

Package: fledge (via r-universe)

October 10, 2026

Title Smoother Change Tracking and Versioning for R Packages

Version 0.1.99.9074

Date 2026-10-10

Description Streamlines the process of updating changelogs (NEWS.md) and versioning R packages developed in git repositories.

License GPL-3

URL <https://fledge.cynkra.com/>, <https://github.com/cynkra/fledge>

BugReports <https://github.com/cynkra/fledge/issues>

Imports brio, callr, cli, curl, desc (>= 1.4.1), devtools, gh (>= 1.5.0), glue, httr2, lifecycle, memoise, pandoc, parsedate, purrr (>= 0.3.2), rematch2, rlang (>= 0.4.12), tibble, usethis (>= 2.1.5), whoami, xml2

Suggests clipr, covr, digest, fansi, foghorn, fs, gert (>= 1.4.0), httptest2, job, jsonlite, knitr, pkgbuild, pkgload, rhub, rmarkdown, rstudioapi, testthat (>= 3.1.10), urlchecker, vctrs, whisker, withr

VignetteBuilder knitr

Config/autostyle/scope line_breaks

Config/autostyle/strict true

Config/Needs/readme r-lib/downlit@f-readme-document

Config/Needs/website cynkra/cynkratemplate

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first auto, demo, parse-news-items, bump_version

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1.9000

Config/pak/sysreqs cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev git make libharfbuzz-dev libgit2-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev libx11-dev zlib1g-dev

Repository <https://cynkra.r-universe.dev>
Date/Publication 2026-10-10 21:51:50 UTC
RemoteUrl <https://github.com/cynkra/fledge>
RemoteRef HEAD
RemoteSha 261e9b8c06f451a4e484551ccdb0507d95b9ff9c

Contents

bump_version	2
commit_version	4
create_demo_project	4
finalize_version	5
get_last_tag	6
get_last_version_tag	7
get_top_level_commits	8
plan_release	9
tag_version	10
unbump_version	11
update_news	12
with_demo_project	13
Index	15

bump_version	<i>Bump package version</i>
--------------	-----------------------------

Description

- Calls the following functions:
1. Verify that the current branch is the main branch if `check_default_branch = TRUE` (the default).
 2. `update_news()`, using the `which` argument, applying logic to determine if meaningful changes have been made since the last version.
 3. From the result, check if there were meaningful changes since the last version.
 4. Depending on the `which` argument:
 - If "dev", `finalize_version()` with `push = FALSE`
 - Otherwise, `commit_version()`.

Usage

```
bump_version(  
  which = c("dev", "pre-patch", "patch", "pre-minor", "minor", "pre-major", "major"),  
  ...,  
  no_change_behavior = c("bump", "noop", "fail"),  
  check_default_branch = TRUE  
)
```

Arguments

which	Component of the version number to update. Supported values are • "auto" (default: "samedev" or "dev", depending on contents of NEWS.md), • "samedev" (a.b.c.900x with stable version), • "dev" (a.b.c.9xxx), • "patch" (a.b.x), • "pre-minor" (a.b.99.9000), • "minor" (a.x.0), • "pre-major" (a.99.99.9000), • "major" (x.0.0).
...	These dots are for future extensions and must be empty.
no_change_behavior	What to do if there was no change since the last version: "bump" for bump the version; "noop" for do nothing; "fail" for erroring.
check_default_branch	Whether to check that the current branch is the default branch.

Value

TRUE if NEWS.md and DESCRIPTION have been updated, FALSE otherwise. Do not rely on this behavior.

Bumped too soon?

Have you just run bump_version(), then realized "oh shoot, I forgot to merge that PR"? Fear not, run [unbump_version\(\)](#), merge that PR, run bump_version().

See Also

[unbump_version\(\)](#)

Examples

```
# Create mock package in a temporary directory.
# Set open to TRUE if you want to play in the mock package.
with_demo_project({
  # Use functions as if inside the newly created package project.
  # (Or go and actually run code inside the newly created package project!)
  # Add a new R file.
  usethis::use_r("cool-function", open = FALSE)
  # Pretend we added useful code inside it.
  # Track the new R file with Git.
  gert::git_add("R/cool-function.R")
  gert::git_commit("- Add cool function.")
  # Bump version with fledge.
  fledge::bump_version()
})
```

commit_version	<i>Commits NEWS.md and DESCRIPTION to Git</i>
----------------	---

Description

Commits changes to NEWS.md and DESCRIPTION, amending a previous commit created by **fledge** if necessary.

Usage

```
commit_version()
```

Value

Invisibly: TRUE if a previous commit for that version has been amended, FALSE if not.

Examples

```
# Create mock package in a temporary directory.
# Set open to TRUE if you want to play in the mock package.
with_demo_project({
  # Use functions as if inside the newly created package project.
  # (Or go and actually run code inside the newly created package project!)
  # Add a new R file.
  usethis::use_r("cool-function", open = FALSE)
  # Pretend we added useful code inside it.
  # Track the new R file with Git.
  gert::git_add("R/cool-function.R")
  gert::git_commit("- Add cool function.")
  # Bump version with fledge.
  fledge::bump_version()
  desc::desc_add_author(given = "Jane", family = "Doe", role = "ctb")
  fledge::commit_version()
})
```

create_demo_project	<i>Create example repo for fledge demos</i>
---------------------	---

Description

Create example repo for fledge demos

Usage

```
create_demo_project(
  open = rlang::is_interactive(),
  name = "tea",
  maintainer = NULL,
  email = NULL,
  date = "2021-09-27",
  dir = file.path(tempdir(), "fledge"),
  news = FALSE
)
```

Arguments

open	Whether to open the new project.
name	Package name.
maintainer	Name for DESCRIPTION and git.
email	Email for DESCRIPTION and git.
date	String of time for DESCRIPTION and git.
dir	Directory within which to create the mock package folder.
news	If TRUE, create a NEWS.md file.

Value

The path to the newly created mock package.

finalize_version	<i>Finalize package version</i>
------------------	---------------------------------

Description

Calls the following functions:

1. `commit_version()`
2. `tag_version()` with `force = TRUE`
3. Force-pushes the created tag to the "origin" remote, if `push = TRUE`.

Usage

```
finalize_version(push = FALSE)
```

Arguments

push	If TRUE, push the created tag.
------	--------------------------------

Value

None

Examples

```
# Create mock package in a temporary directory.
# Set open to TRUE if you want to play in the mock package.
with_demo_project({
  # Use functions as if inside the newly created package project.
  # (Or go and actually run code inside the newly created package project!)
  # Add a new R file.
  usethis::use_r("cool-function", open = FALSE)
  # Pretend we added useful code inside it.
  # Track the new R file with Git.
  gert::git_add("R/cool-function.R")
  gert::git_commit("- Add cool function.")
  # Bump version with fledge.
  fledge::bump_version()
  # Edit news by hand
  # ...
  # Once done
  fledge::finalize_version()
})
```

get_last_tag

*The most recent tag***Description**

Returns the most recent Git tag that is reachable from the current branch. This will not be accurate if the version bump and tag has occurred on a branch which has been squashed into the main branch. See [get_last_version_tag\(\)](#) for a version that works in this scenario.

Usage

```
get_last_tag()
```

Value

A one-row tibble with columns name, ref and commit. For annotated tags (as created by fledge), commit may be different from the SHA of the commit that this tag points to. Use [gert::git_log\(\)](#) to find the actual commit.

Examples

```
# Create mock package in a temporary directory.
# Set open to TRUE if you want to play in the mock package.
with_demo_project({
  # Use functions as if inside the newly created package project.
```

```
# (Or go and actually run code inside the newly created package project!)
# Add a new R file.
usethis::use_r("cool-function", open = FALSE)
# Pretend we added useful code inside it.
# Track the new R file with Git.
gert::git_add("R/cool-function.R")
gert::git_commit("- Add cool function.")
# Bump version with fledge.
fledge::bump_version()
fledge::finalize_version()
print(get_top_level_commits(since = NULL))
print(fledge::get_last_tag())
})
```

get_last_version_tag *The most recent versioned tag*

Description

Returns the Git tag of the form vx.y, vx.y.z or vx.y.z.w with the latest version. An older version of this logic is used in [get_last_tag\(\)](#), which traverses the Git history but does not work with squash-merging of version bumps.

Usage

```
get_last_version_tag()
```

Value

A one-row tibble with columns name, ref and commit. For annotated tags (as created by fledge), commit may be different from the SHA of the commit that this tag points to. Use [gert::git_log\(\)](#) to find the actual commit.

Examples

```
# Create mock package in a temporary directory.
# Set open to TRUE if you want to play in the mock package.
with_demo_project({
  # Use functions as if inside the newly created package project.
  # (Or go and actually run code inside the newly created package project!)
  # Add a new R file.
  usethis::use_r("cool-function", open = FALSE)
  # Pretend we added useful code inside it.
  # Track the new R file with Git.
  gert::git_add("R/cool-function.R")
  gert::git_commit("- Add cool function.")
  # Switch to branch for bumping version.
  gert::git_branch_create("fledge")
  # Bump version with fledge.
  fledge::bump_version(check_default_branch = FALSE)
```

```
fledge::finalize_version()

# Merge the version bump branch into main.
gert::git_branch_checkout("main")
gert::git_merge("fledge", squash = TRUE)

print(get_top_level_commits(since = NULL))

# get_last_tag() doesn't work in this scenario
print(fledge::get_last_tag())

# get_last_version_tag() is better
print(fledge::get_last_version_tag())
})
```

get_top_level_commits *All top-level commits*

Description

Return all top-level commits since a particular version as commit objects.

Usage

```
get_top_level_commits(since = NULL)
```

Arguments

since	A commit SHA, e.g. as returned in the commit component of get_last_version_tag() . If NULL, the entire log is retrieved.
-------	---

Value

A [tibble](#) with at least two columns:

- commit: the commit SHA
- message: the commit message

Examples

```
# Create mock package in a temporary directory.
# Set open to TRUE if you want to play in the mock package.
with_demo_project({
  # Use functions as if inside the newly created package project.
  # (Or go and actually run code inside the newly created package project!)
  # Add a new R file.
  usethis::use_r("cool-function", open = FALSE)
  # Pretend we added useful code inside it.
  # Track the new R file with Git.
  gert::git_add("R/cool-function.R")
})
```

```

gert::git_commit("- Add cool function.")
# Bump version with fledge.
fledge::bump_version()
fledge::finalize_version()
print(get_top_level_commits(since = NULL))
print(fledge::get_last_tag())
})

```

plan_release

Automating CRAN release

Description

plan_release() is run when a milestone in the development of a package is reached and it is ready to be sent to CRAN. By default, this function will initiate a pre-release, indicated by 9900 in the fourth component of the version number. Pass which = "patch", which = "minor", or which = "major" to initiate a release with the corresponding version number.

release() sends to CRAN after performing several checks, and offers help with accepting the submission.

post_release() should be called after the submission has been accepted.

Usage

```

plan_release(
  which = c("pre-patch", "pre-minor", "pre-major", "next", "patch", "minor", "major"),
  force = FALSE
)

release()

post_release()

```

Arguments

which	Component of the version number to update. Supported values are • "pre-patch" (default, x.y.z.9900) • "pre-minor" (x.y.99.9900), • "pre-major" (x.99.99.9900), • "next" ("major" if the current version is x.99.99.9yyy, "minor" if the current version is x.y.99.9zzz, "patch" otherwise), • "patch" • "minor", • "major".
force	Create branches and tags even if they exist. Useful to recover from a previously failed attempt.

Details

plan_release():

- Ensures that no modified files are in the Git index.
- Creates a pre-release or release branch and bumps the version accordingly.
- Writes/updates cran-comments.md with useful information about the current release process.
- Runs `urlchecker::url_update()`, `devtools::check_win_devel()`, and `rhub::rhub_check(platforms = rhub::rhub_platforms())$name` in the background of the RStudio IDE, or prompts the user to do so.
- Opens a pull request for the release branch for final checks.

tag_version

Create a new version tag

Description

Parses NEWS.md and creates/updates the tag for the most recent version.

Usage

```
tag_version(force = FALSE)
```

Arguments

force Re-tag even if the last commit wasn't created by `bump_version()`. Useful when defining a CRAN release.

Value

The created tag, invisibly.

None

Examples

```
# Create mock package in a temporary directory.
# Set open to TRUE if you want to play in the mock package.
with_demo_project({
  # Use functions as if inside the newly created package project.
  # (Or go and actually run code inside the newly created package project!)
  # Add a new R file.
  usethis::use_r("cool-function", open = FALSE)
  # Pretend we added useful code inside it.
  # Track the new R file with Git.
  gert::git_add("R/cool-function.R")
  gert::git_commit("- Add cool function.")
  # Bump version with fledge.
  fledge::bump_version()
})
```

```
fledge::update_news(c("- something I forgot", "- blabla"), which = "patch")
gert::git_add("NEWS.md")
gert::git_commit(message = "release notes tweaking")
fledge::tag_version()
print(fledge::get_last_version_tag())
})
```

unbump_version

Undoes bumping the package version

Description

This undoes the effect of a `bump_version()` call, with a safety check.

Usage

```
unbump_version()
```

Value

NULL, invisibly. This function is called for its side effects.

None

See Also

`bump_version`

Examples

```
# Create mock package in a temporary directory.
# Set open to TRUE if you want to play in the mock package.
with_demo_project({
  # Use functions as if inside the newly created package project.
  # (Or go and actually run code inside the newly created package project!)
  # Add a new R file.
  usethis::use_r("cool-function", open = FALSE)
  # Pretend we added useful code inside it.
  # Track the new R file with Git.
  gert::git_add("R/cool-function.R")
  gert::git_commit("- Add cool function.")
  # Bump version with fledge.
  fledge::bump_version()
  # Oh no, we forgot to also add the awesome function for that version!
  # UNBUMP
  fledge::unbump_version()
  # Add a new R file.
  usethis::use_r("awesome-function", open = FALSE)
  # Pretend we added awesome code inside it.
  # Track the new R file with Git.
```

```

gert::git_add("R/awesome-function.R")
gert::git_commit("- Add awesome function.")
# Bump version with fledge.
fledge::bump_version()
#'
})

```

update_news

Update NEWS.md with messages from top-level commits

Description

Lists all commits from a range (default: top-level commits since the most recent tag as determined by [get_last_version_tag\(\)](#)) and adds bullets from their body to NEWS.md. Creates NEWS.md if necessary.

Usage

```

update_news(
  messages = NULL,
  which = c("auto", "samedev", "dev", "pre-patch", "patch", "pre-minor", "minor",
    "pre-major", "major")
)

```

Arguments

messages	A character vector of commit messages, e.g. as in the message column in the return value of get_top_level_commits() . The default uses the top level commits since the last tag as retrieved by get_last_version_tag() .
which	Component of the version number to update. Supported values are • "auto" (default: "samedev" or "dev", depending on contents of NEWS.md), • "samedev" (a.b.c.900x with stable version), • "dev" (a.b.c.9xxx), • "patch" (a.b.x), • "pre-minor" (a.b.99.9000), • "minor" (a.x.0), • "pre-major" (a.99.99.9000), • "major" (x.0.0).

Value

None

Examples

```
# Create mock package in a temporary directory.
# Set open to TRUE if you want to play in the mock package.
with_demo_project({
  # Use functions as if inside the newly created package project.
  # (Or go and actually run code inside the newly created package project!)
  # Add a new R file.
  usethis::use_r("cool-function", open = FALSE)
  # Pretend we added useful code inside it.
  # Track the new R file with Git.
  gert::git_add("R/cool-function.R")
  gert::git_commit("- Add cool function.")
  # Bump version with fledge.
  fledge::bump_version()
  fledge::update_news(c("- something I forgot", "- blabla"), which = "patch")
  gert::git_add("NEWS.md")
  gert::git_commit(message = "release notes tweaking")
  fledge::tag_version()
  print(fledge::get_last_version_tag())
})
```

with_demo_project	<i>Run code in temporary project</i>
-------------------	--------------------------------------

Description

Run code in temporary project

Usage

```
with_demo_project(code, dir = NULL, news = TRUE, quiet = FALSE)

local_demo_project(
  dir = NULL,
  news = TRUE,
  quiet = FALSE,
  .local_envir = parent.frame()
)
```

Arguments

code	Code to run with temporary active project
dir	Directory within which to create the mock package folder.
news	If TRUE, create a NEWS.md file.
quiet	Whether to show messages from usethis
.local_envir	The environment to use for scoping. Defaults to current execution environment.

Value

`with_demo_project()` returns the result of evaluating code.

`local_demo_project()` is called for its side effect and returns NULL, invisibly.

Examples

```
with_demo_project({  
  # Add a new R file.  
  usethis::use_r("cool-function", open = FALSE)  
  # Pretend we added useful code inside it.  
  # Track the new R file with Git.  
  gert::git_add("R/cool-function.R")  
  gert::git_commit("- Add cool function.")  
  # Bump version with fledge.  
  fledge::bump_version()  
})
```

Index

bump_version, [2](#)
bump_version(), [10](#), [11](#)
bump_version_impl (bump_version), [2](#)

commit_version, [4](#)
commit_version(), [2](#), [5](#)
create_demo_project, [4](#)

finalize_version, [5](#)
finalize_version(), [2](#)
finalize_version_impl
 (finalize_version), [5](#)

gert::git_log(), [6](#), [7](#)
get_last_tag, [6](#)
get_last_tag(), [7](#)
get_last_version_tag, [7](#)
get_last_version_tag(), [6](#), [8](#), [12](#)
get_top_level_commits, [8](#)
get_top_level_commits(), [12](#)

local_demo_project (with_demo_project),
 [13](#)

plan_release, [9](#)
post_release (plan_release), [9](#)

release (plan_release), [9](#)

tag_version, [10](#)
tag_version(), [5](#)
tibble, [8](#)

unbump_version, [11](#)
unbump_version(), [3](#)
unbump_version_impl (unbump_version), [11](#)
update_news, [12](#)
update_news(), [2](#)

with_demo_project, [13](#)