- Creates inst/vendor/pkg/ with LICENSE files and a README
- Creates R/vendor-pkg.R with attribution header and optional vendored code
- Updates DESCRIPTION (Authors@R and Copyright fields)
- Prints an acknowledgement snippet for your README

Usage

```
use_vendor(pkg, ..., branch = "main", path = NULL)
```

Arguments

pkg	GitHub repository specification in "owner/repo" form or a full GitHub URL (e.g. "https://github.com/owner/repo").
...	File paths within the vendor package to copy into your package and append to the vendor R file. If empty (default), only the infrastructure is set up.
branch	Github repository branch name. Defaults to "main"
path	Path to the target package directory. If NULL (the default), uses the current working directory.

Value

Invisibly returns NULL, called for side effects.

Examples

```
## Not run:
use_vendor("wurli/pedant")
use_vendor("https://github.com/wurli/pedant", "R/add_double_colons.R")

## End(Not run)
```

`use_workflow_version_update` *Create a GitHub Actions workflow to auto-update R package version*

Description

Copies the built-in `version_update.yml` workflow template to the target package's `.github/workflows/` directory. The workflow automatically bumps the R package version based on commit messages, or manually via `workflow_dispatch` with a specified version type.

Version bump rules (from commit messages, case-insensitive):

- **major / breaking** - increments the major version (X.0.0)
- **feat / feature / minor** - increments the minor version (x.Y.0)
- **patch / fix / bug** - increments the patch version (x.y.Z)
- Otherwise, no version bump occurs

The `workflow_dispatch` input always overrides commit message detection.

Usage

```
use_workflow_version_update(
  path = NULL,
  overwrite = FALSE,
  color = "blue",
  ...
)
```

Arguments

`path`	Character. Path to the package root directory. If `NULL` (the default), uses the current working directory.
`overwrite`	Logical. If `TRUE`, overwrite an existing workflow file. Defaults to `FALSE`.
`color`	badge color
`...`	pass to `badger::badge_devel`

Value

Invisibly returns the path to the created workflow file.

Examples

```
## Not run:
temp <- tempdir()
usethis::create_package(temp)
use_workflow_version_update(temp)
use_workflow_version_update(temp, overwrite = TRUE)

## End(Not run)
```

`use_zzz` *Create a (pkgname)-package.R file from a template*

Description

Copies the built-in `zzz_template.R` to the target package's `R/` directory and replaces template placeholders (all-caps words) with values from the package `DESCRIPTION` file.

Template placeholders replaced:

- `PKG` — package name
- `PKG-package` — `{pkgname}-package`
- `TITLE` — package title
- `DESCRIPTION` — package description (multiline values get `#'` prefix)
- `LICENSE` — license type

Other information:

- `.onLoad` and `.onAttach` is added to the file.
- `usethis` namespace is added to the file.

Usage

```
use_zzz(
  path = NULL,
  file_name = paste0(get_package_name(path = path), "-package.R"),
  overwrite = FALSE,
  open = rlang::is_interactive(),
  ...
)
```

Arguments

`path`	Character. Path to the package root directory. If `NULL` (the default), uses the current working directory.
`file_name`	Character. Output file name. Defaults to `"<pkg_name>-package.R"`.
`overwrite`	Logical. If `TRUE`, overwrite an existing file. Defaults to `FALSE`.
`open`	Logical. Whether to open the created file in the default editor.
`...`	Not used.

Value

Invisibly returns the path to the created file.

Examples

```
## Not run:
dir <- tempdir()
usethis::create_package(dir)
use_zzz(dir)

## End(Not run)
```

Index