

Package: packager (via r-universe)

September 18, 2026

Title Create, Build and Maintain Packages

Version 1.15.3.9000

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, pkgdown, rasciidoc (>= 3.0.1), rmarkdown, roxygen2, RUnit, stats, spelling, testthat, usethis, withr

VignetteBuilder rasciidoc

Encoding UTF-8

RoxygenNote 7.3.3

Config/pak/sysreqs cmake git libglpk-dev make libgit2-dev libuv1-dev libxml2-dev libssl-dev

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

Date/Publication 2026-04-16 20:32:01 UTC

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

RemoteRef HEAD

RemoteSha 3456e8f9d1e92711ed1c93ece90c9c842def6f74

Contents

packager-package	2
build_manual	3
bump_package	3
bump_version	4
check_archive	5
check_codetags	6
check_cyclomatic_complexity	7
check_news	8
check_usage	8
create	9
get_package_makelist	10
get_package_version	10
get_pkg_archive_path	11
git_add_commit	11
git_diff	12
git_tag	13
infect	13
lint_package	14
provide_cran_comments	15
submit	16
use_build_ignore	17
use_dev_version	18
use_directory	18
Index	19

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

Build a Package's Manual

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.

 bump_package

Bump Version in DESCRIPTION and NEWS.md

Description

Regularly need to bump a version, there are packages for that, but not invented here.

Usage

```
bump_package(which, path = ".")
```

Arguments

which	The version part to bump.
path	The path to the package.

Value

Invisibly TRUE.

bump_version	<i>Bump a Version</i>
--------------	-----------------------

Description

Regularly need to bump a version, there are packages for that, but not invented here.

Usage

```
bump_version(x, which = c("dev", "patch", "minor", "major"))
```

Arguments

x	A version string or numeric_version .
which	The version part to bump.

Value

The bumped version.

Examples

```
bump_version("1.12.3", "minor")
bump_version("1.12.3", "patch")
bump_version("1.12.3", "dev")
bump_version("1.12.3.9000", "dev")
bump_version("1.12.3", "major")
```

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 <code>callr::rcmd</code>).

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	<i>Check for Code Tags</i>
----------------	----------------------------

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]$|\\..[Rr]md$",  
  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	<i>Check for 'NEWS.md' Being Up to Date</i>
------------	---

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	<i>Check Usage with codetools' checkUsagePackage</i>
-------------	---

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 *Query Installed Package Versions*

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	<i>Git Add All Changes and Commit</i>
----------------	---------------------------------------

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	<i>Show a Git Diff for a File</i>
----------	-----------------------------------

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

path	Path to the package directory (see <code>devtools::as.package</code>).
fakemake	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> .
git_add_and_commit	Add and commit changes in git?
...	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

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

path	The package's root directory.
stop_on_git	Stop if git has uncommitted changes or is not synced with the origin?
stop_on_devel	Stop if the package has a development version? (That is, a four part version.)
force	Skip user interaction?
verbose	Be verbose?
consider_untracked	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, 11
 - git_tag, 13
- * **maintenance functions**
 - check_archive, 5
 - check_codetags, 6
 - check_cyclomatic_complexity, 7
 - check_news, 8
 - check_usage, 8
- * **package**
 - packager-package, 2

as.package, 17, 18

build_manual, 3

bump_package, 3

bump_version, 4

callr::rcmd, 5

callr::rcmd_safe, 5

check_archive, 5, 6–8

check_archive_as_cran(check_archive), 5

check_codetags, 5, 6, 7, 8

check_cyclomatic_complexity, 5, 6, 7, 8

check_news, 5–7, 8, 8

check_usage, 5–7, 8, 8

checkUsagePackage, 8

create, 9, 14

cyclocomp_package_dir, 7

devtools::as.package, 7, 8, 11, 13, 15, 18

devtools::build_manual, 3

fakemake::make, 10, 13

FALSE, 13

fritools::get_package_version, 10

gert::git_commit_all, 12

gert::git_tag_list, 13

get_check_status, 5–8

get_package_makelist, 10, 14

get_package_version, 10

get_pkg_archive_path, 11

git_add_commit, 11, 13

git_diff, 12

git_tag, 12, 13

infect, 9, 13

Invisibly, 3, 4, 7–9, 14, 17

lint_package, 14

lintr::lint_package, 14

makelist, 13

NULL, 9, 13, 14, 17

numeric_version, 4

packageDescription, 10

packager(packager-package), 2

packager-package, 2

provide_cran_comments, 15

rcmdcheck::rcmdcheck, 5

release, 17

release(submit), 16

set_package_info, 13

submit, 15, 16

TRUE, 4, 7, 8, 17

unlink, 9

use_build_ignore, 17

use_dev_version, 18

use_devel_version(use_dev_version), 18

use_directory, 18

usethis::use_dev_version, 18

writeLines, 5