Package 'usethis'

August 14, 2018

Title Automate Package and Project Setup

Version 1.4.0

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://github.com/r-lib/usethis

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

Depends R (>= 3.1)

Imports clipr (>= 0.3.0), clisymbols, crayon, curl (>= 2.7), desc, fs (>= 1.2.0), gh, git2r (>= 0.23), glue, rlang, rprojroot (>= 1.2), rstudioapi, utils, whisker

Suggests covr, knitr, magick, rmarkdown, roxygen2, spelling (>= 1.2), styler (>= 1.0.2), testthat (>= 2.0.0), withr

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 6.1.0

NeedsCompilation no

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

Maintainer Hadley Wickham <hadley@rstudio.com>

Repository CRAN

Date/Publication 2018-08-14 12:10:02 UTC

R topics documented:

badges . . . 3
browse-this . . . 4
browse_github_pat . . . 5
ci . . . 6
create_from_github . . . 7
create_package . . . 8
edit . . . 9
licenses . . . 10
proj_sitrep . . . 11
proj_utils . . . 11
tidyverse . . . 13
use_blank_slate . . . 14
use_build_ignore . . . 15
use_code_of_conduct . . . 15
use_course . . . 16
use_cran_comments . . . 17
use_data . . . 17
use_description . . . 18
use_directory . . . 19
use_git . . . 20
use_github . . . 21
use_github_labels . . . 22
use_github_links . . . 23
use_git_config . . . 24
use_git_hook . . . 25
use_git_ignore . . . 25
use_logo . . . 26
use_namespace . . . 26
use_news_md . . . 27
use_package . . . 27
use_package_doc . . . 28
use_pipe . . . 28
use_pkgdown . . . 29
use_r . . . 29
use_rcpp . . . 29
use_readme_rmd . . . 30
use_revdep . . . 31
use_rmarkdown_template . . . 31
use_roxygen_md . . . 32
use_rstudio . . . 32
use_spell_check . . . 33
use_template . . . 33
use_testthat . . . 34
use_tibble . . . 35
use_tidy_thanks . . . 35
use_usethis . . . 36

use_version 37
use_vignette 38

Index **39**

`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.

Usage

```
use_badge(badge_name, href, src)

use_cran_badge()

use_bioc_badge()

use_lifecycle_badge(stage)

use_binder_badge()
```

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

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
  - Retired
  - 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)

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"`.

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_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_pat` *Create a GitHub personal access token*

Description

Opens a browser window to the GitHub page where you can generate a Personal Access Token. Make sure you have signed up for a free GitHub.com account and that you are signed in. Click "Generate token" after you have verified the scopes. Copy the token right away! You probably want to store it in `.Renviron` as the `GITHUB_PAT` environment variable. `edit_r_environ()` can help you do that. Use `gh::gh_whoami()` to get information on an existing token.

Usage

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

Arguments

`scopes`	Character vector of token permissions. These are just defaults that will be pre-selected in the web form. You can select the final values on the GitHub page. 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"

Examples

```
## Not run:
browse_github_pat()
## 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)
```

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 = interactive())

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

use_appveyor(browse = interactive())
```

Arguments

`browse`	Open a browser window to enable automatic builds for the package.
`type`	Which web service to use for test reporting. Currently supports Codecov and Coveralls.

`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_coverage()`

Adds test coverage reports to a package that is already using Travis CI.

`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.

`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_pat()`. 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. `use_github()` has more detailed advice on working with the `protocol` and `credentials` arguments.

Usage

```
create_from_github(repo_spec, destdir = NULL, fork = NA,
  rstudio = NULL, open = interactive(), protocol = c("ssh", "https"),
  credentials = NULL, auth_token = NULL, 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. Defaults to user's Desktop.
`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. If a fork is created, the original target repo is added to the local repo as the `upstream` remote, using your preferred `protocol`, to make it easier to pull upstream changes in the future.
`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` and in RStudio, a new RStudio project is opened in a new instance, if possible, or is switched to, otherwise. If `TRUE` and not in RStudio (or new project is not an RStudio project), working directory is set to the new project.
`protocol`	transfer protocol, either "ssh" (the default) or "https"
`credentials`	A `git2r::cred_ssh_key()` specifying specific ssh credentials or `NULL` for default ssh key and ssh-agent behaviour.
`auth_token`	Provide a personal access token (PAT) from `https://github.com/settings/tokens`. If `NULL`, will use the logic described in `gh::gh_whoami()` to look for a token stored in an environment variable. Use `browse_github_pat()` to help set up your 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"

See Also

use_github() for GitHub setup advice. 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")

create_from_github("r-lib/usethis", protocol = "https")

## various ways code can look when specifying ssh credential explicitly
create_from_github("r-lib/usethis", credentials = git2r::cred_ssh_key())

cred <- git2r::cred_ssh_key(
  publickey = "path/to/id_rsa.pub",
  privatekey = "path/to/id_rsa"
)
create_from_github("cran/TailRank", credentials = cred)

## 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 add project infrastructure to an existing directory of files or can create a completely new project. Both functions change the active project, so that subsequent `use_*()` calls affect the project that you've just created. See proj_set() to manually reset it.

Usage

```
create_package(path, fields = NULL,
  rstudio = rstudioapi::isAvailable(), open = interactive())

create_project(path, rstudio = rstudioapi::isAvailable(),
  open = 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.
open	If TRUE and in RStudio, a new RStudio project is opened in a new instance, if possible, or is switched to, otherwise. If TRUE and not in RStudio (or new project is not an RStudio project), working directory is set to the new project.

---

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

edit_rstudio_snippets(type = "R")

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. One of: "R", "markdown", "C_Cpp", "Tex", "Javascript", "HTML", "SQL"

Details

The `edit_r_*()` and `edit_rstudio_*()` functions 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.

Value

Path to the file, invisibly.

`licenses` *License a package*

Description

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

- 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.

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

Usage

```
use_mit_license(name = find_name())

use_gpl3_license(name = find_name())

use_apl2_license(name = find_name())

use_cc0_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_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(quiet = FALSE)

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

proj_path(..., ext = "")
```

Arguments

quiet	Logical. Whether to announce project activation.
path	Path to set.
force	If TRUE, use this path without checking the usual criteria. 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.
ext	An optional extension to append to the generated path.

Details

In general, end user code should not call `usethis::proj_get()`, `usethis::proj_set()`, or `usethis::proj_path()`. 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. Thin wrapper around `fs::path()`.

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")

## End(Not run)
```

`tidyverse` *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_ci(browse = interactive())

use_tidy_description()

use_tidy_versions(overwrite = FALSE)

use_tidy_eval()

use_tidy_contributing()

use_tidy_issue_template()

use_tidy_support()

use_tidy_coc()

use_tidy_github()

use_tidy_style(strict = TRUE)
```

Arguments

`browse`	Open a browser window to enable automatic builds for the package.
`overwrite`	By default (`FALSE`), only dependencies without version specifications will be modified. Set to `TRUE` to modify all dependencies.
`strict`	Boolean indicating whether or not a strict version of styling should be applied. See `styler::tidyverse_style()` for details.

Details

- `use_tidy_ci()`: sets up Travis CI and Codecov, ensuring that the package works on all versions of R starting at 3.1.
- `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_versions()`: pins all dependencies to require at least the currently installed version.
- `use_tidy_contributing()`: adds standard tidyverse contributing guidelines.
- `use_tidy_issue_template()`: adds a standard tidyverse issue template.
- `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_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.

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 $.

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 http://contributor-covenant.org, version 1: http://contributor-covenant.org/version/1/0/0/.

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.

`use_course` *Download course materials*

Description

Special-purpose function to download a folder of course materials. The only demand on the user is to confirm or specify where the new folder should be stored. Workflow:

- User executes something like: `use_course("bit.ly/xxx-yyy-zzz")`.
- User is asked to notice and confirm the location of the new folder. Specify `destdir` to skip this.
- User is asked if they'd like to delete the ZIP file.
- If new folder contains an `.Rproj` file, it is opened. Otherwise, the folder is opened in the file manager, e.g. Finder or File Explorer.

Usage

```
use_course(url, destdir = NULL)
```

Arguments

`url`	Link to a ZIP file containing the materials, possibly behind a shortlink. Function developed with DropBox and GitHub in mind, but should work for ZIP files generally. If no "http" prefix is found, "https://" is prepended. See use_course_details for more.
`destdir`	The new folder is stored here. Defaults to user's Desktop.

Details

If `url` has no "http" prefix, "https://" is prepended, allowing for even less typing by the user. Most URL shorteners give HTTPS links and, anecdotally, we note this appears to work with bit.ly links, even though they are nominally HTTP.

Value

Path to the new directory holding the course materials, invisibly.

See Also

Other download functions: `use_course_details`

Examples

```
## Not run:
## bit.ly shortlink example
## should work with and without http prefix
use_course("bit.ly/usethis-shortlink-example")
use_course("http://bit.ly/usethis-shortlink-example")

## demo with a small CRAN package available in various places

## from CRAN
use_course("https://cran.r-project.org/bin/windows/contrib/3.4/rematch2_2.0.1.zip")

## from GitHub, 2 ways
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)
```

---

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 = 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.

Usage

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

use_data_raw()
```

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".

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)
```

---

use_description *Create or modify a DESCRIPTION file*

Description

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"))',
    License = "MIT + file LICENSE",
    Language: es
  )
)
```

Usage

```
use_description(fields = NULL)

use_description_defaults()
```

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

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 = c("ssh",
  "https"), credentials = NULL, auth_token = NULL, host = NULL)
```

Arguments

`organisation`	If supplied, the repo will be created under this organisation. You must have access to create repositories.
`private`	If `TRUE`, creates a private repository.
`protocol`	transfer protocol, either "ssh" (the default) or "https"
`credentials`	A `git2r::cred_ssh_key()` specifying specific ssh credentials or `NULL` for default ssh key and ssh-agent behaviour.
`auth_token`	Provide a personal access token (PAT) from `https://github.com/settings/tokens`. If `NULL`, will use the logic described in `gh::gh_whoami()` to look for a token stored in an environment variable. Use `browse_github_pat()` to help set up your 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"

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 in an environment variable. Use `browse_github_pat()` to get help obtaining and storing your PAT. See `gh::gh_whoami()` for even more detail.

The argument `protocol` specifies the transport protocol you wish to use for this repo in the long run. This determines the form of the URL for the `origin` remote:

- `protocol = "ssh"`: `git@github.com:<OWNER>/<REPO>.git`
- `protocol = "https"`: `https://github.com/<OWNER>/<REPO>.git`

For `protocol = "ssh"`, it is assumed that public and private keys are in the default locations, `~/.ssh/id_rsa.pub` and `~/.ssh/id_rsa`, respectively, and that `ssh-agent` is configured to manage any associated passphrase. Alternatively, specify a `git2r::cred_ssh_key()` object via the `credentials` parameter. Read more about ssh setup in Happy Git, especially the troubleshooting section.

Examples

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

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

## create github repository and configure as git remote
use_github()                   ## to use default ssh protocol
use_github(protocol = "https") ## to use https

## End(Not run)
```

`use_github_labels` *Manage GitHub issue labels*

Description

`use_github_labels()` creates new labels and/or changes label colours. It does not generally remove labels. But if you set `delete_default = TRUE`, it will delete labels that are (1) flagged by the API as being GitHub default labels and (2) not present in the labels you provide via `labels`.

`tidy_labels()` returns the labels and colours commonly used by tidyverse packages.

Usage

```
use_github_labels(labels = tidy_labels(), delete_default = FALSE,
  auth_token = NULL, host = NULL)

tidy_labels()
```

Arguments

`labels`	Named character vector of labels. The names are the label text, such as "bug", and the values are the label colours in hexadecimal, such as "e02a2a". First, labels that don't yet exist are created, then label colours are updated.
`delete_default`	If `TRUE`, will remove GitHub default labels that do not appear in the `labels` vector (presumably defaults that aren't relevant to your workflow).
`auth_token`	Provide a personal access token (PAT) from `https://github.com/settings/tokens`. If `NULL`, will use the logic described in `gh::gh_whoami()` to look for a token stored in an environment variable. Use `browse_github_pat()` to help set up your 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"

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 type, indicates an unexpected problem or unintended behavior.
- `feature` type, indicates a feature request or enhancement.
- `docs` type, indicates an issue with the documentation.
- `performance` indicates a non-breaking area related to performance.
- `wip` indicates that someone else 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)

## 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 = NULL, host = "https://api.github.com",
  overwrite = FALSE)
```

Arguments

`auth_token`	Provide a personal access token (PAT) from https://github.com/settings/tokens. If `NULL`, will use the logic described in `gh::gh_whoami()` to look for a token stored in an environment variable. Use `browse_github_pat()` to help set up your 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_git_config` *Configure Git*

Description

Sets Git options, for either the user or the project ("global" or "local", in Git terminology). The mandate is currently very narrow: to manage the user name and email. The `scope` argument is consulted when writing. When reading, `use_git_config()` ignores `scope` and simply reports the options in effect, where local config overrides global, if present. Use `git2r::config()` directly or the command line for general Git configuration.

Usage

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

Arguments

`scope`	Edit globally for the current **user**, or locally for the current **project**
`...`	Additional options to write or delete from the configuration.

Value

A list with components `user.name` and `user.email`.

See Also

Other git helpers: `use_git_hook`, `use_git_ignore`, `use_git`

Examples

```
## Not run:
## see if user name and email are currently configured
use_git_config()

## 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_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 = "120x140")
```

Arguments

`img`	The path to an existing image file
`geometry`	a magick::geometry string specifying size

Examples

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

## End(Not run)
```

use_namespace *Use a basic* `NAMESPACE`

Description

This `NAMESPACE` exports everything, except functions that start with a `.`.

Usage

```
use_namespace()
```

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 = 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")

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.

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 internally in your package and to re-export it for users of your package:

- Adds magrittr to "Imports" in DESCRIPTION
- Creates `R/utils-pipe.R` with the necessary roxygen template

Usage

```
use_pipe()
```

use_pkgdown *Use pkgdown*

Description

pkgdown makes it easy to turn your package into a beautiful website. This helper creates `_pkgdown.yml` and `docs/` for you, and adds them to `.Rbuildignore`

Usage

```
use_pkgdown()
```

use_r *Create or edit a .R file*

Description

Create or edit a .R file

Usage

```
use_r(name = NULL)
```

Arguments

`name`	File name, without extension; will create if it doesn't already exist. If not specified, and you're currently in a test file, will guess name based on test name.

See Also

`use_test()`, and also the R code chapter of R Packages.

use_rcpp *Use Rcpp*

Description

Creates `src/` and adds needed packages to `DESCRIPTION`.

Usage

```
use_rcpp()
```

`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 = interactive())

use_readme_md(open = 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_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 = asciify(template_name),
  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`.
`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 roxygen with markdown*

Description

You'll need to manually re-document once enabled. If you are already using roxygen2, but not with markdown, the roxygen2md package will be used to convert many Rd expressions to markdown. The package uses heuristics, so you'll need 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()
```

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`	Name of file to create. 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 determine the path to your template file; it will be expected at `system.file("templates", template, package = package)`.

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` *Create tests*

Description

`use_testthat()` sets up testing infrastructure, creating 'tests/testthat.R' and 'tests/testthat/', and adding **testthat** to the suggested packages. `use_test()` creates 'tests/testthat/test-<name>.R' and opens it for editing.

Usage

```
use_testthat()

use_test(name = NULL, open = interactive())
```

Arguments

`name`	Test name. if `NULL`, and you're using RStudio, will use the name of the file open in the source editor.
`open`	Open the newly created file for editing? Happens in RStudio, if applicable, or via `utils::file.edit()` otherwise.

See Also

The testing chapter of R Packages.

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_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_usethis *Make usethis available in interactive sessions*

Description

Opens your .Rprofile and gives you the code you need to paste in.

Usage

```
use_usethis()
```

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*

Description

Performs general set up for vignettes and initializes a new individual vignette:

- Adds needed packages to `DESCRIPTION`
- Adds `inst/doc` to `.gitignore` so built vignettes aren't tracked
- Creates a new draft `.Rmd` vignette in `vignettes/` and, if possible, opens it for editing

Usage

```
use_vignette(name)
```

Arguments

`name`	Base for file name to use for new vignette. Should consist only of numbers, letters, _ and -. I recommend using lower case.

See Also

The vignettes chapter of R Packages.

Examples

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

## End(Not run)
```

Index

badges, 3
browse-this, 4
browse_cran *(*browse-this*)*, 4
browse_github *(*browse-this*)*, 4
browse_github_issues *(*browse-this*)*, 4
browse_github_pat, 5
browse_github_pat(), *7*, *21–23*
browse_github_pulls *(*browse-this*)*, 4
browse_travis *(*browse-this*)*, 4

ci, 6
create_from_github, 7
create_package, 8
create_package(), *18*, *32*
create_project *(*create_package*)*, 8
create_project(), *32*

data(), *18*

edit, 9
edit_git_config *(*edit*)*, 9
edit_git_ignore *(*edit*)*, 9
edit_r_environ *(*edit*)*, 9
edit_r_environ(), *5*
edit_r_makevars *(*edit*)*, 9
edit_r_profile *(*edit*)*, 9
edit_rstudio_snippets *(*edit*)*, 9

fs::path(), *12*
fs::path_home(), *10*
functions that set up continuous integration services, *4*

gh::gh_whoami(), *5*, *7*, *21–23*
git2r::config(), *24*
git2r::cred_ssh_key(), *7*, *21*

licenses, 10

magick::geometry, *26*

proj_get *(*proj_utils*)*, 11
proj_path *(*proj_utils*)*, 11
proj_set *(*proj_utils*)*, 11
proj_set(), *8*
proj_sitrep, 11, *12*
proj_utils, *11*, 11

reverse dependency checks, *17*

save(), *18*
spelling, *33*
spelling::wordlist, *33*
styler::tidyverse_style(), *13*

tidy_labels *(*use_github_labels*)*, 22
tidyverse, 13

use_apl2_license *(*licenses*)*, 10
use_appveyor *(*ci*)*, 6
use_badge *(*badges*)*, 3
use_binder_badge *(*badges*)*, 3
use_bioc_badge *(*badges*)*, 3
use_blank_slate, 14
use_build_ignore, 15
use_cc0_license *(*licenses*)*, 10
use_code_of_conduct, 15
use_course, 16
use_course(), *8*
use_course_details, *16*
use_coverage *(*ci*)*, 6
use_cran_badge *(*badges*)*, 3
use_cran_comments, 17
use_data, 17
use_data(), *19*
use_data_raw *(*use_data*)*, 17
use_description, 18
use_description(), *9*, *18*, *19*
use_description_defaults *(*use_description*)*, 18
use_dev_package *(*use_package*)*, 27

use_dev_version *(*use_version*)*, 37
use_directory, 19
use_directory(), *15*
use_git, 20, *24, 25*
use_git(), *21*
use_git_config, *20*, 24, *25*
use_git_hook, *20*, *24, 25*, 25
use_git_ignore, *20*, *24, 25*, 25
use_github, 21
use_github(), *7, 8*
use_github_labels, 22
use_github_links, 23
use_gpl3_license *(*licenses*)*, 10
use_lifecycle_badge *(*badges*)*, 3
use_logo, 26
use_mit_license *(*licenses*)*, 10
use_namespace, 26
use_news_md, 27
use_package, 27
use_package_doc, 28
use_package_doc(), *35*
use_pipe, 28
use_pkgdown, 29
use_r, 29
use_rcpp, 29
use_readme_md *(*use_readme_rmd*)*, 30
use_readme_rmd, 30
use_revdep, 31
use_rmarkdown_template, 31
use_roxygen_md, 32
use_rstudio, 32
use_rstudio(), *9*
use_spell_check, 33
use_template, 33
use_test *(*use_testthat*)*, 34
use_test(), *29*
use_testthat, 34
use_tibble, 35
use_tidy_ci *(*tidyverse*)*, 13
use_tidy_coc *(*tidyverse*)*, 13
use_tidy_contributing *(*tidyverse*)*, 13
use_tidy_description *(*tidyverse*)*, 13
use_tidy_eval *(*tidyverse*)*, 13
use_tidy_github *(*tidyverse*)*, 13
use_tidy_issue_template *(*tidyverse*)*, 13
use_tidy_style *(*tidyverse*)*, 13
use_tidy_support *(*tidyverse*)*, 13
use_tidy_thanks, 35
use_tidy_versions *(*tidyverse*)*, 13
use_travis *(*ci*)*, 6
use_usethis, 36
use_version, 37
use_vignette, 38
use_vignette(), *19*
utils::file.edit(), *17*, *27*, *30*, *33, 34*

whisker::whisker.render(), *34*