

# Package ‘packager’

January 29, 2021

**Title** Create, Build and Maintain Packages

**Version** 1.10.0

**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 (>= 1.2.0), gert, httr, pkgbuild, pkgload, rcmdcheck, remotes, rhub, rprojroot, tools, usethis, whisker, whoami, withr

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

**VignetteBuilder** rasciidoc

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Andreas Dominik Cullmann [aut, cre]

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

**Repository** CRAN

**Date/Publication** 2021-01-29 13:50:02 UTC

## R topics documented:

packager-package . . . . .	2
check_archive . . . . .	3
check_codetags . . . . .	4
check_cyclomatic_complexity . . . . .	5
check_news . . . . .	6
check_usage . . . . .	6
create . . . . .	7
get_package_makelist . . . . .	8
get_package_version . . . . .	8
get_pkg_archive_path . . . . .	9
git_add_commit . . . . .	9
git_diff . . . . .	10
git_tag . . . . .	11
infect . . . . .	11
lint_package . . . . .	12
provide_cran_comments . . . . .	13
submit . . . . .	14
use_build_ignore . . . . .	15
use_dev_version . . . . .	16
use_directory . . . . .	16
<b>Index</b>	<b>17</b>

---

packager-package      *Helps Me Create, Build and Maintain Packages*

---

### 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").

---

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 <b>codetools</b>' <a href="#">checkUsagePackage</a></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 using `usethis::create_package` and infect it using `infect`.

## Usage

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

## Arguments

<code>path</code>	See <code>usethis::create_package</code> .
<code>force</code>	Recursively <code>unlink</code> path before calling <code>usethis::create_package(path)</code> ?
<code>...</code>	Arguments to be passed to <code>infect</code> .

## 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

```
m1 <- packager::get_package_makelist()
cbind(lapply(m1, function(x) x[["target"]]),
      lapply(m1, function(x) x[["alias"]]))

c1 <- packager::get_package_makelist(is_cran = TRUE)
setdiff(sapply(m1, function(x) x[["target"]]),
        sapply(c1, 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    *Create a Package's Archive Path From the Package's 'DESCRIPTION'*

---

**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 <a href="#">devtools::as.package</a> ).
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	<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	<i>Create a Git Tag Based on the Current Version Number</i>
---------	---

---

**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	<i>Adjust a Package</i>
--------	-------------------------

---

**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 indendently and we don't want to run it twice. So this is just a wrapper to code [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 <code>'log/check.(log Rout)'</code> .
<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 <code>'cran-comment.md'</code> ?
<code>private_token</code>	Provide a private token to access <a href="https://about.gitlab.com">https://about.gitlab.com</a> .
<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"]] == "fvafrebianCU") {
  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
)

release(
  path = ".",
  stop_on_git = TRUE,
  stop_on_devel = TRUE,
  force = FALSE,
  verbose = 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?

**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 <a href="#">as.package</a> ).

**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 <code>devtools::as.package</code> ).
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 <code>`.Rbuildignore`</code> ?
pkg	Path to the package directory (see <code>as.package</code> ).



# Index

- \* **git wrappers**
  - git\_add\_commit, 9
  - git\_tag, 11
- \* **maintenance functions**
  - check\_archive, 3
  - check\_codetags, 4
  - check\_cyclomatic\_complexity, 5
  - check\_news, 6
  - check\_usage, 6
- \* **package**
  - packager-package, 2

as.package, 15, 16

callr::rcmd, 3

callr::rcmd\_safe, 3

check\_archive, 3, 4–6

check\_archive\_as\_cran (check\_archive), 3

check\_codetags, 3, 4, 5, 6

check\_cyclomatic\_complexity, 3, 4, 5, 6

check\_news, 3–6, 6

check\_usage, 3–6, 6

checkUsagePackage, 6

create, 7, 12

cyclocomp\_package\_dir, 5

devtools::as.package, 5, 6, 9, 11, 13, 16

fakemake::make, 8, 11

FALSE, 11

fritools::get\_package\_version, 8

gert::git\_commit\_all, 10

gert::git\_tag\_list, 11

get\_check\_status, 3–6

get\_package\_makelist, 8, 12

get\_package\_version, 8

get\_pkg\_archive\_path, 9

git\_add\_commit, 9, 11

git\_diff, 10

git\_tag, 10, 11

infect, 7, 11

Invisibly, 5–7, 12, 15

lint\_package, 12

lintr::lint\_package, 12

makelist, 11

NULL, 7, 11, 12, 15

packageDescription, 8

packager-package, 2

provide\_cran\_comments, 13

rcmdcheck::rcmdcheck, 3

release, 15

release (submit), 14

release function, 14

set\_package\_info, 11

submit, 13, 14

TRUE, 5, 6

unlink, 7

use\_build\_ignore, 15

use\_dev\_version, 16

use\_devel\_version (use\_dev\_version), 16

use\_directory, 16

usethis::create\_package, 7

usethis::use\_dev\_version, 16

writeLines, 3