Package 'usethis'

April 29, 2020

Title Automate Package and Project Setup

Version 1.6.1

Description Automate package and project setup tasks that are otherwise
performed manually. This includes setting up unit testing, test
coverage, continuous integration, Git, 'GitHub', licenses, 'Rcpp', 'RStudio'
projects, and more.

License GPL-3

URL <https://usethis.r-lib.org>, <https://github.com/r-lib/usethis>

BugReports <https://github.com/r-lib/usethis/issues>

Depends R (>= 3.2)

Imports clipr (>= 0.3.0), cli, crayon, curl (>= 2.7), desc, fs (>=
1.3.0), gh (>= 1.1.0), git2r (>= 0.23), glue (>= 1.3.0), purrr,
rematch2, rlang (>= 0.4.3), rprojroot (>= 1.2), rstudioapi,
stats, utils, whisker, withr, yaml

Suggests covr, knitr, magick, pkgdown (>= 1.4.0), rmarkdown, roxygen2,
spelling (>= 1.2), styler (>= 1.2.0), testthat (>= 2.1.0)

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.1.0

NeedsCompilation no

Author Hadley Wickham [aut] (<https://orcid.org/0000-0003-4757-117X>),
Jennifer Bryan [aut, cre] (<https://orcid.org/0000-0002-6983-2759>),
RStudio [cph, fnd]

Maintainer Jennifer Bryan <jenny@rstudio.com>

Repository CRAN

Date/Publication 2020-04-29 05:50:02 UTC

R **topics documented:**

---

badges   *README badges*

Description

These helpers produce the markdown text you need in your README to include badges that report information, such as the CRAN version or test coverage, and link out to relevant external resources. To add badges automatically ensure your badge block starts with a line containing only <!-- badges: start --> and ends with a line containing only <!-- badges: end -->.

Usage

```
use_badge(badge_name, href, src)

use_cran_badge()

use_bioc_badge()

use_lifecycle_badge(stage)

use_binder_badge(urlpath = NULL)
```

Arguments

`badge_name`	Badge name. Used in error message and alt text
`href`, `src`	Badge link and image src
`stage`	Stage of the package lifecycle
`urlpath`	An optional `urlpath` component to add to the link, e.g. `"rstudio"` to open an RStudio IDE instead of a Jupyter notebook. See the binder documentation for additional examples.

Details

- `use_badge()`: a general helper used in all badge functions

- `use_bioc_badge()`: badge indicates BioConductor build status

- `use_cran_badge()`: badge indicates what version of your package is available on CRAN, powered by `https://www.r-pkg.org`

- `use_lifecycle_badge()`: badge declares the developmental stage of a package, according to `https://www.tidyverse.org/lifecycle/`:

  – Experimental
  – Maturing
  – Stable
  – Superseded
  – Archived
  – Dormant
  – Questioning

- `use_binder_badge()`: badge indicates that your repository can be launched in an executable environment on `https://mybinder.org/`

See Also

The functions that set up continuous integration services also create badges.

Examples

```
## Not run:
use_cran_badge()
use_lifecycle_badge("stable")

## End(Not run)
```

browse-this	*Quickly browse to important package webpages*

Description

These functions take you to various webpages associated with a package and return the target URL invisibly. Some URLs are formed from first principles and there is no guarantee there will be content at the destination.

Usage

```
browse_github(package = NULL)

browse_github_issues(package = NULL, number = NULL)

browse_github_pulls(package = NULL, number = NULL)

browse_travis(package = NULL, ext = c("com", "org"))

browse_circleci(package = NULL)

browse_cran(package = NULL)
```

Arguments

package	Name of package; leave as NULL to use current package
number	For GitHub issues and pull requests. Can be a number or "new".
ext	Version of travis to use.

Details

- `browse_github()`: Looks for a GitHub URL in the URL field of DESCRIPTION.
- `browse_github_issues()`: Visits the GitHub Issues index or one specific issue.
- `browse_github_pulls()`: Visits the GitHub Pull Request index or one specific pull request.
- `browse_travis()`: Visits the package's page on Travis CI.
- `browse_circleci()`: Visits the package's page on Circle CI
- `browse_cran()`: Visits the package on CRAN, via the canonical URL.

Examples

```
browse_github("gh")
browse_github_issues("backports")
browse_github_issues("backports", 1)
browse_github_pulls("rprojroot")
browse_github_pulls("rprojroot", 3)
browse_travis("usethis")
browse_cran("MASS")
```

---

browse_github_token   *Create and retrieve a GitHub personal access token*

Description

A personal access token (PAT) is needed for git operations via the GitHub API. Two helper functions
are provided:

- `browse_github_token()` is synonymous with `browse_github_pat()`: Both open a browser
  window to the GitHub form to generate a PAT. See below for advice on how to store this.

- `github_token()` retrieves a stored PAT by consulting, in this order:

    - `GITHUB_PAT` environment variable
    - `GITHUB_TOKEN` environment variable
    - the empty string `""`

Usage

```
browse_github_token(
  scopes = c("repo", "gist", "user:email"),
  description = "R:GITHUB_PAT",
  host = "https://github.com"
)

browse_github_pat(
  scopes = c("repo", "gist", "user:email"),
  description = "R:GITHUB_PAT",
  host = "https://github.com"
)

github_token()
```

Arguments

scopes	Character vector of token scopes, pre-selected in the web form. Final choices are made in the GitHub form. Read more about GitHub API scopes at `https://developer.github.com/apps/building-oauth-apps/scopes-for-oauth-apps/`.
description	Short description or nickname for the token. It helps you distinguish various tokens on GitHub.
host	GitHub API host to use. Override with the endpoint-root for your GitHub enterprise instance, for example, "https://github.hostname.com/api/v3".

Value

`github_token()` returns a string, a GitHub PAT or `""`.

Get and store a PAT: Sign up for a free GitHub.com account and sign in. Call `browse_github_token()`. Verify the scopes and click "Generate token". Copy the token right away! A common approach is to store in `.Renviron` as the GITHUB_PAT environment variable. `edit_r_environ()` opens this file for editing.

See Also

`gh::gh_whoami()` for information on an existing token.

Examples

```
## Not run:
browse_github_token()
## COPY THE PAT!!!
## almost certainly to be followed by ...
edit_r_environ()
## which helps you store the PAT as an env var

## End(Not run)
# for safety's sake, just reveal first 4 characters
substr(github_token(), 1, 4)
```

---

ci   *Continuous integration setup and badges*

Description

Sets up continuous integration (CI) services for an R package that is developed on GitHub. CI services can run R CMD check automatically on various platforms, triggered by each push or pull request. These functions

- Add service-specific configuration files and add them to `.Rbuildignore`.
- Activate a service or give the user a detailed prompt.
- Provide the markdown to insert a badge into README.

Usage

```
use_travis(browse = rlang::is_interactive(), ext = c("com", "org"))

use_travis_badge(ext = c("com", "org"))

use_appveyor(browse = rlang::is_interactive())

use_appveyor_badge()

use_gitlab_ci()

use_circleci(browse = rlang::is_interactive(), image = "rocker/verse:latest")

use_circleci_badge()
```

Arguments

browse	Open a browser window to enable automatic builds for the package.
ext	Which travis website to use. Defaults to "com" for https://travis-ci.com. Change to "org" for https://travis-ci.org.
image	The Docker image to use for build. Must be available on DockerHub. The rocker/verse image includes TeX Live, pandoc, and the tidyverse packages. For a minimal image, try rocker/r-ver. To specify a version of R, change the tag from latest to the version you want, e.g. rocker/r-ver:3.5.3.

use_travis()

Adds a basic .travis.yml to the top-level directory of a package. This is a configuration file for the Travis CI continuous integration service.

use_travis_badge()

Only adds the Travis CI badge. Use for a project where Travis is already configured.

use_appveyor()

Adds a basic appveyor.yml to the top-level directory of a package. This is a configuration file for the AppVeyor continuous integration service for Windows.

use_appveyor_badge()

Only adds the AppVeyor badge. Use for a project where AppVeyor is already configured.

use_gitlab_ci()

Adds a basic .gitlab-ci.yml to the top-level directory of a package. This is a configuration file for the GitLab CI/CD continuous integration service for GitLab.

use_circleci()

Adds a basic .circleci/config.yml to the top-level directory of a package. This is a configuration file for the CircleCI continuous integration service.

use_circleci_badge()

Only adds the Circle CI badge. Use for a project where Circle CI is already configured.

---

create_from_github   *Create a project from a GitHub repo*

Description

Creates a new local Git repository from a repository on GitHub. It is highly recommended that you pre-configure or pass a GitHub personal access token (PAT), which is facilitated by browse_github_token(). In particular, a PAT is required in order for create_from_github() to do "fork and clone". It is also required by use_github(), which connects existing local projects to GitHub.

Usage

```
create_from_github(
  repo_spec,
  destdir = NULL,
  fork = NA,
  rstudio = NULL,
  open = rlang::is_interactive(),
  protocol = git_protocol(),
  credentials = NULL,
  auth_token = github_token(),
  host = NULL
)
```

Arguments

repo_spec   GitHub repo specification in this form: owner/repo. The repo part will be the name of the new local repo.

destdir   The new folder is stored here. If NULL, defaults to user's Desktop or some other conspicuous place. You can also set a default location using the option usethis.destdir, e.g. options(usethis.destdir = "a/good/dir"), perhaps saved to your .Rprofile with edit_r_profile()

fork   If TRUE, we create and clone a fork. If FALSE, we clone repo_spec itself. Will be set to FALSE if no auth_token (a.k.a. PAT) is provided or preconfigured. Otherwise, defaults to FALSE if you can push to repo_spec and TRUE if you cannot. In the case of a fork, the original target repo is added to the local repo as the upstream remote, using the preferred protocol. The master branch is set to track upstream/master and is immediately pulled, which matters in the case of a pre-existing, out-of-date fork.

rstudio   Initiate an RStudio Project? Defaults to TRUE if in an RStudio session and project has no pre-existing .Rproj file. Defaults to FALSE otherwise.

open   If TRUE, activates the new project:

                • If RStudio desktop, the package is opened in a new session.
                • If on RStudio server, the current RStudio project is activated.
                • Otherwise, the working directory and active project is changed.

protocol	Optional. Should be "ssh" or "https", if specified. Defaults to the option usethis.protocol and, if unset, to an interactive choice or, in non-interactive sessions, "ssh". NA triggers the interactive menu.
credentials	A git2r credential object produced with git2r::cred_env(), git2r::cred_ssh_key(), git2r::cred_token(), or git2r::cred_user_pass().
auth_token	GitHub personal access token (PAT).
host	GitHub API host to use. Override with the endpoint-root for your GitHub enterprise instance, for example, "https://github.hostname.com/api/v3".

Using SSH Keys on Windows

If you are a Windows user who connects to GitHub using SSH, as opposed to HTTPS, you may need to explicitly specify the paths to your keys and register this credential in the current R session. This helps if git2r, which usethis uses for Git operations, does not automatically find your keys or handle your passphrase.

In the snippet below, do whatever is necessary to make the paths correct, e.g., replace <USERNAME> with your Windows username. Omit the passphrase part if you don't have one. Replace <OWNER/REPO> with the appropriate GitHub specification. You get the idea.

```
creds <- git2r::cred_ssh_key(
  publickey  = "C:/Users/<USERNAME>/.ssh/id_rsa.pub",
  privatekey = "C:/Users/<USERNAME>/.ssh/id_rsa",
  passphrase = character(0)
)
use_git_protocol("ssh")
use_git_credentials(credentials = creds)

create_from_github(
  repo_spec = "<OWNER/REPO>",
  ...
)
```

See Also

use_github() for GitHub setup advice. git_protocol() and git_credentials() for background on protocol and credentials. use_course() for one-time download of all files in a Git repo, without any local or remote Git operations.

Examples

```
## Not run:
create_from_github("r-lib/usethis")

## End(Not run)
```

---

create_package *Create a package or project*

Description

These functions create an R project:

- create_package() creates an R package
- create_project() creates a non-package project, i.e. a data analysis project

Both functions can be called on an existing project; you will be asked before any existing files are changed.

Usage

```
create_package(
  path,
  fields = list(),
  rstudio = rstudioapi::isAvailable(),
  roxygen = TRUE,
  check_name = TRUE,
  open = rlang::is_interactive()
)

create_project(
  path,
  rstudio = rstudioapi::isAvailable(),
  open = rlang::is_interactive()
)
```

Arguments

path   A path. If it exists, it is used. If it does not exist, it is created, provided that the parent path exists.

fields   A named list of fields to add to DESCRIPTION, potentially overriding default values. See use_description() for how you can set personalized defaults using package options

rstudio   If TRUE, calls use_rstudio() to make the new package or project into an RStudio Project. If FALSE and a non-package project, a sentinel .here file is placed so that the directory can be recognized as a project by the here or rprojroot packages.

roxygen   Do you plan to use roxygen2 to document your package?

check_name   Whether to check if the name is valid for CRAN and throw an error if not

open   If TRUE, activates the new project:

- If RStudio desktop, the package is opened in a new session.
- If on RStudio server, the current RStudio project is activated.
- Otherwise, the working directory and active project is changed.

Value

Path to the newly created project or package, invisibly.

See Also

create_tidy_package() is a convenience function that extends create_package() by immediately applying as many of the tidyverse development conventions as possible.

---

edit   *Open configuration files*

Description

- edit_r_profile() opens .Rprofile

- edit_r_environ() opens .Renviron

- edit_r_makevars() opens .R/Makevars

- edit_git_config() opens .gitconfig or .git/config

- edit_git_ignore() opens .gitignore

- edit_rstudio_snippets(type) opens .R/snippets/{type}.snippets

Usage

```
edit_r_profile(scope = c("user", "project"))

edit_r_environ(scope = c("user", "project"))

edit_r_buildignore(scope = c("user", "project"))

edit_r_makevars(scope = c("user", "project"))

edit_rstudio_snippets(
  type = c("r", "markdown", "c_cpp", "css", "html", "java", "javascript", "python",
    "sql", "stan", "tex")
)

edit_git_config(scope = c("user", "project"))

edit_git_ignore(scope = c("user", "project"))
```

Arguments

scope   Edit globally for the current **user**, or locally for the current **project**

type   Snippet type (case insensitive text).

Details

The edit_r_*() functions and `edit_rstudio_snippets()` consult R's notion of user's home directory. The edit_git_*() functions – and **usethis** in general – inherit home directory behaviour from the **fs** package, which differs from R itself on Windows. The **fs** default is more conventional in terms of the location of user-level Git config files. See `fs::path_home()` for more details.

Files created by `edit_rstudio_snippets()` will *mask*, not supplement, the built-in default snippets. If you like the built-in snippets, copy them and include with your custom snippets.

Value

Path to the file, invisibly.

---

github_actions   *GitHub Actions setup and badges*

Description

Sets up continuous integration (CI) for an R package that is developed on GitHub using GitHub Actions. These functions

- Add the necessary configuration files and place them in `.Rbuildignore`.
- Provide the markdown to insert a badge into your README

Usage

```
use_github_actions()

use_github_actions_badge(name = "R-CMD-check")
```

Arguments

name   The name to give to the GitHub Actions workflow.

`use_github_actions()`

Adds a basic `R-CMD-check.yaml` file to the `.github/workflows` directory of a package. This is a configuration file for the GitHub Actions service.

`use_github_actions_badge()`

Only adds the GitHub Actions badge. Use for a project where GitHub Actions is already configured.

See Also

`use_github_action()` for setting up a specific action.

`git_credentials` *Produce or register git credentials*

Description

Credentials are needed for git operations like git push that address a remote, typically GitHub. usethis uses the git2r package. git2r tries to use the same credentials as command line git, but sometimes fails. usethis tries to increase the chance that things "just work" and, when they don't, to provide the user a way to intervene:

- `git_credentials()` returns any `credentials` that have been registered with `use_git_credentials()` and, otherwise, implements usethis's default strategy.
- `use_git_credentials()` allows you to register `credentials` explicitly for use in all usethis functions in an R session. Do this only after proven failure of the defaults.

Usage

```
git_credentials(protocol = git_protocol(), auth_token = github_token())

use_git_credentials(credentials)
```

Arguments

`protocol`	Optional. Should be "ssh" or "https", if specified. Defaults to the option `usethis.protocol` and, if unset, to an interactive choice or, in non-interactive sessions, "ssh". `NA` triggers the interactive menu.
`auth_token`	GitHub personal access token (PAT).
`credentials`	A git2r credential object produced with [git2r::cred_env()](), [git2r::cred_ssh_key()](), [git2r::cred_token()](), or [git2r::cred_user_pass()]().

Value

Either `NULL` or a git2r credential object, invisibly, i.e. something to be passed to git2r as `credentials`.

Default credentials

If the default behaviour of usethis + git2r works, rejoice and leave well enough alone. Keep reading if you need more control or understanding.

SSH credentials

For `protocol = "ssh"`, by default, usethis passes `NULL` credentials to git2r. This will work if you have the exact configuration expected by git2r:

1. Your public and private keys are in the default locations, ~/.ssh/id_rsa.pub and ~/.ssh/id_rsa, respectively.

2. All the relevant software agrees on the definition of ~/, i.e. your home directory. This is harder than it sounds on Windows.

3. Your `ssh-agent` is configured to manage your SSH passphrase, if you have one. This too can be a problem on Windows. Read more about SSH setup in Happy Git and GitHub for the useR, especially the troubleshooting section.

If the `NULL` default doesn't work, you can make `credentials` explicitly with `git2r::cred_ssh_key()` and register that with use_git_credentials() for the rest of the session:

```
my_cred <- git2r::cred_ssh_key(
   publickey  = "path/to/your/id_rsa.pub",
   privatekey = "path/to/your/id_rsa",
   # include / omit passphrase as appropriate to your situation
   passphrase = askpass::askpass()
)
use_git_credentials(credentials = my_cred)
```

For the remainder of the session, `git_credentials()` will return `my_cred`.

HTTPS credentials

For `protocol = "https"`, we must send username and password. It is possible that your OS has cached this and git2r will successfully use that. However, usethis can offer even more chance of success in the HTTPS case. GitHub also accepts a personal access token (PAT) via HTTPS. If `credentials = NULL` and a PAT is available, we send it. Preference is given to any `auth_token` that is passed explicitly. Otherwise, `github_token()` is called. If a PAT is found, we make an HTTPS credential with `git2r::cred_user_pass()`. The PAT is sent as the password and dummy text is sent as the username (the PAT is what really matters in this case). You can also register an explicit credential yourself in a similar way:

```
my_cred <- git2r::cred_user_pass(
  username = "janedoe",
  password = askpass::askpass()
)
use_git_credentials(credentials = my_cred)
```

For the remainder of the session, `git_credentials()` will return `my_cred`.

Examples

```
git_credentials()
git_credentials(protocol = "ssh")

## Not run:
# these calls look for a GitHub PAT
git_credentials(protocol = "https")
git_credentials(protocol = "https", auth_token = "MY_GITHUB_PAT")

## End(Not run)
```

git_protocol	*Produce or register git protocol*

Description

Git operations that address a remote use a so-called "transport protocol". usethis supports SSH and HTTPS. The protocol affects two things:

- The default URL format for repos with no existing remote protocol:
  - protocol = "ssh" implies git@github.com:<OWNER>/<REPO>.git
  - protocol = "https" implies https://github.com/<OWNER>/<REPO>.git
- The strategy for creating credentials when none are given. See [git_credentials()](#) for details. Two helper functions are available:
- git_protocol() returns the user's preferred protocol, if known, and, otherwise, asks the user (interactive session) or defaults to SSH (non-interactive session).
- use_git_protocol() allows the user to set the git protocol, which is stored in the usethis.protocol option. Any interactive choice re: protocol comes with a reminder of how to set the protocol at startup by setting an option in .Rprofile:

```
options(usethis.protocol = "ssh")
## or
options(usethis.protocol = "https")
```

Usage

```
git_protocol()

use_git_protocol(protocol)
```

Arguments

protocol   Optional. Should be "ssh" or "https", if specified. Defaults to the option usethis.protocol and, if unset, to an interactive choice or, in non-interactive sessions, "ssh". NA triggers the interactive menu.

Value

"ssh" or "https"

Examples

```
## Not run:
## consult the option and maybe get an interactive menu
git_protocol()

## explicitly set the protocol
use_git_protocol("ssh")
```

```
use_git_protocol("https")

## End(Not run)
```

---

git_sitrep   *git/GitHub sitrep*

Description

Get a situation report on your current git/GitHub status. Useful for diagnosing problems. [git_vaccinate()](git_vaccinate) adds some basic R- and RStudio-related entries to the user-level git ignore file.

Usage

```
git_sitrep()
```

Examples

```
git_sitrep()
```

---

git_vaccinate   *Vaccinate your global git ignore*

Description

Adds .DS_Store, .Rproj.user, and .Rhistory to your global .gitignore. This is good practices as it ensures that you will never accidentally leak credentials to GitHub.

Usage

```
git_vaccinate()
```

`licenses`   *License a package*

Description

Adds the necessary infrastructure to declare your package as licensed with one of these popular open source licenses:

- CC0: dedicated to public domain. Appropriate for data packages.
- MIT: simple and permissive.
- Apache 2.0: provides patent protection.
- GPL v3: requires sharing of improvements.
- AGPL v3: requires sharing of improvements.
- LGPL v3: requires sharing of improvements.
- CCBY 4.0: Free to share and adapt, must give appropriate credit. Appropriate for data packages.

See https://choosealicense.com for more details and other options.

Usage

```
use_mit_license(name = find_name())

use_gpl3_license(name = find_name())

use_agpl3_license(name = find_name())

use_lgpl_license(name = find_name())

use_apl2_license(name = find_name())

use_cc0_license(name = find_name())

use_ccby_license(name = find_name())
```

Arguments

name   Name of the copyright holder or holders. Separate multiple individuals with
                    ;. You can supply a global default with `options(usethis.full_name = "My`
                    `name")`.

Details

CRAN does not allow you to include copies of standard licenses in your package, so these functions save the license as `LICENSE.md` and add it to `.Rbuildignore`.

See Also

The license section of R Packages.

---

proj_activate	*Activate a project*

Description

Activates a project in usethis, R session, and (if relevant) RStudio senses. If you are in RStudio, this will open a new RStudio session. If not, it will change the working directory and active project.

Usage

```
proj_activate(path)
```

Arguments

path   Project directory

Value

Single logical value indicating if current session is modified.

---

proj_sitrep	*Report working directory and usethis/RStudio project*

Description

proj_sitrep() reports

- current working directory
- the active usethis project
- the active RStudio Project

Call this function if things seem weird and you're not sure what's wrong or how to fix it. Usually, all three of these should coincide (or be unset) and proj_sitrep() provides suggested commands for getting back to this happy state.

Usage

```
proj_sitrep()
```

Value

A named list, with S3 class sitrep (for printing purposes), reporting current working directory, active usethis project, and active RStudio Project

See Also

Other project functions: `proj_utils`

Examples

```
proj_sitrep()
```

---

proj_utils   *Utility functions for the active project*

Description

Most use_*() functions act on the **active project**. If it is unset, usethis uses rprojroot to find the project root of the current working directory. It establishes the project root by looking for a `.here` file, an RStudio Project, a package `DESCRIPTION`, Git infrastructure, a `remake.yml` file, or a `.projectile` file. It then stores the active project for use for the remainder of the session.

Usage

```
proj_get()

proj_set(path = ".", force = FALSE)

proj_path(..., ext = "")

with_project(
  path = ".",
  code,
  force = FALSE,
  quiet = getOption("usethis.quiet", default = FALSE)
)

local_project(
  path = ".",
  force = FALSE,
  quiet = getOption("usethis.quiet", default = FALSE),
  .local_envir = parent.frame()
)
```

Arguments

path   Path to set. This `path` should exist or be `NULL`.

force   If `TRUE`, use this path without checking the usual criteria for a project. Use
            sparingly! The main application is to solve a temporary chicken-egg problem:
            you need to set the active project in order to add project-signalling infrastructure,
            such as initialising a Git repo or adding a `DESCRIPTION` file.

...	character vectors, if any values are NA, the result will also be NA. The paths follow the recycling rules used in the tibble package, namely that only length 1 arguments are recycled.
ext	An optional extension to append to the generated path.
code	Code to run with temporary active project.
quiet	Whether to suppress user-facing messages, while operating in the temporary active project.
.local_envir	The environment to use for scoping. Defaults to current execution environment.

Details

In general, end user scripts should not contain direct calls to usethis::proj_*() utility functions. They are internal functions that are exported for occasional interactive use or use in packages that extend usethis. End user code should call functions in rprojroot or its simpler companion, here, to programmatically detect a project and build paths within it.

Functions

- `proj_get`: Retrieves the active project and, if necessary, attempts to set it in the first place.

- `proj_set`: Sets the active project.

- `proj_path`: Builds a path within the active project returned by `proj_get()`. Thin wrapper around `fs::path()`.

- `with_project`: Runs code with a temporary active project. It is an example of the with_*() functions in withr.

- `local_project`: Sets an active project until the current execution environment goes out of scope, e.g. the end of the current function or test. It is an example of the local_*() functions in withr.

See Also

Other project functions: `proj_sitrep()`

Examples

```
## Not run:
## see the active project
proj_get()

## manually set the active project
proj_set("path/to/target/project")

## build a path within the active project (both produce same result)
proj_path("R/foo.R")
proj_path("R", "foo", ext = "R")

## build a path within SOME OTHER project
with_project("path/to/some/other/project", proj_path("blah.R"))
```

```
## convince yourself that with_project() temporarily changes the project
with_project("path/to/some/other/project", print(proj_sitrep()))

## End(Not run)
```

---

pr_init   *Helpers for GitHub pull requests*

Description

The pr_* family of functions is designed to make working with GitHub PRs as painless as possible for both contributors and package maintainers. They are designed to support the git and GitHub best practices described in Happy Git and GitHub for the useR.

Usage

```
pr_init(branch)

pr_fetch(number, owner = NULL)

pr_push()

pr_pull()

pr_pull_upstream()

pr_sync()

pr_view()

pr_pause()

pr_finish(number = NULL)
```

Arguments

branch	branch name. Should usually consist of lower case letters, numbers, and -.
number	Number of PR to fetch.
owner	Name of the owner of the repository that is the target of the pull request. Default of NULL tries to identify the source repo and uses the owner of the upstream remote, if present, or the owner of origin otherwise.

Set up advice

These functions make heavy use of git2r and the GitHub API. You'll need a GitHub personal access token (PAT); see browse_github_token() for help with that. If git2r does not seem to be finding your git credentials, read git_credentials() for troubleshooting advice. The transport protocol (SSH vs HTTPS) is determined from the existing remote URL(s) of the repo.

For contributors

To contribute to a package, first use `create_from_github("OWNER/REPO",fork = TRUE)` to fork the source repository, and then check out a local copy. Next use `pr_init()` to create a branch for your PR (**never** submit a PR from the `master` branch). You'll then work locally, making changes to files and checking them into git. Once you're ready to submit, run `pr_push()` to push your local branch to GitHub, and open a webpage that lets you initiate the PR (or draft PR). To learn more about the process of making a pull request, read the Pull Request Helpers vignette.

If you are lucky, your PR will be perfect, and the maintainer will accept it. You can then run `pr_finish()` to close and delete your PR branch. In most cases, however, the maintainer will ask you to make some changes. Make the changes, then run `pr_push()` to sync back up to GitHub.

It's also possible that the maintainer will contribute some code to your PR: you get that code back to your computer, run `pr_pull()`. It's also possible that other changes have occurred to the package while you've been working on your PR, and you need to "merge master". Do that by running `pr_pull_upstream()`: this makes sure that your copy of the package is up-to-date with the maintainer's latest changes. Either of the pull functions may cause merge conflicts, so be prepared to resolve before continuing.

For maintainers

To download a PR locally so that you can experiment with it, run pr_fetch(<pr_number>). If you make changes, run `pr_push()` to push them back to GitHub. After you have merged the PR, run `pr_finish()` to delete the local branch.

Other helpful functions

- `pr_sync()` is a shortcut for `pr_pull()`, `pr_pull_upstream()`, and `pr_push()`

- `pr_pause()` makes sure you're synced with the PR and then switches back to master.

- `pr_view()` opens the PR in the browser

Examples

```
## Not run:
## scenario: current project is a local copy of fork of a repo owned by
## 'tidyverse', not you
pr_fetch(123, owner = "tidyverse")

## End(Not run)
```

---

rename_files   *Automatically rename paired* R/ *and* test/ *files*

Description

- Moves R/{old}.R to R/{new}.R

- Moves tests/testthat/test-{old}.R to tests/testthat/test-{new}.R

- Moves tests/testthat/test-{old}-*.* to tests/testthat/test-{new}-*.* and updates paths in the test file.

- Removes context() calls from the test file, which are unnecessary (and discouraged) as of testthat v2.1.0.

This is a potentially dangerous operation, so you must be using Git in order to use this function.

Usage

```
rename_files(old, new)
```

Arguments

old, new   Old and new file names (with or without extensions).

---

rprofile-helper   *Helpers to make useful changes to* .Rprofile

---

Description

All functions open your .Rprofile and give you the code you need to paste in.

- use_devtools(): makes devtools available in interactive sessions.

- use_usethis(): makes usethis available in interactive sessions.

- use_reprex(): makes reprex available in interactive sessions.

- use_conflicted(): makes conflicted available in interactive sessions.

- use_partial_warning(): warns on partial matches.

Usage

```
use_conflicted()

use_reprex()

use_usethis()

use_devtools()

use_partial_warnings()
```

---

use_addin	*Add minimal RStudio Addin binding*

Description

This function helps you add a minimal RStudio Addin binding to inst/rstudio/addins.dcf.

Usage

```
use_addin(addin = "new_addin", open = rlang::is_interactive())
```

Arguments

addin   Name of the addin function, which should be defined in the R folder.

open   Open the newly created file for editing? Happens in RStudio, if applicable, or via utils::file.edit() otherwise.

---

use_blank_slate	*Don't save/load user workspace between sessions*

Description

R can save and reload the user's workspace between sessions via an .RData file in the current directory. However, long-term reproducibility is enhanced when you turn this feature off and clear R's memory at every restart. Starting with a blank slate provides timely feedback that encourages the development of scripts that are complete and self-contained. More detail can be found in the blog post Project-oriented workflow.

Usage

```
use_blank_slate(scope = c("user", "project"))
```

Arguments

scope   Edit globally for the current **user**, or locally for the current **project**

Details

Only use_blank_slate("project") is automated so far, since RStudio currently only supports modification of user-level or global options via the user interface.

---

use_build_ignore   *Add files to* `.Rbuildignore`

Description

`.Rbuildignore` has a regular expression on each line, but it's usually easier to work with specific file names. By default, `use_build_ignore()` will (crudely) turn a filename into a regular expression that will only match that path. Repeated entries will be silently removed.

`use_build_ignore()` is designed to ignore *individual* files. If you want to ignore *all* files with a given extension, consider providing an "as-is" regular expression, using `escape = FALSE`; see examples.

Usage

```
use_build_ignore(files, escape = TRUE)
```

Arguments

files   Character vector of path names.

escape   If TRUE, the default, will escape . to \\. and surround with ^ and $.

Examples

```
## Not run:
# ignore all Excel files
use_build_ignore("[.]xlsx$", escape = FALSE)

## End(Not run)
```

---

use_citation   *Create a CITATION template*

Description

Use this if you want to encourage users of your package to cite an article or book.

Usage

```
use_citation()
```

use_code_of_conduct	*Add a code of conduct*

Description

Adds a `CODE_OF_CONDUCT.md` file to the active project and lists in `.Rbuildignore`, in the case of a package. The goal of a code of conduct is to foster an environment of inclusiveness, and to explicitly discourage inappropriate behaviour. The template comes from https://contributor-covenant.org, version 2: https://contributor-covenant.org/version/2/0.

Usage

```
use_code_of_conduct(path = NULL)
```

Arguments

path	Path of the directory to put `CODE_OF_CONDUCT.md` in, relative to the active project. Passed along to [use_directory()](). Default is to locate at top-level, but .github/ is also common.

Details

If your package is going to CRAN, the link to the CoC in your README must be an absolute link to a rendered website as `CODE_OF_CONDUCT.md` is not included in the package sent to CRAN. use_code_of_conduct() will automatically generate this link if you use pkgdown and have set the `url` field in `pkgdown.yml`; otherwise it'll link to https://contributor-covenant.org/version/2/0.

use_coverage	*Test coverage*

Description

use_coverage() Adds test coverage reports to a package.

Usage

```
use_coverage(type = c("codecov", "coveralls"))

use_covr_ignore(files)
```

Arguments

type	Which web service to use for test reporting. Currently supports [Codecov]() and [Coveralls]().
files	Character vector of file globs.

---

use_cran_comments   *CRAN submission comments*

Description

Creates `cran-comments.md`, a template for your communications with CRAN when submitting a package. The goal is to clearly communicate the steps you have taken to check your package on a wide range of operating systems. If you are submitting an update to a package that is used by other packages, you also need to summarize the results of your reverse dependency checks.

Usage

```
use_cran_comments(open = rlang::is_interactive())
```

Arguments

open   Open the newly created file for editing? Happens in RStudio, if applicable, or
                via `utils::file.edit()` otherwise.

---

use_data   *Create package data*

Description

`use_data()` makes it easy to save package data in the correct format. I recommend you save scripts that generate package data in `data-raw`: use `use_data_raw()` to set it up. You also need to document exported datasets.

Usage

```
use_data(
  ...,
  internal = FALSE,
  overwrite = FALSE,
  compress = "bzip2",
  version = 2
)

use_data_raw(name = "DATASET", open = rlang::is_interactive())
```

Arguments

`...`	Unquoted names of existing objects to save.
`internal`	If `FALSE`, saves each object in its own `.rda` file in the data/ directory. These data files bypass the usual export mechanism and are available whenever the package is loaded (or via [data()](#) if LazyData is not true).
	If `TRUE`, stores all objects in a single R/sysdata.rda file. Objects in this file follow the usual export rules. Note that this means they will be exported if you are using the common exportPattern() rule which exports all objects except for those that start with `.`.
`overwrite`	By default, use_data() will not overwrite existing files. If you really want to do so, set this to `TRUE`.
`compress`	Choose the type of compression used by [save()](#). Should be one of "gzip", "bzip2", or "xz".
`version`	The serialization format version to use. The default, 2, was the default format from R 1.4.0 to 3.5.3. Version 3 became the default from R 3.6.0 and can only be read by R versions 3.5.0 and higher.
`name`	Name of the dataset to be prepared for inclusion in the package.
`open`	Open the newly created file for editing? Happens in RStudio, if applicable, or via [utils::file.edit()](#) otherwise.

See Also

The [data chapter](#) of [R Packages](#).

Examples

```
## Not run:
x <- 1:10
y <- 1:100

use_data(x, y) # For external use
use_data(x, y, internal = TRUE) # For internal use

## End(Not run)
## Not run:
use_data_raw("daisy")

## End(Not run)
```

---

use_data_table	*Prepare for importing data.table*

Description

use_data_table facilitates importing data.table by handling up-front some common set-up tasks for using it in your package.

This function does two main things:

1. Import the entire data.table namespace (with @import).

2. Block the usage of data.table as a dependency (DESCRIPTION field Depends); data.table should be used as an *import* or *suggested* package only. See this discussion.

Usage

```
use_data_table()
```

---

use_description   *Create or modify a DESCRIPTION file*

Description

use_description() creates a DESCRIPTION file. Although mostly associated with R packages, a DESCRIPTION file can also be used to declare dependencies for a non-package projects. Within such a project, devtools::install_deps() can then be used to install all the required packages. Note that, by default, use_decription() checks for a CRAN-compliant package name. You can turn this off with check_name = FALSE.

usethis consults the following sources, in this order, to set DESCRIPTION fields:

- fields argument of create_package() or use_description().

- getOption("usethis.description") or getOption("devtools.desc"). The devtools option is consulted only for backwards compatibility and it's recommended to switch to an option named "usethis.description".

- Defaults built into usethis.

The fields discovered via options or the usethis package can be viewed with use_description_defaults().

If you create a lot of packages, consider storing personalized defaults as a named list in an option named "usethis.description". Here's an example of code to include in .Rprofile:

```
options(
  usethis.description = list(
   `Authors@R` = 'person("Jane", "Doe", email = "jane@example.com", role = c("aut", "cre"),
                          comment = c(ORCID = "YOUR-ORCID-ID"))',
    License = "MIT + file LICENSE",
    Language =  "es"
  )
)
```

Usage

```
use_description(fields = list(), check_name = TRUE, roxygen = TRUE)

use_description_defaults(package = NULL, roxygen = TRUE, fields = list())
```

Arguments

`fields`	A named list of fields to add to `DESCRIPTION`, potentially overriding default values. See `use_description()` for how you can set personalized defaults using package options
`check_name`	Whether to check if the name is valid for CRAN and throw an error if not
`roxygen`	If TRUE, sets RoxygenNote to current roxygen2 version.
`package`	Package name

See Also

The description chapter of R Packages.

Examples

```
## Not run:
use_description()

use_description(fields = list(Language = "es"))

use_description_defaults()

## End(Not run)
```

---

use_directory   *Use a directory*

Description

`use_directory()` creates a directory (if it does not already exist) in the project's top-level directory. This function powers many of the other use_ functions such as `use_data()` and `use_vignette()`.

Usage

```
use_directory(path, ignore = FALSE)
```

Arguments

`path`	Path of the directory to create, relative to the project.
`ignore`	Should the newly created file be added to `.Rbuildignore`?

Examples

```
## Not run:
use_directory("inst")

## End(Not run)
```

---

use_git   *Initialise a git repository*

Description

use_git() initialises a Git repository and adds important files to .gitignore. If user consents, it also makes an initial commit.

Usage

```
use_git(message = "Initial commit")
```

Arguments

message   Message to use for first commit.

See Also

Other git helpers: use_git_config(), use_git_hook(), use_git_ignore()

Examples

```
## Not run:
use_git()

## End(Not run)
```

---

use_github   *Connect a local repo with GitHub*

Description

use_github() takes a local project, creates an associated repo on GitHub, adds it to your local repo as the origin remote, and makes an initial push to synchronize. use_github() requires that your project already be a Git repository, which you can accomplish with use_git(), if needed. See the Authentication section below for other necessary setup.

Usage

```
use_github(
  organisation = NULL,
  private = FALSE,
  protocol = git_protocol(),
  credentials = NULL,
  auth_token = github_token(),
  host = NULL
)
```

Arguments

organisation   If supplied, the repo will be created under this organisation, instead of the ac-
                count of the user associated with the `auth_token`. You must have permission to
                create repositories.

private   If TRUE, creates a private repository.

protocol   Optional. Should be "ssh" or "https", if specified. Defaults to the option `usethis.protocol`
                and, if unset, to an interactive choice or, in non-interactive sessions, "ssh". NA
                triggers the interactive menu.

credentials   A git2r credential object produced with `git2r::cred_env()`, `git2r::cred_ssh_key()`,
                `git2r::cred_token()`, or `git2r::cred_user_pass()`.

auth_token   GitHub personal access token (PAT).

host   GitHub API host to use. Override with the endpoint-root for your GitHub enter-
                prise instance, for example, "https://github.hostname.com/api/v3".

Authentication

A new GitHub repo will be created via the GitHub API, therefore you must make a GitHub personal
access token (PAT) available. You can either provide this directly via the `auth_token` argument or
store it for retrieval with `github_token()`.

Examples

```
## Not run:
pkgpath <- file.path(tempdir(), "testpkg")
create_package(pkgpath) # creates package below temp directory
proj_set(pkgpath)

## now, working inside "testpkg", initialize git repository
use_git()

## create github repository and configure as git remote
use_github()

## End(Not run)
```

use_github_action   *Use a specific GitHub action*

Description

Use a specific action, either one of the example actions from r-lib/actions/examples or a custom
action given by the `url` parameter.

Usage

```
use_github_action(
  name,
  url = NULL,
  save_as = NULL,
  ignore = TRUE,
  open = FALSE
)

use_github_action_check_release(
  save_as = "R-CMD-check.yaml",
  ignore = TRUE,
  open = FALSE
)

use_github_action_check_standard(
  save_as = "R-CMD-check.yaml",
  ignore = TRUE,
  open = FALSE
)

use_github_action_check_full(
  save_as = "R-CMD-check.yaml",
  ignore = TRUE,
  open = FALSE
)

use_github_action_pr_commands(
  save_as = "pr-commands.yaml",
  ignore = TRUE,
  open = FALSE
)
```

Arguments

name	Name of the GitHub action, with or without `.yaml` extension
url	The full URL to the GitHub Actions yaml file. By default, the corresponding action in https://github.com/r-lib/actions will be used.

save_as	Name of the actions file. Defaults to basename(url) for use_github_action().
ignore	Should the newly created file be added to .Rbuildignore?
open	Open the newly created file for editing? Happens in RStudio, if applicable, or via utils::file.edit() otherwise.

use_github_action_check_release()

This action installs the latest release R version on macOS and runs R CMD check via the rcmdcheck package.

use_github_action_check_standard()

This action runs R CMD check via the rcmdcheck package on the three major OSs (linux, macOS and Windows) on the release version of R and R-devel. This action is most appropriate when you plan to eventually submit your package to CRAN or Bioconductor.

use_github_action_check_full()

This action installs the last 5 minor R versions and runs R CMD check via the rcmdcheck package on the three major OSs (linux, macOS and Windows). This action is what the tidyverse teams uses on their repositories, but is overkill for less widely used packages, which are better off using the simpler use_github_action_check_release().

use_github_action_pr_commands()

This workflow enables the use of 2 R specific commands in pull request issue comments. /document will use roxygen2 to rebuild the documentation for the package and commit the result to the pull request. /style will use styler to restyle your package.

See Also

github_actions for generic workflows.

---

use_github_labels *Manage GitHub issue labels*

Description

use_github_labels() can create new labels, update colours and descriptions, and optionally delete GitHub's default labels (if delete_default = TRUE). It will never delete labels that have associated issues.

use_tidy_labels() calls use_github_labels() with tidyverse conventions powered by tidy_labels(), tidy_labels_rename(), tidy_label_colours() and tidy_label_descriptions().

Usage

```
use_github_labels(
  repo_spec = github_repo_spec(),
  labels = character(),
  rename = character(),
  colours = character(),
  descriptions = character(),
  delete_default = FALSE,
  auth_token = github_token(),
  host = NULL
)

use_tidy_labels(
  repo_spec = github_repo_spec(),
  auth_token = github_token(),
  host = NULL
)

tidy_labels()

tidy_labels_rename()

tidy_label_colours()

tidy_label_descriptions()
```

Arguments

repo_spec   Optional repository specification (`owner/repo`) if you don't want to target the
                 current project.

labels   A character vector giving labels to add.

rename   A named vector with names giving old names and values giving new names.

colours, descriptions

                 Named character vectors giving hexadecimal colours (like `e02a2a`) and longer
                 descriptions. The names should match label names, and anything unmatched
                 will be left unchanged. If you create a new label, and don't supply colours, it
                 will be given a random colour.

delete_default   If `TRUE`, removes GitHub default labels that do not appear in the `labels` vector
                 and that do not have associated issues.

auth_token   GitHub personal access token (PAT).

host   GitHub API host to use. Override with the endpoint-root for your GitHub enter-
                 prise instance, for example, "https://github.hostname.com/api/v3".

Label usage

Labels are used as part of the issue-triage process, designed to minimise the time spent re-reading
issues. The absence of a label indicates that an issue is new, and has yet to be triaged.

- reprex indicates that an issue does not have a minimal reproducible example, and that a reply has been sent requesting one from the user.

- bug indicates an unexpected problem or unintended behavior.

- feature indicates a feature request or enhancement.

- docs indicates an issue with the documentation.

- wip indicates that someone is working on it or has promised to.

- good first issue indicates a good issue for first-time contributors.

- help wanted indicates that a maintainer wants help on an issue.

Examples

```
## Not run:
# typical use in, e.g., a new tidyverse project
use_github_labels(delete_default = TRUE)

# create labels without changing colours/descriptions
use_github_labels(
  labels = c("foofy", "foofier", "foofiest"),
  colours = NULL,
  descriptions = NULL
)

# change descriptions without changing names/colours
use_github_labels(
  labels = NULL,
  colours = NULL,
  descriptions = c("foofiest" = "the foofiest issue you ever saw")
)

## End(Not run)
```

---

use_github_links   *Use GitHub links in URL and BugReports*

Description

Populates the URL and BugReports fields of a GitHub-using R package with appropriate links.

Usage

```
use_github_links(
  auth_token = github_token(),
  host = "https://api.github.com",
  overwrite = FALSE
)
```

Arguments

`auth_token`	GitHub personal access token (PAT).
`host`	GitHub API host to use. Override with the endpoint-root for your GitHub enterprise instance, for example, "https://github.hostname.com/api/v3".
`overwrite`	By default, `use_github_links()` will not overwrite existing fields. Set to `TRUE` to overwrite existing links.

Examples

```
## Not run:
use_github_links()

## End(Not run)
```

---

use_github_release   *Draft a GitHub release*

Description

Creates a **draft** GitHub release for the current package using the current version and `NEWS.md`. If you are comfortable that it is correct, you will need to publish the release from GitHub. It also deletes `CRAN-RELEASE` and checks that you've pushed all commits to GitHub.

Usage

```
use_github_release(host = NULL, auth_token = github_token())
```

Arguments

`host`	GitHub API host to use. Override with the endpoint-root for your GitHub enterprise instance, for example, "https://github.hostname.com/api/v3".
`auth_token`	GitHub personal access token (PAT).

---

use_git_config   *Configure Git*

Description

Sets Git options, for either the user or the project ("global" or "local", in Git terminology).

Usage

```
use_git_config(scope = c("user", "project"), ...)
```

Arguments

scope   Edit globally for the current **user**, or locally for the current **project**

...   Name-value pairs.

Value

Invisibly, the previous values of the modified components.

See Also

Other git helpers: use_git_hook(), use_git_ignore(), use_git()

Examples

```
## Not run:
# set the user's global user.name and user.email
use_git_config(user.name = "Jane", user.email = "jane@example.org")

# set the user.name and user.email locally, i.e. for current repo/project
use_git_config(
  scope = "project",
  user.name = "Jane",
  user.email = "jane@example.org"
)

## End(Not run)
```

---

use_git_hook   *Add a git hook*

Description

Sets up a git hook using specified script. Creates hook directory if needed, and sets correct permissions on hook.

Usage

```
use_git_hook(hook, script)
```

Arguments

hook   Hook name. One of "pre-commit", "prepare-commit-msg", "commit-msg", "post-commit", "applypatch-msg", "pre-applypatch", "post-applypatch", "pre-rebase", "post-rewrite", "post-checkout", "post-merge", "pre-push", "pre-auto-gc".

script   Text of script to run

See Also

Other git helpers: use_git_config(), use_git_ignore(), use_git()

---

use_git_ignore   *Tell git to ignore files*

Description

Tell git to ignore files

Usage

```
use_git_ignore(ignores, directory = ".")
```

Arguments

ignores	Character vector of ignores, specified as file globs.
directory	Directory relative to active project to set ignores

See Also

Other git helpers: `use_git_config()`, `use_git_hook()`, `use_git()`

---

use_git_remote   *Configure and report Git remotes*

Description

Two helpers are available:

- `use_git_remote()` sets the remote associated with `name` to `url`.
- `git_remotes()` reports the configured remotes, similar to git remote -v.

Usage

```
use_git_remote(name = "origin", url, overwrite = FALSE)

git_remotes()
```

Arguments

name	A string giving the short name of a remote.
url	A string giving the url of a remote.
overwrite	Logical. Controls whether an existing remote can be modified.

Value

Named list of Git remotes.

Examples

```
## Not run:
# see current remotes
git_remotes()

# add new remote named 'foo', a la `git remote add <name> <url>`
use_git_remote(name = "foo", url = "https://github.com/<OWNER>/<REPO>.git")

# remove existing 'foo' remote, a la `git remote remove <name>`
use_git_remote(name = "foo", url = NULL, overwrite = TRUE)

# change URL of remote 'foo', a la `git remote set-url <name> <newurl>`
use_git_remote(
  name = "foo",
  url = "https://github.com/<OWNER>/<REPO>.git",
  overwrite = TRUE
)

# Scenario: Fix remotes when you cloned someone's repo, but you should
# have fork-and-cloned (in order to make a pull request).

# Store origin = main repo's URL, e.g., "git@github.com:<OWNER>/<REPO>.git"
upstream_url <- git_remotes()[["origin"]]

# IN THE BROWSER: fork the main GitHub repo and get your fork's remote URL
my_url <- "git@github.com:<ME>/<REPO>.git"

# Rotate the remotes
use_git_remote(name = "origin", url = my_url)
use_git_remote(name = "upstream", url = upstream_url)
git_remotes()

# Scenario: Add upstream remote to a repo that you fork-and-cloned, so you
# can pull upstream changes.
# Note: If you fork-and-clone via `usethis::create_from_github()`, this is
# done automatically!

# Get URL of main GitHub repo, probably in the browser
upstream_url <- "git@github.com:<OWNER>/<REPO>.git"
use_git_remote(name = "upstream", url = upstream_url)

## End(Not run)
```

---

use_jenkins   *Create Jenkinsfile for Jenkins CI Pipelines*

Description

use_jenkins() adds a basic Jenkinsfile for R packages to the project root directory. The Jenkinsfile stages take advantage of calls to make, and so calling this function will also run use_make() if a Makefile does not already exist at the project root.

Usage

```
use_jenkins()
```

See Also

The [documentation on Jenkins Pipelines](#).

[use_make()](#)

---

use_lifecycle   *Use lifecycle badges*

Description

Call this to import the lifecycle badges and Rd macro into your package.

* The SVG badges are imported in `man/figures`.
* The `RdMacros` field of the DESCRIPTION file is updated so you can use the \\lifecycle{} macro in your documentation.

See the [getting started vignette](#) of the lifecycle package.

Usage

```
use_lifecycle()
```

See Also

[use_lifecycle_badge()](#) to signal the [global lifecycle stage](#) of your package.

---

use_logo   *Use a package logo*

Description

This function helps you use a logo in your package:

* Enforces a specific size
* Stores logo image file at `man/figures/logo.png`
* Produces the markdown text you need in README to include the logo

Usage

```
use_logo(img, geometry = "240x278", retina = TRUE)
```

Arguments

img	The path to an existing image file
geometry	a magick::geometry string specifying size. The default assumes that you have a hex logo using spec from `http://hexb.in/sticker.html`.
retina	TRUE, the default, scales the image on the README, assuming that geometry is double the desired size.

Examples

```
## Not run:
use_logo("usethis.png")

## End(Not run)
```

---

use_make   *Create Makefile*

Description

use_make() adds a basic Makefile to the project root directory.

Usage

```
use_make()
```

See Also

The documentation for GNU Make.

---

use_namespace   *Use a basic* NAMESPACE

Description

If roxygen is TRUE generates an empty NAMESPACE that exports nothing; you'll need to explicitly export functions with @export. If roxygen is FALSE, generates a default NAMESPACE that exports all functions except those that start with ..

Usage

```
use_namespace(roxygen = TRUE)
```

Arguments

roxygen	Do you plan to manage NAMESPACE with roxygen2?

See Also

The namespace chapter of R Packages.

---

use_news_md	*Create a simple* NEWS.md

Description

This creates a basic NEWS.md in the root directory.

Usage

```
use_news_md(open = rlang::is_interactive())
```

Arguments

open   Open the newly created file for editing? Happens in RStudio, if applicable, or via utils::file.edit() otherwise.

See Also

The important files section of R Packages.

---

use_package	*Depend on another package*

Description

use_package() adds a CRAN package dependency to DESCRIPTION and offers a little advice about how to best use it. use_dev_package() adds a versioned dependency on an in-development GitHub package, adding the repo to Remotes so it will be automatically installed from the correct location.

Usage

```
use_package(package, type = "Imports", min_version = NULL)

use_dev_package(package, type = "Imports")
```

Arguments

package	Name of package to depend on.
type	Type of dependency: must be one of "Imports", "Depends", "Suggests", "Enhances", or "LinkingTo" (or unique abbreviation). Matching is case insensitive.
min_version	Optionally, supply a minimum version for the package. Set to TRUE to use the currently installed version.

See Also

The dependencies section of R Packages.

Examples

```
## Not run:
use_package("ggplot2")
use_package("dplyr", "suggests")
use_dev_package("glue")

## End(Not run)
```

---

use_package_doc	*Package-level documentation*

Description

Adds a dummy `.R` file that will prompt roxygen to generate basic package-level documentation. If your package is named "foo", this will make help available to the user via `?foo` or `package?foo`. Once you call `devtools::document()`, roxygen will flesh out the `.Rd` file using data from the `DESCRIPTION`. That ensures you don't need to repeat the same information in multiple places. This `.R` file is also a good place for roxygen directives that apply to the whole package (vs. a specific function), such as global namespace tags like @importFrom.

Usage

```
use_package_doc()
```

See Also

The documentation chapter of R Packages

---

use_pipe	*Use magrittr's pipe in your package*

Description

Does setup necessary to use magrittr's pipe operator, %>% in your package. This function requires the use roxygen.

- Adds magrittr to "Imports" in `DESCRIPTION`.
- Imports the pipe operator specifically, which is necessary for internal use.
- Exports the pipe operator, if `export = TRUE`, which is necessary to make %>% available to the users of your package.

Usage

```
use_pipe(export = TRUE)
```

Arguments

export   If TRUE, the file R/utils-pipe.R is added, which provides the roxygen tem-
              plate to import and re-export %>%. If FALSE, the necessary roxygen directive is
              added, if possible, or otherwise instructions are given.

Examples

```
## Not run:
use_pipe()

## End(Not run)
```

---

use_pkgdown   *Use pkgdown*

Description

pkgdown makes it easy to turn your package into a beautiful website. There are two helper func-
tions:

- use_pkgdown(): creates a pkgdown config file, adds relevant files or directories to .Rbuildignore
  and .gitignore, and builds favicons if your package has a logo
- use_pkgdown_travis(): helps you set up pkgdown for automatic deployment on Travis-
  CI. As part of a general pivot away from Travis-CI and towards GitHub Actions, the tidyverse
  team now builds pkgdown sites with a workflow configured via use_github_action("pkgdown").
  use_pkgdown_travis() is still here, but is no longer actively exercised by us. Key steps:
    - Creates an empty gh-pages branch for the pkgdown site to be deployed to.
    - Prompts you about what to do next regarding Travis-CI deployment keys and updating
      your .travis.yml.

Usage

```
use_pkgdown(config_file = "_pkgdown.yml", destdir = "docs")

use_pkgdown_travis()
```

Arguments

config_file   Path to the pkgdown yaml config file
destdir   Target directory for pkgdown docs

See Also

https://pkgdown.r-lib.org/articles/pkgdown.html#configuration

---

use_r   *Create or edit R or test files*

Description

This pair of functions makes it easy to create paired R and test files, using the convention that the tests for R/foofy.R should live in tests/testhat/test-foofy.R. You can use them to create new files from scratch by supplying name, or if you use RStudio, you can call to create (or navigate to) the paired file based on the currently open script.

Usage

```
use_r(name = NULL, open = rlang::is_interactive())

use_test(name = NULL, open = rlang::is_interactive())
```

Arguments

name   Either a name without extension, or NULL to create the paired file based on cur-
              rently open file in the script editor. If the R file is open, use_test() will cre-
              ate/open the corresponding test file; if the test file is open, use_r() will cre-
              ate/open the corresponding R file.

open   Whether to open the file for interactive editing.

See Also

The testing and R code chapters of R Packages.

---

use_rcpp   *Use C, C++, RcppArmadillo, or RcppEigen*

Description

Adds infrastructure commonly needed when using compiled code:

- Creates src/

- Adds required packages to DESCRIPTION

- May create an initial placeholder .c or .cpp file

- Creates Makevars and Makevars.win files (use_rcpp_armadillo() only)

Usage

```
use_rcpp(name = NULL)

use_rcpp_armadillo(name = NULL)

use_rcpp_eigen(name = NULL)

use_c(name = NULL)
```

Arguments

name   If supplied, creates and opens src/name.{c,cpp}.

Details

When using compiled code, please note that there must be at least one file inside the src/ directory prior to building the package. As a result, if an empty src/ directory is detected, either a .c or .cpp file will be added.

---

use_readme_rmd   *Create README files*

Description

Creates skeleton README files with sections for

- a high-level description of the package and its goals
- R code to install from GitHub, if GitHub usage detected
- a basic example

Use Rmd if you want a rich intermingling of code and data. Use md for a basic README. README.Rmd will be automatically added to .Rbuildignore. The resulting README is populated with default YAML frontmatter and R fenced code blocks (md) or chunks (Rmd).

Usage

```
use_readme_rmd(open = rlang::is_interactive())

use_readme_md(open = rlang::is_interactive())
```

Arguments

open   Open the newly created file for editing? Happens in RStudio, if applicable, or
                via utils::file.edit() otherwise.

See Also

The important files section of R Packages.

Examples

```
## Not run:
use_readme_rmd()
use_readme_md()

## End(Not run)
```

---

use_release_issue   *Create a release issue checklist*

Description

When preparing to release a package there are quite a few steps that need to be performed, and some of the steps can take multiple hours. This function creates an issue checklist so that you can keep track of where you are in the process, and feel a sense of satisfaction as you progress. It also helps watchers of your package stay informed about where you are in the process.

Usage

```
use_release_issue(version = NULL)
```

Arguments

version   Version number for release

Examples

```
## Not run:
use_release_issue("2.0.0")

## End(Not run)
```

---

use_revdep   *Reverse dependency checks*

Description

Performs set up for checking the reverse dependencies of an R package, as implemented by the revdepcheck package:

- Adds `revdep` directory and adds it to `.Rbuildignore`
- Populates `revdep/.gitignore` to prevent tracking of various revdep artefacts
- Creates `revdep/email.yml` for use with `revdepcheck::revdep_email()`
- Prompts user to run the checks with `revdepcheck::revdep_check()`

Usage

```
use_revdep()
```

use_rmarkdown_template

Add an RMarkdown Template

Description

Adds files and directories necessary to add a custom rmarkdown template to RStudio. It creates:

- inst/rmarkdown/templates/{{template_dir}}. Main directory.

- skeleton/skeleton.Rmd. Your template Rmd file.

- template.yml with basic information filled in.

Usage

```
use_rmarkdown_template(
  template_name = "Template Name",
  template_dir = NULL,
  template_description = "A description of the template",
  template_create_dir = FALSE
)
```

Arguments

template_name	The name as printed in the template menu.
template_dir	Name of the directory the template will live in within inst/rmarkdown/templates. If none is provided by the user, it will be created from template_name.
template_description	
	Sets the value of description in template.yml.
template_create_dir	
	Sets the value of create_dir in template.yml.

Examples

```
## Not run:
use_rmarkdown_template()

## End(Not run)
```

---

use_roxygen_md   *Use roxygen2 with markdown*

Description

If you are already using roxygen2, but not with markdown, you'll need to use [roxygen2md](#) to convert existing Rd expressions to markdown. The conversion is not perfect, so make sure to check the results.

Usage

```
use_roxygen_md()
```

---

use_rstudio   *Add RStudio Project infrastructure*

Description

It is likely that you want to use [create_project()](#) or [create_package()](#) instead of use_rstudio()! Both create_*() functions can add RStudio Project infrastructure to a pre-existing project or package. use_rstudio() is mostly for internal use or for those creating a usethis-like package for their organization. It does the following in the current project, often after executing proj_set(..., force = TRUE):

- Creates an .Rproj file
- Adds RStudio files to .gitignore
- Adds RStudio files to .Rbuildignore, if project is a package

Usage

```
use_rstudio(line_ending = c("posix", "windows"))
```

Arguments

line_ending   Line ending

---

use_spell_check   *Use spell check*

Description

Adds a unit test to automatically run a spell check on documentation and, optionally, vignettes during R CMD check, using the spelling package. Also adds a WORDLIST file to the package, which is a dictionary of whitelisted words. See spelling::wordlist for details.

Usage

```
use_spell_check(vignettes = TRUE, lang = "en-US", error = FALSE)
```

Arguments

vignettes	Logical, TRUE to spell check all rmd and rnw files in the vignettes/ folder.
lang	Preferred spelling language. Usually either "en-US" or "en-GB".
error	Logical, indicating whether the unit test should fail if spelling errors are found. Defaults to FALSE, which does not error, but prints potential spelling errors

---

use_template   *Use a usethis-style template*

Description

Creates a file from data and a template found in a package. Provides control over file name, the addition to .Rbuildignore, and opening the file for inspection.

Usage

```
use_template(
  template,
  save_as = template,
  data = list(),
  ignore = FALSE,
  open = FALSE,
  package = "usethis"
)
```

Arguments

`template`	Path to template file relative to templates/ directory within package; see details.
`save_as`	Path of file to create, relative to root of active project. Defaults to `template`
`data`	A list of data passed to the template.
`ignore`	Should the newly created file be added to `.Rbuildignore`?
`open`	Open the newly created file for editing? Happens in RStudio, if applicable, or via `utils::file.edit()` otherwise.
`package`	Name of the package where the template is found.

Details

This function can be used as the engine for a templating function in other packages. The `template` argument is used along with the `package` argument to derive the path to your template file; it will be expected at `fs::path_package(package = package,"templates",template)`. We use `fs::path_package()` instead of `base::system.file()` so that path construction works even in a development workflow, e.g., works with devtools::load_all() or pkgload::load_all(). *Note this describes the behaviour of* `fs::path_package()` *in fs v1.2.7.9001 and higher.*

To interpolate your data into the template, supply a list using the `data` argument. Internally, this function uses `whisker::whisker.render()` to combine your template file with your data.

Value

A logical vector indicating if file was modified.

Examples

```
## Not run:
  # Note: running this will write `NEWS.md` to your working directory
  use_template(
    template = "NEWS.md",
    data = list(Package = "acme", Version = "1.2.3"),
    package = "usethis"
  )

## End(Not run)
```

---

use_testthat	*Sets up overall testing infrastructure*

Description

Creates tests/testthat/, tests/testthat.R, and adds the testthat package to the Suggests field. Learn more in https://r-pkgs.org/tests.html

Usage

```
use_testthat()
```

See Also

[use_test()](#) to create individual test files

Examples

```
## Not run:
use_testthat()

use_test()

use_test("something-management")

## End(Not run)
```

---

use_tibble   *Prepare to return a tibble*

Description

Does minimum setup such that a tibble returned by your package is handled using the tibble method for generics like print() or [. Presumably you care about this if you've chosen to store and expose an object with class tbl_df. Specifically:

- Check that the active package uses roxygen2
- Add the tibble package to "Imports" in DESCRIPTION
- Reveal the roxygen directive necessary to import at least one function from tibble.
- Offer support re: where to put this directive. Preferred location is in the roxygen snippet produced by [use_package_doc()](#).

This is necessary when your package returns a stored data object that has class tbl_df, but the package code does not make direct use of functions from the tibble package. If you do nothing, the tibble namespace is not necessarily loaded and your tibble may therefore be printed and subsetted like a base data.frame.

Usage

```
use_tibble()
```

Examples

```
## Not run:
use_tibble()

## End(Not run)
```

---

use_tidy_github_actions

Helpers for tidyverse development

Description

These helpers follow tidyverse conventions which are generally a little stricter than the defaults, reflecting the need for greater rigor in commonly used packages.

Usage

```
use_tidy_github_actions()

create_tidy_package(path, name = "RStudio")

use_tidy_description()

use_tidy_eval()

use_tidy_contributing()

use_tidy_support()

use_tidy_issue_template()

use_tidy_coc()

use_tidy_github()

use_tidy_style(strict = TRUE)

use_tidy_release_test_env()
```

Arguments

path	A path. If it exists, it is used. If it does not exist, it is created, provided that the parent path exists.
name	Name of the copyright holder or holders. Separate multiple individuals with ;. You can supply a global default with options(usethis.full_name = "My name").
strict	Boolean indicating whether or not a strict version of styling should be applied. See `styler::tidyverse_style()` for details.

Details

- use_tidy_github_actions(): Sets up the following workflows using GitHub Actions:

- **–** Runs R CMD check on the current release, devel, and four previous versions of R.
- **–** Adds adds two commands to be used in pull requests: /document to run roxygen2::roxygenise() and update the PR, and /style to run styler::style_pkg() and update the PR.
- **–** Builds a pkgdown site for the package.

- create_tidy_package(): creates a new package, immediately applies as many of the tidy-verse conventions as possible, issues a few reminders, and activates the new package.

- use_tidy_description(): puts fields in standard order and alphabetises dependencies.

- use_tidy_eval(): imports a standard set of helpers to facilitate programming with the tidy eval toolkit.

- use_tidy_style(): styles source code according to the tidyverse style guide. This function will overwrite files! See below for usage advice.

- use_tidy_contributing(): adds standard tidyverse contributing guidelines.

- use_tidy_issue_template(): adds a standard tidyverse issue template.

- use_tidy_release_test_env(): updates the test environment section in cran-comments.md.

- use_tidy_support(): adds a standard description of support resources for the tidyverse.

- use_tidy_coc(): equivalent to use_code_of_conduct(), but puts the document in a .github/ subdirectory.

- use_tidy_github(): convenience wrapper that calls use_tidy_contributing(), use_tidy_issue_template(), use_tidy_support(), use_tidy_coc().

use_tidy_style()

Uses the styler package package to style all code in a package, project, or directory, according to the tidyverse style guide.

Warning: This function will overwrite files! It is strongly suggested to only style files that are under version control or to first create a backup copy.

Invisibly returns a data frame with one row per file, that indicates whether styling caused a change.

---

use_tidy_thanks   *Identify contributors via GitHub activity*

Description

Derives a list of GitHub usernames, based on who has opened issues or pull requests. Used to populate the acknowledgment section of package release blog posts at https://www.tidyverse.org/articles/. All arguments can potentially be determined from the active project, if the project follows standard practices around the GitHub remote and GitHub releases. Unexported helper functions, releases() and ref_df() can be useful interactively to get a quick look at release tag names and a data frame about refs (defaulting to releases), respectively.

Usage

```
use_tidy_thanks(
  repo_spec = github_repo_spec(),
  from = releases(repo_spec)[[1]],
  to = NULL
)
```

Arguments

`repo_spec`	GitHub repo specification in this form: `owner/repo`. Default is to infer from Git remotes of active project.
`from, to`	GitHub ref (i.e., a SHA, tag, or release) or a timestamp in ISO 8601 format, specifying the start or end of the interval of interest. Examples: "08a560d", "v1.3.0", "2018-02-24T00:13:45Z", "2018-05-01". `NULL` means there is no bound on that end of the interval.

Value

A character vector of GitHub usernames, invisibly.

Examples

```
## Not run:
## active project, interval = since the last release
use_tidy_thanks()

## active project, interval = since a specific datetime
use_tidy_thanks(from = "2018-02-24T00:13:45Z")

## r-lib/usethis, inteval = since a certain date
use_tidy_thanks("r-lib/usethis", from = "2018-05-01")

## r-lib/usethis, up to a specific release
use_tidy_thanks("r-lib/usethis", from = NULL, to = "v1.3.0")

## r-lib/usethis, since a specific commit, up to a specific date
use_tidy_thanks("r-lib/usethis", from = "08a560d", to = "2018-05-14")

## End(Not run)
```

---

use_tutorial   *Create a learnr tutorial*

Description

Creates a new tutorial below inst/tutorials/. Tutorials are interactive R Markdown documents built with the learnr package. use_tutorial() does this setup:

- Adds learnr to Suggests in DESCRIPTION.

- Gitignores inst/tutorials/*.html so you don't accidentally track rendered tutorials.
- Creates a new .Rmd tutorial from a template and, optionally, opens it for editing.
- Adds new .Rmd to .Rbuildignore.

Usage

```
use_tutorial(name, title, open = rlang::is_interactive())
```

Arguments

name	Base for file name to use for new .Rmd tutorial. Should consist only of numbers, letters, _ and -. We recommend using lower case.
title	The human-facing title of the tutorial.
open	Open the newly created file for editing? Happens in RStudio, if applicable, or via utils::file.edit() otherwise.

See Also

The learnr package documentation.

Examples

```
## Not run:
use_tutorial("learn-to-do-stuff", "Learn to do stuff")

## End(Not run)
```

---

use_version   *Increment package version*

Description

use_version() increments the "Version" field in DESCRIPTION, adds a new heading to NEWS.md (if it exists), and commits those changes (if package uses Git).

use_dev_version() increments to a development version, e.g. from 1.0.0 to 1.0.0.9000. If the existing version is already a development version with four components, it does nothing. Thin wrapper around use_version().

Usage

```
use_version(which = NULL)

use_dev_version()
```

Arguments

which	A string specifying which level to increment, one of: "major", "minor", "patch", "dev". If NULL, user can choose interactively.

See Also

The version section of R Packages.

Examples

```
## Not run:
## for interactive selection, do this:
use_version()

## request a specific type of increment
use_version("minor")
use_dev_version()

## End(Not run)
```

---

use_vignette   *Create a vignette or article.*

Description

Creates a new vignette or article in vignettes/. Articles are a special type of vignette that appear on pkgdown websites, but are not included in the package itself (because they are added to .Rbuildignore automatically).

Usage

```
use_vignette(name, title = name)

use_article(name, title = name)
```

Arguments

name   Base for file name to use for new vignette. Should consist only of numbers, letters, _ and -. Lower case is recommended.

title   The title of the vignette.

General setup

- Adds needed packages to DESCRIPTION.
- Adds inst/doc to .gitignore so built vignettes aren't tracked.
- Adds vignettes/*.html and vignettes/*.R to .gitignore so you never accidentally track rendered vignettes.

See Also

The vignettes chapter of R Packages.

Examples

```
## Not run:
use_vignette("how-to-do-stuff", "How to do stuff")

## End(Not run)
```

---

zip-utils   *Download and unpack a ZIP file*

Description

Functions to download and unpack a ZIP file into a local folder of files, with very intentional default behaviour. Useful in pedagogical settings or anytime you need a large audience to download a set of files quickly and actually be able to find them. The underlying helpers are documented in use_course_details.

Usage

```
use_course(url, destdir = getOption("usethis.destdir"))

use_zip(
  url,
  destdir = getwd(),
  cleanup = if (rlang::is_interactive()) NA else FALSE
)
```

Arguments

url   Link to a ZIP file containing the materials. To reduce the chance of typos in live
                settings, these shorter forms are accepted:

                ```
                * GitHub repo spec: "OWNER/REPO". Auto-expands to
                  `https://github.com/r-lib/OWNER/REPO/master.zip`.
                * bit.ly or rstd.io shortlinks: "bit.ly/xxx-yyy-zzz" or "rstd.io/foofy".
                  The instructor must then arrange for the shortlink to point to a valid
                   download URL for the target ZIP file. The helper
                  [create_download_url()] helps to create such URLs for GitHub, DropBox,
                   and Google Drive.
                ```

destdir   The new folder is stored here. If NULL, defaults to user's Desktop or some
                other conspicuous place. You can also set a default location using the option
                usethis.destdir, e.g. options(usethis.destdir = "a/good/dir"), per-
                haps saved to your .Rprofile with edit_r_profile()

cleanup   Whether to delete the original ZIP file after unpacking its contents. In an in-
                teractive setting, NA leads to a menu where user can approve the deletion (or
                decline).

Value

Path to the new directory holding the unpacked ZIP file, invisibly.

Functions

- use_course: Designed with live workshops in mind. Includes intentional friction to highlight the download destination. Workflow:

    - User executes, e.g., use_course("bit.ly/xxx-yyy-zzz").
    - User is asked to notice and confirm the location of the new folder. Specify destdir or configure the "usethis.destdir" option to prevent this.
    - User is asked if they'd like to delete the ZIP file.
    - If new folder contains an .Rproj file, a new instance of RStudio is launched. Otherwise, the folder is opened in the file manager, e.g. Finder or File Explorer.

- use_zip: More useful in day-to-day work. Downloads in current working directory, by default, and allows cleanup behaviour to be specified.

Examples

```
## Not run:
# download the source of usethis from GitHub, behind a bit.ly shortlink
use_course("bit.ly/usethis-shortlink-example")
use_course("http://bit.ly/usethis-shortlink-example")

# download the source of rematch2 package from CRAN
use_course("https://cran.r-project.org/bin/windows/contrib/3.4/rematch2_2.0.1.zip")

# download the source of rematch2 package from GitHub, 3 ways
use_course("r-lib/rematch2")
use_course("https://github.com/r-lib/rematch2/archive/master.zip")
use_course("https://api.github.com/repos/r-lib/rematch2/zipball/master")

## End(Not run)
```

Index