

Package ‘usethis’

April 7, 2019

Title Automate Package and Project Setup

Version 1.5.0

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://usethis.r-lib.org>, <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 (>= 1.2.0), purrr, rlang, rprojroot (>= 1.2), rstudioapi, stats, utils, whisker, withr, yaml

Suggests covr, knitr, magick, pkgdown (>= 1.1.0), rmarkdown, roxygen2, spelling (>= 1.2), styler (>= 1.0.2), testthat (>= 2.0.0)

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 6.1.1

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 2019-04-07 10:50:44 UTC

R topics documented:

badges	3
browse-this	4
browse_github_token	5
ci	7
create_from_github	8
create_package	9
edit	10
git_credentials	11
git_protocol	13
git_sitrep	14
git_vaccinate	14
licenses	15
proj_activate	16
proj_sitrep	16
proj_utils	17
pr_init	19
rprofile-helper	20
tidyverse	21
use_addin	23
use_blank_slate	23
use_build_ignore	24
use_citation	24
use_code_of_conduct	24
use_course	25
use_coverage	26
use_cran_comments	27
use_data	27
use_description	28
use_directory	29
use_git	30
use_github	31
use_github_labels	32
use_github_links	34
use_github_release	34
use_git_config	35
use_git_hook	36
use_git_ignore	36
use_git_remote	37
use_logo	38
use_namespace	39
use_news_md	39
use_package	40
use_package_doc	40
use_pipe	41
use_pkgdown	41
use_r	42

use_rcpp	42
use_readme_rmd	43
use_release_issue	44
use_revdep	44
use_rmarkdown_template	45
use_roxygen_md	45
use_rstudio	46
use_spell_check	46
use_template	47
use_testthat	48
use_tibble	49
use_tidy_thanks	49
use_tutorial	50
use_version	51
use_vignette	52

Index	54
--------------	-----------

badges	<i>README badges</i>
--------	----------------------

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()
```

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

```
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_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"),
  description = "R:GITHUB_PAT", host = "https://github.com")

browse_github_pat(scopes = c("repo", "gist"),
  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](https://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 `.Renv` as the `GITHUB_PAT` environment variable. `edit_r_extern()` 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_extern()
## 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)

```

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

```
use_appveyor(browse = interactive())
```

```
use_gitlab_ci()
```

Arguments

browse	Open a browser window to enable automatic builds for the package.
ext	which travis website to use. default to "org" for https://travis-ci.org. Change to "com" for https://travis-ci.com.

```
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_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_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.

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 = 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. 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. 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 immediately pulled from upstream, 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: <ul style="list-style-type: none"> • 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 <code>usethis.protocol</code> 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 <code>git2r::cred_env()</code> , <code>git2r::cred_ssh_key()</code> , <code>git2r::cred_token()</code> , or <code>git2r::cred_user_pass()</code> .
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".

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	<i>Create a package or project</i>
----------------	------------------------------------

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 = 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 <code>.here</code> file is placed so that the directory can be recognized as a project by the here or rprojroot packages.
open	If TRUE, activates the new project: <ul style="list-style-type: none"> • 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.

edit	<i>Open configuration files</i>
------	---------------------------------

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 = "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.

git_credentials	<i>Produce or register git credentials</i>
-----------------	--

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

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

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. 1. All the relevant software agrees on the definition of `~`, i.e. your home directory. This is harder than it sounds on Windows. 1. 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](#), 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	<i>Produce or register git protocol</i>
--------------	---

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

<code>protocol</code>	Optional. Should be "ssh" or "https", if specified. Defaults to the option <code>usethis.protocol</code> 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	<i>git/GitHub sitrep</i>
------------	--------------------------

Description

Get a situation report on your current git/GitHub status. Useful for diagnosing problems.

Usage

```
git_sitrep()
```

Examples

```
git_sitrep()
```

git_vaccinate	<i>Vaccinate your global git ignore</i>
---------------	---

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.
- **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_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	<i>Activate a project</i>
---------------	---------------------------

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	<i>Report working directory and usethis/RStudio project</i>
-------------	---

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

<code>path</code>	Path to set. This path should exist or be NULL.
<code>force</code>	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.
<code>...</code>	character vectors, if any values are NA, the result will also be NA.
<code>ext</code>	An optional extension to append to the generated path.
<code>code</code>	Code to run with temporary active project.
<code>quiet</code>	Whether to suppress user-facing messages, while operating in the temporary active project.
<code>.local_envir</code>	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)
```

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 <http://happygitwithr.com/>.

Usage

```
pr_init(branch)

pr_fetch(number, owner = NULL)

pr_push()

pr_pull()

pr_pull_upstream()

pr_sync()

pr_view()

pr_pause()

pr_finish()
```

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)` 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.

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)
```

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()
```

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

```
create_tidy_package(path, name = "RStudio")

use_tidy_ci(browse = interactive())

use_tidy_description()

use_tidy_versions(overwrite = FALSE, source = c("local", "CRAN"))

use_tidy_eval()

use_tidy_contributing()

use_tidy_issue_template()

use_tidy_support()

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 <code>options(usethis.full_name = "My name")</code> .
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.
source	Use "local" or "CRAN" package versions.
strict	Boolean indicating whether or not a strict version of styling should be applied. See <code>styler::tidyverse_style()</code> for details.

Details

- `create_tidy_package()`: creates a new package, immediately applies as many of the tidyverse conventions as possible, issues a few reminders, and activates the new package.
- `use_tidy_ci()`: sets up **Travis CI** and **Codecov**, ensuring that the package works on all versions of R starting at 3.1. It also ignores `compat-` and `deprec-` files from code coverage.
- `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_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** 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_addin	<i>Add minimal RStudio Addin binding</i>
-----------	--

Description

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

Usage

```
use_addin(addin = "new_addin", open = 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 <code>utils::file.edit()</code> otherwise.

use_blank_slate	<i>Don't save/load user workspace between sessions</i>
-----------------	--

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_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 <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 <code>use_directory()</code> . Default is to locate at top-level, but <code>.github/</code> is also common.
------	---

use_course	<i>Download course materials</i>
------------	----------------------------------

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. Links of the form "repo/user" are expanded to the appropriate url for a repo on GitHub. 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. A url of the form "repo/user", is assumed to refer to a GitHub repository and is expanded appropriately, unless url matches a URL shortener service, currently only "bit.ly" or "rstd.io". 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, 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)
```

use_coverage

Test coverage

Description

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

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	<i>CRAN submission comments</i>
-------------------	---------------------------------

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 <code>utils::file.edit()</code> otherwise.
------	---

use_data	<i>Create package data</i>
----------	----------------------------

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", version = 2)

use_data_raw(name = "DATASET", open = interactive())
```

Arguments

...	Unquoted names of existing objects to save.
internal	If FALSE, saves each object in its own <code>.rda</code> file in the <code>data/</code> directory. These data files bypass the usual export mechanism and are available whenever the package is loaded (or via <code>data()</code> if LazyData is not true). If TRUE, stores all objects in a single <code>R/sysdata.rda</code> file. Objects in this file follow the usual export rules. Note that this means they will be exported if you are using the common <code>exportPattern()</code> rule which exports all objects except for those that start with <code>..</code>

overwrite	By default, <code>use_data()</code> will not overwrite existing files. If you really want to do so, set this to <code>TRUE</code> .
compress	Choose the type of compression used by <code>save()</code> . 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 <code>utils::file.edit()</code> 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_description	<i>Create or modify a DESCRIPTION file</i>
-----------------	--

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"),  
                          comment = c(ORCID = "YOUR-ORCID-ID"))',  
    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	<i>Initialise a git repository</i>
---------	------------------------------------

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	<i>Connect a local repo with GitHub</i>
------------	---

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 account 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 use_this.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".

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_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 values.
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, will remove GitHub default labels that do not appear in the labels vector, and do not have associated issues. to your workflow).
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".

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.

- repress 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.
- 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)

# 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	<i>Configure Git</i>
----------------	----------------------

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	<i>Add a git hook</i>
--------------	-----------------------

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	<i>Tell git to ignore files</i>
----------------	---------------------------------

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_remotestypes()[["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_remotestypes()

# 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_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

<code>img</code>	The path to an existing image file
<code>geometry</code>	a <code>magick::geometry</code> string specifying size. The default assumes that you have a hex logo using spec from http://hexb.in/sticker.html .
<code>retina</code>	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_namespace	<i>Use a basic NAMESPACE</i>
---------------	------------------------------

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	<i>Create a simple NEWS.md</i>
-------------	--------------------------------

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 <code>utils::file.edit()</code> 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 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. There are two helper functions:

- `use_pkgdown()`: creates a pkgdown config file and adds the file and destination directory to `.Rbuildignore`.
- `use_pkgdown_travis()`: helps you set up pkgdown for automatic deployment on Travis-CI.

Usage

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

```
use_pkgdown_travis()
```

Arguments

<code>config_file</code>	Path to the pkgdown yaml config file
<code>destdir</code>	Target directory for pkgdown docs

See Also

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

use_r	<i>Create or edit a .R file</i>
-------	---------------------------------

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	<i>Use C, C++, RcppArmadillo, or RcppEigen</i>
----------	--

Description

Creates src/, adds required packages to DESCRIPTION, optionally creates .c or .cpp files, and where needed, Makevars and Makevars.win files.

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}.
------	--

use_readme_rmd	<i>Create README files</i>
----------------	----------------------------

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_release_issue	<i>Create a release issue checklist</i>
-------------------	---

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	<i>Reverse dependency checks</i>
------------	----------------------------------

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 = tolower(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	<i>Add RStudio Project infrastructure</i>
-------------	---

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	<i>Use spell check</i>
-----------------	------------------------

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	<i>Use a usethis-style template</i>
--------------	-------------------------------------

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

Create tests

Description

There are two helper functions:

- `use_testthat()` sets up overall testing infrastructure: creates `tests/testthat/`, `tests/testthat.R`, and adds `testthat` to `Suggests`.
- `use_test()` sets up individual test files: creates `tests/testthat/test-<name>.R` and, optionally, opens it for editing.

Usage

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

Arguments

<code>name</code>	Base of test file name. If <code>NULL</code> , and you're using RStudio, will be based on the name of the file open in the source editor.
<code>open</code>	Open the newly created file for editing? Happens in RStudio, if applicable, or via <code>utils::file.edit()</code> otherwise.

See Also

The [testing chapter](#) of *R Packages*.

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_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, interval = 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 = 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 <code>utils::file.edit()</code> 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	<i>Increment package version</i>
-------------	----------------------------------

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	<i>Create a vignette or article.</i>
--------------	--------------------------------------

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)
```

Index

activates, [8, 9](#)
active project, [16](#)

badges, [3](#)
browse-this, [4](#)
browse_cran (browse-this), [4](#)
browse_github (browse-this), [4](#)
browse_github_issues (browse-this), [4](#)
browse_github_pat
 (browse_github_token), [5](#)
browse_github_pulls (browse-this), [4](#)
browse_github_token, [5](#)
browse_github_token(), [8, 19](#)
browse_travis (browse-this), [4](#)

ci, [7](#)
create_from_github, [8](#)
create_package, [9](#)
create_package(), [28, 46](#)
create_project (create_package), [9](#)
create_project(), [46](#)
create_tidy_package (tidyverse), [21](#)

data(), [27](#)

edit, [10](#)
edit_git_config (edit), [10](#)
edit_git_ignore (edit), [10](#)
edit_r_buildignore (edit), [10](#)
edit_r_environ (edit), [10](#)
edit_r_environ(), [6](#)
edit_r_makevars (edit), [10](#)
edit_r_profile (edit), [10](#)
edit_rstudio_snippets (edit), [10](#)

fs::path(), [18](#)
fs::path_home(), [10](#)
functions that set up continuous
 integration services, [4](#)

gh::gh_whoami(), [6](#)

git2r::cred_env(), [8, 11, 31](#)
git2r::cred_ssh_key(), [8, 11, 12, 31](#)
git2r::cred_token(), [8, 11, 31](#)
git2r::cred_user_pass(), [8, 11, 12, 31](#)
git_credentials, [11](#)
git_credentials(), [9, 13, 19](#)
git_protocol, [13](#)
git_protocol(), [9](#)
git_remotes (use_git_remote), [37](#)
git_sitrep, [14](#)
git_vaccinate, [14](#)
github_token (browse_github_token), [5](#)
github_token(), [12, 31](#)

licenses, [15](#)
local_project (proj_utils), [17](#)

magick::geometry, [38](#)

pr_fetch (pr_init), [19](#)
pr_finish (pr_init), [19](#)
pr_init, [19](#)
pr_pause (pr_init), [19](#)
pr_pull (pr_init), [19](#)
pr_pull_upstream (pr_init), [19](#)
pr_push (pr_init), [19](#)
pr_sync (pr_init), [19](#)
pr_view (pr_init), [19](#)
proj_activate, [16](#)
proj_get (proj_utils), [17](#)
proj_path (proj_utils), [17](#)
proj_set (proj_utils), [17](#)
proj_sitrep, [16, 18](#)
proj_utils, [16, 17](#)

reverse dependency checks, [27](#)
rprofile-helper, [20](#)

save(), [28](#)
spelling, [46](#)
spelling::wordlist, [46](#)

- styler::tidyverse_style(), 22
- tidy_label_colours (use_github_labels), 32
- tidy_label_descriptions (use_github_labels), 32
- tidy_labels (use_github_labels), 32
- tidy_labels_rename (use_github_labels), 32
- tidyverse, 21
- use_addin, 23
- use_apl2_license (licenses), 15
- use_appveyor (ci), 7
- use_article (use_vignette), 52
- use_badge (badges), 3
- use_binder_badge (badges), 3
- use_bioc_badge (badges), 3
- use_blank_slate, 23
- use_build_ignore, 24
- use_c (use_rcpp), 42
- use_cc0_license (licenses), 15
- use_ccby_license (licenses), 15
- use_citation, 24
- use_code_of_conduct, 24
- use_conflicted (rprofile-helper), 20
- use_course, 25
- use_course(), 9
- use_course_details, 25, 26
- use_coverage, 26
- use_covr_ignore (use_coverage), 26
- use_cran_badge (badges), 3
- use_cran_comments, 27
- use_data, 27
- use_data(), 29
- use_data_raw (use_data), 27
- use_description, 28
- use_description(), 9, 28, 29
- use_description_defaults (use_description), 28
- use_dev_package (use_package), 40
- use_dev_version (use_version), 51
- use_devtools (rprofile-helper), 20
- use_directory, 29
- use_directory(), 25
- use_git, 30, 35, 36
- use_git(), 31
- use_git_config, 30, 35, 36
- use_git_credentials (git_credentials), 11
- use_git_hook, 30, 35, 36, 36
- use_git_ignore, 30, 35, 36, 36
- use_git_protocol (git_protocol), 13
- use_git_remote, 37
- use_github, 31
- use_github(), 8, 9
- use_github_labels, 32
- use_github_links, 34
- use_github_release, 34
- use_gitlab_ci (ci), 7
- use_gpl3_license (licenses), 15
- use_lgpl_license (licenses), 15
- use_lifecycle_badge (badges), 3
- use_logo, 38
- use_mit_license (licenses), 15
- use_namespace, 39
- use_news_md, 39
- use_package, 40
- use_package_doc, 40
- use_package_doc(), 49
- use_partial_warnings (rprofile-helper), 20
- use_pipe, 41
- use_pkgdown, 41
- use_pkgdown_travis (use_pkgdown), 41
- use_r, 42
- use_rcpp, 42
- use_rcpp_armadillo (use_rcpp), 42
- use_rcpp_eigen (use_rcpp), 42
- use_readme_md (use_readme_rmd), 43
- use_readme_rmd, 43
- use_release_issue, 44
- use_reprex (rprofile-helper), 20
- use_revdep, 44
- use_rmarkdown_template, 45
- use_roxygen_md, 45
- use_rstudio, 46
- use_rstudio(), 9
- use_spell_check, 46
- use_template, 47
- use_test (use_testthat), 48
- use_test(), 42
- use_testthat, 48
- use_tibble, 49
- use_tidy_ci (tidyverse), 21
- use_tidy_coc (tidyverse), 21

`use_tidy_contributing` (tidyverse), 21
`use_tidy_description` (tidyverse), 21
`use_tidy_eval` (tidyverse), 21
`use_tidy_github` (tidyverse), 21
`use_tidy_issue_template` (tidyverse), 21
`use_tidy_labels` (use_github_labels), 32
`use_tidy_release_test_env` (tidyverse),
21
`use_tidy_style` (tidyverse), 21
`use_tidy_support` (tidyverse), 21
`use_tidy_thanks`, 49
`use_tidy_versions` (tidyverse), 21
`use_travis` (ci), 7
`use_tutorial`, 50
`use_usethis` (rprofile-helper), 20
`use_version`, 51
`use_vignette`, 52
`use_vignette()`, 29
`utils::file.edit()`, 23, 27, 28, 39, 43, 47,
48, 51

`whisker::whisker.render()`, 47
`with_project` (proj_utils), 17