

Package ‘packager’

May 24, 2025

Title Create, Build and Maintain Packages

Version 1.15.3

Description Helper functions for package creation, building and maintenance. Designed to work with a build system such as 'GNU make' or package 'fakemake' to help you to conditionally work through the stages of package development (such as spell checking, linting, testing, before building and checking a package).

License BSD_2_clause + file LICENSE

URL <https://gitlab.com/fvafrcu/packager>

Depends R (>= 3.3.0)

Imports callr, checkmate, codetools, crayon, cyclocomp, desc, fakemake (>= 1.10.1), fritools (>= 3.4.0), fs, gert, httr, pkgbuild, pkgload, rcmdcheck, remotes, rhub, rprojroot, tinytest, tools, whisker, whoami

Suggests cleanr, covr, devtools, digest, document, knitr, lintr, rasciidoc (>= 3.0.1), rmarkdown, roxygen2, RUnit, stats, spelling, testthat, usethis, withr

VignetteBuilder rasciidoc

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Andreas Dominik Cullmann [aut, cre]

Maintainer Andreas Dominik Cullmann <fvafrcu@mailbox.org>

Repository CRAN

Date/Publication 2025-05-24 14:30:02 UTC

Contents

packager-package	2
build_manual	3

check_archive	3
check_codetags	4
check_cyclomatic_complexity	5
check_news	6
check_usage	7
create	7
get_package_makelist	8
get_package_version	9
get_pkg_archive_path	9
git_add_commit	10
git_diff	11
git_tag	11
infect	12
lint_package	13
provide_cran_comments	13
submit	15
use_build_ignore	16
use_dev_version	16
use_directory	17
Index	18

packager-package	<i>Helps Me Create, Build and Maintain Packages</i>
------------------	---

Description

Helper functions for package creation, building and maintenance, heavily borrowing from **devtools** 1.13.3.

Details

You will find the details in
 vignette("An_Introduction_to_packager", package = "packager").

Author(s)

Maintainer: Andreas Dominik Cullmann <fvafrcu@mailbox.org>

See Also

Useful links:

- <https://gitlab.com/fvafrcu/packager>

build_manual	<i>Build a Package's Manual</i>
--------------	---------------------------------

Description

devtools version ([devtools::build_manual](#)) does not run **roxygen** first and by defaults puts it in `../` instead of `.Rcheck`

Usage

```
build_manual(  
  path = ".",  
  output_directory = NULL,  
  roxygenise = TRUE,  
  verbose = TRUE  
)
```

Arguments

path	Path to the package.
output_directory	Where to put the manual. Defaults to the Rcheck directory.
roxygenise	Run roxygen first?
verbose	Be verbose?

Value

[Invisibly](#) the value of the call to R CMD Rd2pdf.

check_archive	<i>Check a Package Archive</i>
---------------	--------------------------------

Description

This is a wrapper to [callr::rcmd_safe](#)("check"), similar to, but leaner than [rcmdcheck::rcmdcheck](#). While the latter parses the output of `rcmd_safe` and uses **clisymbols** in the callback, we here just return bare output and use [writeLines](#) as callback. This should result in a screen display that is identical to the output of R CMD check.

Usage

```
check_archive(path, cmdargs = NULL)  
  
check_archive_as_cran(path)
```

Arguments

path Path to the package archive.
 cmdargs Command line arguments (see `callr::rcmd`).

Value

A list with standard output, standard error and exit status of the check. (see `callr::rcmd`).

Note

`check_archive_as_cran` is a convenience Wrapper to `check_archive`.

See Also

Other maintenance functions: `check_codetags()`, `check_cyclomatic_complexity()`, `check_news()`, `check_usage()`, `get_check_status()`

Other maintenance functions: `check_codetags()`, `check_cyclomatic_complexity()`, `check_news()`, `check_usage()`, `get_check_status()`

Examples

```
## Not run:
package_path <- file.path(tempdir(), "fakepack")
usethis::create_package(path = package_path)
file.copy(system.file("templates", "throw.R", package = "fakemake"),
           file.path(package_path, "R"))
roxygen2::roxygenize(package_path)
print(tarball <- get_pkg_archive_path(package_path))
pkgbuild::build(pkg = package_path, path = package_path)
print(check_archive(tarball))

## End(Not run)
```

check_codetags

Check for Code Tags

Description

You do use code tags (see [PEP 350](#) for example)? This function searches for files under a directory containing such tags.

Usage

```
check_codetags(
  path = ".",
  exclude_pattern = "\\Rcheck/",
  include_pattern = "\\Rr$|\\Rrmd$",
  pattern = "XXX:|FIXME:|TODO:"
)
```

Arguments

path	to a directory, typically a package root.
exclude_pattern	A pattern for exclusions based on the file names. Stronger than include_pattern.
include_pattern	A pattern for inclusions based on the file names.
pattern	The pattern to search for.

Value

A character vector of hits.

See Also

Other maintenance functions: [check_archive\(\)](#), [check_cyclomatic_complexity\(\)](#), [check_news\(\)](#), [check_usage\(\)](#), [get_check_status\(\)](#)

Examples

```
dir <- system.file("runit_tests", package = "packager")
r <- check_codetags(dir)
print(r)
```

check_cyclomatic_complexity
Check Cyclomatic Complexity

Description

Run [cyclocomp_package_dir](#) on the package throwing an error when the maximum complexity is exceeded.

Usage

```
check_cyclomatic_complexity(path = ".", max_complexity = 10)
```

Arguments

path	Path to the package directory (see devtools::as.package).
max_complexity	The maximum cyclomatic complexity (which must not be exceeded).

Value

[Invisibly TRUE](#) if maximum cyclomatic complexity is not exceeded, throws an error otherwise.

See Also

Other maintenance functions: [check_archive\(\)](#), [check_codetags\(\)](#), [check_news\(\)](#), [check_usage\(\)](#), [get_check_status\(\)](#)

Examples

```
## Not run:
# download and untar sources of some archived package
package <- "excerptr"
root <- paste0("http://cran.r-project.org/src/contrib/Archive/", package)
version <- "1.0.0"
tarball <- paste0(paste(package, version, sep = "_"), ".tar.gz")
remote_tarball <- paste(root, tarball, sep = "/")
local_tarball <- file.path(tempdir(), tarball)
utils::download.file(remote_tarball, local_tarball)
utils::untar(local_tarball, exdir = tempdir())
res <- tryCatch(check_cyclomatic_complexity(path = file.path(tempdir(),
                                                             package)),
               error = identity)
print(res)

## End(Not run)
```

check_news

Check for 'NEWS.md' Being Up to Date

Description

Compare your 'NEWS.md' file to the 'Version' entry in DESCRIPTION.

Usage

```
check_news(path = ".")
```

Arguments

path Path to the package directory (see [devtools::as.package](#)).

Value

[Invisibly TRUE](#) if 'NEWS.md' matches DESCRIPTION, throws an error otherwise.

See Also

Other maintenance functions: [check_archive\(\)](#), [check_codetags\(\)](#), [check_cyclomatic_complexity\(\)](#), [check_usage\(\)](#), [get_check_status\(\)](#)

check_usage	Check Usage with codetools ' checkUsagePackage
-------------	---

Description

This is just a convenience wrapper to [checkUsagePackage](#) (which needs loading of the [development version of the] package).

Usage

```
check_usage(path = ".")
```

Arguments

path	Path to the package directory (see devtools::as.package).
------	--

Value

A character vector of issues found by [checkUsagePackage](#).

See Also

Other maintenance functions: [check_archive\(\)](#), [check_codetags\(\)](#), [check_cyclomatic_complexity\(\)](#), [check_news\(\)](#), [get_check_status\(\)](#)

create	Create a Package Template
--------	---------------------------

Description

This is just a wrapper to create a package and infect it using [infect](#).

Usage

```
create(path, force = TRUE, ...)
```

Arguments

path	The package to create.
force	Recursively unlink path before calling creating the package?
...	Arguments to be passed to infect .

Value

[Invisibly NULL](#).

See Also[infect](#)**Examples**

```

path <- file.path(tempdir(), "myFirstPackage")
packager::create(path = path, fakemake = "roxygen2")
list.files(path, recursive = TRUE)
## Not run:
if (require("roxygen2")) {
  ml <- packager::get_package_makelist(is_cran = TRUE)
  d <- file.path(tempdir(), "somePackage")
  dir.create(d)
  packager::create(d, fakemake = FALSE, fakemake = FALSE)
  withr::with_dir(d, fakemake::make("check", ml))
  check_log <- file.path(d, "log", "check.Rout")
  status <- packager::get_check_status(check_log)
  RUnit::checkEqualsNumeric(status[["status"]][["errors"]], 0)
  list.files(d, recursive = TRUE)
  unlink(d, recursive = TRUE)
}

## End(Not run)

```

get_package_makelist *Provide a makelist Suitable for Packages with **packager***

Description

Provide a makelist Suitable for Packages with **packager**

Usage

```
get_package_makelist(is_cran = FALSE, gitlab_token = NULL)
```

Arguments

is_cran Streamline makelist for usage on CRAN?

gitlab_token A private gitlab token. Used to query logs on <https://about.gitlab.com>.

Value

A list for [fakemake::make](#).

Examples

```
ml <- packager::get_package_makelist()
cbind(lapply(ml, function(x) x[["target"]]),
      lapply(ml, function(x) x[["alias"]]))

cl <- packager::get_package_makelist(is_cran = TRUE)
setdiff(sapply(ml, function(x) x[["target"]]),
        sapply(cl, function(x) x[["target"]]))
```

get_package_version	<i>Query Installed Package Versions</i>
---------------------	---

Description

See [fritools::get_package_version](#).

Usage

```
get_package_version(x, lib_loc = NULL)
```

Arguments

x	A character giving the package name.
lib_loc	See argument lib.loc in packageDescription .

Value

A character giving the package version.

get_pkg_archive_path	<i>Create a Package's Archive Path From the Package's 'DESCRIPTION'</i>
----------------------	---

Description

The archive file does not have to exist. Use `file.exists(get_pkg_archive_path())` to test existence.

Usage

```
get_pkg_archive_path(path = ".", absolute = TRUE)
```

Arguments

path	Path to the package directory (see devtools::as.package).
absolute	Return the absolute path?

Value

Path to the package's archive file.

Examples

```
package_path <- file.path(tempdir(), "anRpackage")
usethis::create_package(path = package_path)
print(tarball <- get_pkg_archive_path(package_path))
file.exists(tarball)
```

git_add_commit

Git Add All Changes and Commit

Description

Much like `git commit -am"M"`, where M is the message.

Usage

```
git_add_commit(
  path,
  message = "Uncommented Changes: Backing Up",
  untracked = FALSE
)
```

Arguments

path	The path to the repository.
message	The commit message to use.
untracked	Add files not tracked yet before committing?

Value

The return value of `gert::git_commit_all`.

See Also

Other git wrappers: `git_tag()`

git_diff	Show a Git Diff for a File
----------	----------------------------

Description

Show a Git Diff for a File

Usage

```
git_diff(x, path, verbose = TRUE)
```

Arguments

x	The path to the file relative to the repository given by path.
path	The path to the git repository.
verbose	Be verbose? This is the main purpose of this tiny little wrapper!

Value

The git diff.

git_tag	Create a Git Tag Based on the Current Version Number
---------	--

Description

This is basically the same as `git tag -a T -m M` where T is the version read from the package's DESCRIPTION file and M is given by message (see below).

Usage

```
git_tag(path = ".", tag_uncommitted = FALSE, message = "CRAN release")
```

Arguments

path	Path to the package.
tag_uncommitted	Tag if there are uncommitted changes?
message	The tag message to be used.

Value

`FALSE` or the value of `gert::git_tag_list`.

See Also

Other git wrappers: `git_add_commit()`

infect*Adjust a Package*

Description

Add a variety of extensions to a package (skeleton) and run `fakemake::make` on it.

Usage

```
infect(path, fakemake = "check", git_add_and_commit = TRUE, ...)
```

Arguments

<code>path</code>	Path to the package directory (see <code>devtools::as.package</code>).
<code>fakemake</code>	The name for a <code>makelist</code> for <code>fakemake</code> . Set to <code>NULL</code> or <code>FALSE</code> to disable running <code>fakemake::make</code> .
<code>git_add_and_commit</code>	Add and commit changes in git?
<code>...</code>	Arguments to be passed to <code>set_package_info</code> .

Value

Invisibly `NULL`.

See Also

`create`

Examples

```
## Not run:
if (require("roxygen2")) {
  path <- file.path(tempdir(), "mySecondPackage")
  usethis::create_package(path = path, open = FALSE)
  l1 <- list.files(path, recursive = TRUE)
  packager::infect(path = path, fakemake = "roxygen2", fakemake = FALSE)
  l2 <- list.files(path, recursive = TRUE)
  print(l1); print(l2)
  unlink(path, recursive = TRUE)
}

## End(Not run)
```

lint_package *Customize `lintr::lint_package`*

Description

lintr now runs **cyclocomp**, which we use independently and we don't want to run it twice. So this is just a wrapper to `lintr::lint_package` where we hardcode the exclusion of unwanted linters (more may be added to **lintr**) so other packages using **packager**'s 'Makefile' or `get_package_makelist` don't have to care of changes to the default linters in **lintr**.

Usage

```
lint_package(path)
```

Arguments

path The path to the package, passed to `lintr::lint_package`.

Value

See the return value of `lintr::lint_package`.

provide_cran_comments *Provide a Template for Your Comments To CRAN*

Description

`submit` reads a file 'cran-comments.md'. This function provides a template based on your R version, your R CMD check output and the package's 'NEWS.md'.

Usage

```
provide_cran_comments(
  check_log = NULL,
  path = ".",
  initial = FALSE,
  write_to_file = TRUE,
  private_token = NULL,
  name = NA,
  proxy = NULL
)
```

Arguments

check_log	Deprecated, will be removed in a future release. The local R CMD check log is now supposed to live be 'log/check.(log Rout)'.
path	Path to the package directory (see <code>devtools::as.package</code>).
initial	Is this an initial submission?
write_to_file	Do write the comment to 'cran-comment.md'?
private_token	Provide a private token to access https://about.gitlab.com .
name	The name to sign with, if NA, the given name of the package maintainer as stated in file DESCRIPTION is used.
proxy	A proxy to use.

Value

Character vector containing the CRAN comments, which are written to 'cran-comments.md' (see Note).

Note

By default this function writes to disk as side effect.

Examples

```
## Not run:

if (Sys.info()[["nodename"]] == "fvafrdebianCU") {
  gitlab_token <- readLines(file.path("~", ".gitlab_private_token.txt"))
  proxy <- httr::use_proxy("10.127.255.17", 8080)
  comments <- provide_cran_comments(path = ".",
                                    write_to_file = TRUE,
                                    private_token = gitlab_token,
                                    proxy = proxy)

} else {
  gitlab_token <- readLines(file.path("~", ".gitlab_private_token.txt"))
  comments <- provide_cran_comments(path = ".",
                                    write_to_file = TRUE,
                                    private_token = gitlab_token)
}

cat(comments, sep = "")

## End(Not run)
```

submit

Release a Package to CRAN

Description

This is a stripped version of **devtools**' [release function](#), omitting most of the interactive checks.

Usage

```
submit(  
  path = ".",  
  stop_on_git = TRUE,  
  stop_on_devel = TRUE,  
  force = FALSE,  
  verbose = TRUE,  
  consider_untracked = TRUE  
)  
  
release(  
  path = ".",  
  stop_on_git = TRUE,  
  stop_on_devel = TRUE,  
  force = FALSE,  
  verbose = TRUE,  
  consider_untracked = TRUE  
)
```

Arguments

<code>path</code>	The package's root directory.
<code>stop_on_git</code>	Stop if git has uncommitted changes or is not synced with the origin?
<code>stop_on_devel</code>	Stop if the package has a development version? (That is, a four part version.)
<code>force</code>	Skip user interaction?
<code>verbose</code>	Be verbose?
<code>consider_untracked</code>	Consider untracked files if <i>consider_untracked</i> is TRUE ?

Value

[Invisibly NULL](#)

Note

`release` is just a link to `submit` as [release](#) is the original function from **devtools**.

use_build_ignore	<i>Add Files to '.Rbuildignore'</i>
------------------	-------------------------------------

Description

This is verbatim copy of git commit a5e5805ecd630ebc46e080bd78ebcf32322efe3c of **usethis**.
 '.Rbuildignore' has a regular expression on each line, but it's usually easier to work with specific file names. By default, will (crudely) turn filenames into regular expressions that will only match these paths. Repeated entries will be silently removed.

Usage

```
use_build_ignore(files, escape = TRUE, pkg = ".")
```

Arguments

files	Paths of files.
escape	If TRUE, the default, will escape . to \. and surround with ^ and \$.
pkg	Path to the package directory (see as.package).

Value

Nothing, called for its side effect.

use_dev_version	<i>Use a Development Version in DESCRIPTION and 'NEWS.md'</i>
-----------------	---

Description

This is much like [usethis::use_dev_version](#), but their conventions keep changing.

Usage

```
use_dev_version(path = ".", force = FALSE)
```

```
use_devel_version(path = ".", force = FALSE)
```

Arguments

path	Path to the package directory (see devtools::as.package).
force	Set to TRUE to force version bumping with uncommitted git changes.

Note

From [usethis::use_dev_version](#), the name was use_dev_version, but use_devel_version seems more natural. But it is just a link.

use_directory	<i>Use a Directory</i>
---------------	------------------------

Description

Create a directory.
this is verbatim copy of git commit a5e5805ecd630ebc46e080bd78ebcf32322efe3c of **usethis**.

Usage

```
use_directory(path, ignore = FALSE, pkg = ".")
```

Arguments

path	Path of the directory to create, relative to the project.
ignore	Add the directory to ‘.Rbuildignore’?
pkg	Path to the package directory (see as.package).

Index

- * **git wrappers**
 - git_add_commit, [10](#)
 - git_tag, [11](#)
- * **maintenance functions**
 - check_archive, [3](#)
 - check_codetags, [4](#)
 - check_cyclomatic_complexity, [5](#)
 - check_news, [6](#)
 - check_usage, [7](#)
- * **package**
 - packager-package, [2](#)

as.package, [16](#), [17](#)

build_manual, [3](#)

callr::rcmd, [4](#)

callr::rcmd_safe, [3](#)

check_archive, [3](#), [5–7](#)

check_archive_as_cran(check_archive), [3](#)

check_codetags, [4](#), [4](#), [6](#), [7](#)

check_cyclomatic_complexity, [4](#), [5](#), [5](#), [6](#), [7](#)

check_news, [4–6](#), [6](#), [7](#)

check_usage, [4–6](#), [7](#)

checkUsagePackage, [7](#)

create, [7](#), [12](#)

cyclocomp_package_dir, [5](#)

devtools::as.package, [5–7](#), [9](#), [12](#), [14](#), [16](#)

devtools::build_manual, [3](#)

fakemake::make, [8](#), [12](#)

FALSE, [11](#), [12](#)

fritools::get_package_version, [9](#)

gert::git_commit_all, [10](#)

gert::git_tag_list, [11](#)

get_check_status, [4–7](#)

get_package_makelist, [8](#), [13](#)

get_package_version, [9](#)

get_pkg_archive_path, [9](#)

git_add_commit, [10](#), [11](#)

git_diff, [11](#)

git_tag, [10](#), [11](#)

infect, [7](#), [8](#), [12](#)

Invisibly, [3](#), [5–7](#), [12](#), [15](#)

lint_package, [13](#)

lintr::lint_package, [13](#)

makelist, [12](#)

NULL, [7](#), [12](#), [15](#)

packageDescription, [9](#)

packager(packager-package), [2](#)

packager-package, [2](#)

provide_cran_comments, [13](#)

rcmdcheck::rcmdcheck, [3](#)

release, [15](#)

release(submit), [15](#)

set_package_info, [12](#)

submit, [13](#), [15](#)

TRUE, [5](#), [6](#), [15](#)

unlink, [7](#)

use_build_ignore, [16](#)

use_dev_version, [16](#)

use_devel_version(use_dev_version), [16](#)

use_directory, [17](#)

usethis::use_dev_version, [16](#)

writeLines, [3](#)