

Package: pkgtools (via r-universe)

September 20, 2026

Title Code generation and linting functions for R packages

Version 0.2.1

Description Perform common tasks and fix common errors in project and package development. This is a developer tool rather than an end user package.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3.9007

URL <https://terminological.github.io/pkgtools/index.html>,
<https://github.com/terminological/pkgtools>

BugReports <https://github.com/terminological/pkgtools/issues>

Imports desc, devtools, dplyr, forcats, fs, here, magrittr, pkgload, purrr, readr, remotes, stringi, stringr, tidyr, rlang, withr, rmarkdown, renv, jsonlite, usethis, cli, yaml, pins, gert, gh, shiny, bslib, glue, utils, whisker, roxygen2, rstudioapi, testthat, styler, rprojroot

Suggests spelling, urlchecker, clipr

Config/testthat/edition 3

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

Repository <https://terminological.r-universe.dev>

Date/Publication 2025-09-25 16:30:31 UTC

RemoteUrl <https://github.com/terminological/pkgtools>

RemoteRef 0.2.1

RemoteSha e72c1af813834b312e7aa9dd900dd1b2f0258604

Contents

bump_dev_version	2
document	3
find_and_replace	3
fix_check	4
fix_dependencies	4
fix_global_variables	5
fix_non_standard_files	5
fix_unqualified_fns	6
fix_unqualified_fns_bulk	6
fix_utf8_encoding	7
install_local	7
merge_code	9
migrate_to_pins	10
qcheck	11
roxy_tag_parse.roxy_tag_dparam	12
roxy_tag_parse.roxy_tag_unit	13
roxy_tag_rd.roxy_tag_dparam	13
roxy_tag_rd.roxy_tag_unit	14
run_commented_code	15
set_renv_repos	16
standalone_snapshot	16
style_text	17
switch_expect_equals	18
switch_standalone_snapshot	18
sync_standalone_to_master	18
undo	19
unstable	19
update_standalone	21
use_standalone	22
what_has_changed	24
Index	25

bump_dev_version	<i>Update the version of a package, incrementing dev versions.</i>
------------------	--

Description

This makes no checks and accepts no responsibility. No backups are made.

Usage

```
bump_dev_version(pkg = ".")
```

Arguments

pkg the path to the package

Value

the new version

document

Document package and non-package projects

Description

`devtools::document()` requires a DESCRIPTION file, which in general we don't include in analysis projects. Sometimes we want to produce development documentation for functions in analysis projects so we can describe options and behaviour. `pkgtools::document` will also work in a non package project to produce .Rd files in the man directory for any functions declared in the R sub-folder. It also override the default behaviour of `?` to return the development documentation of matching functions in the current project.

Usage

```
document(project.dir = ".", roclets = NULL, quiet = FALSE)
```

Arguments

<code>project.dir</code>	the directory of the current (non package project)
<code>roclets</code>	Character vector of roclet names to use with package. The default, <code>NULL</code> , uses the roxygen roclets option, which defaults to <code>c("collate", "namespace", "rd")</code> .
<code>quiet</code>	if <code>TRUE</code> suppresses output from this function.

Value

nothing

`find_and_replace`

Regex find and replace on a codebase

Description

Find and replace within a package after committing the current contents

Usage

```
find_and_replace(
  regex,
  replacement,
  pkg = ".",
  rDirectories = c("data-raw", "R", "vignettes", "tests")
)
```

Arguments

regex	a PCRE regex
replacement	a replacement
pkg	the package to scan
rDirectories	the subdirectories (defaults to data-raw, R, vignettes, tests)

Value

a list of changed files

fix_check	<i>Fix common check errors</i>
-----------	--------------------------------

Description

This function runs a set of common package development errors. It will create a commit before running checks and fixing unqualified functions, excluding non standard files, creating global variables, fixing utf8 encoding issues and ensuring all dependencies line up.

Usage

```
fix_check(pkg = ".")
```

Arguments

pkg	the package to fix
-----	--------------------

Value

nothing

fix_dependencies	<i>Fixes dependencies in the namespace file using the output of R CMD check.</i>
------------------	--

Description

Fixes dependencies in the namespace file using the output of R CMD check.

Usage

```
fix_dependencies(pkg = ".", check)
```

Arguments

pkg	the package to scan
check	output of a devtools:::check() command (will check automatically if not provided)

Value

nothing

fix_global_variables *Adds global variables identified at R CMD check“ to a globals.R‘ file*

Description

Adds global variables identified at R CMD check` to a globals.R‘ file

Usage

```
fix_global_variables(pkg = ".", check)
```

Arguments

pkg	the package location
check	the results of a devtools:::check command (will check automatically if not provided)

Value

nothing

fix_non_standard_files *Adds non standard and hidden files to the .Rbuildignore file*

Description

Adds non standard and hidden files to the .Rbuildignore file

Usage

```
fix_non_standard_files(pkg = ".", check)
```

Arguments

pkg	the package location
check	the results of a devtools::check command (will check automatically if not provided)

Value

nothing

fix_unqualified_fns	<i>Fix unqualified functions in active source pane</i>
---------------------	--

Description

Interactively find and replace unqualified, e.g. `mutate(...)` calls with fully qualified `dplyr::mutate(...)` calls. This version operates on the currently open file in the RStudio and allows for interactive review of the proposed changes.

Usage

```
fix_unqualified_fns()
```

Value

nothing - called for side effects

fix_unqualified_fns_bulk	<i>Fix unqualified namespaced functions.</i>
--------------------------	--

Description

This is a code linting function and expected to be called at the console during package development. It will scan the files in the current project and replace unqualified references to e.g. `mutate` with ones to `dplyr::mutate` etc. All changes are made after a commit which can be reverted.

Usage

```
fix_unqualified_fns_bulk(
  pkg = ".",
  rDirectories = c(here::here("R"), here::here("tests/testthat")),
  prioritise = c("dplyr", "rlang", "stringr", "forcats", "ggplot2", "purrr", "tidyr",
    "readr", "stats", "utils")
)
```

Arguments

pkg	the package
rDirectories	the locations of the R code to fix (by default R scripts, tests, but not data-raw or vignettes)
prioritise	a list of package names to pick from first

Value

nothing. called for side effects.

fix_utf8_encoding	<i>Fixes utf8 encoded characters in source files replaincg them with \uXXXX</i>
-------------------	---

Description

Fixes utf8 encoded characters in source files replaincg them with \uXXXX

Usage

```
fix_utf8_encoding(pkg = ".", check)
```

Arguments

pkg	the package to scan
check	output of a devtools:::check() command (will check automatically if not provided)

Value

nothing

install_local	<i>Install package locally using renv if available.</i>
---------------	---

Description

devtools::install_local does not play well with renv in this version of install_local we intercept installation of locally developed packages when we are in a renv managed project and installing a local dependency, it builds a source project into renv cellar and installs it from there. This allows a copy of a locally developed package to be deployed with the renv managed analysis project without specifically being deployed to CRAN or r-universe.

Usage

```
install_local(
  path = ".",
  ...,
  force = TRUE,
  upgrade = "never",
  quiet = TRUE,
  wd = here::here()
)
```

Arguments

path	the package (defaults to .)
...	Named arguments passed on to <code>remotes::install_local</code>
subdir	subdirectory within url bundle that contains the R package.
dependencies	Which dependencies do you want to check? Can be a character vector (selecting from "Depends", "Imports", "LinkingTo", "Suggests", or "Enhances"), or a logical vector. TRUE is shorthand for "Depends", "Imports", "LinkingTo" and "Suggests". NA is shorthand for "Depends", "Imports" and "LinkingTo" and is the default. FALSE is shorthand for no dependencies (i.e. just check this package, not its dependencies). The value "soft" means the same as TRUE, "hard" means the same as NA. You can also specify dependencies from one or more additional fields, common ones include: • Config/Needs/website - for dependencies used in building the pkgdown site. • Config/Needs/coverage for dependencies used in calculating test coverage.
build	If TRUE build the package before installing.
build_opts	Options to pass to R CMD build, only used when build is TRUE.
build_manual	If FALSE, don't build PDF manual ('-no-manual').
build_vignettes	If FALSE, don't build package vignettes ('-no-build-vignettes').
repos	A character vector giving repositories to use.
type	Type of package to update.
...	Other arguments passed on to <code>utils::install.packages()</code> .
force	do the install regardless of versions
upgrade	upgrade out of data CRAN packages
quiet	do it quietly
wd	the project root directory of the current project (defaults to <code>here::here()</code>)

Details

If installed locally for a non-renv project (e.g. a package development) the usual behaviour applies to version management. Installation of new versions of the project will happen when the package is released and then installed from the release location (e.g. github, cran, r-universe).

If a locally developed package is deployed to an renv project once it is released onto a valid distribution platform e.g. CRAN, r-universe or github, we will want to use that version in our renv. This we can do using the `rebuild = TRUE` option of `renv::install`, e.g.: `renv::install(...pkgname/github..., repo = ...r-universe?... , rebuild = TRUE)` followed by a `renv::snapshot()` to update the lock file. The locally built package version will remain in the `<projroot>/renv/local` cellar until removed by hand.

See Also

Other package installation: [install_bioc\(\)](#), [install_bitbucket\(\)](#), [install_cran\(\)](#), [install_dev\(\)](#), [install_github\(\)](#), [install_gitlab\(\)](#), [install_git\(\)](#), [install_svn\(\)](#), [install_url\(\)](#), [install_version\(\)](#)

Examples

```
## Not run:
dir <- tempfile()
dir.create(dir)
pkg <- download.packages("testthat", dir, type = "source")
install_local(pkg[, 2])

## End(Not run)
```

merge_code

Merge Code Versions using Shiny and CodeMirror 5 MergeView

Description

This function launches a Shiny gadget to visually merge two versions of code/YAML.

Usage

```
merge_code(
  value,
  new,
  old = NULL,
  lhs = "original",
  rhs = "suggested",
  accept = "Save",
  rhs_accept = "Accept all",
  mode = "r"
)
```

Arguments

value	the content for the central panel.
new	character vector of code lines. Content of the changed / RHS version.
old	character vector of code lines. Content of the old / LHS version.

lhs	the title of the LHS
rhs	the title of the RHS
accept	the button text for the LHS
rhs_accept	the button text for the RHS
mode	the language mode

Value

Character string of the merged content, or stops with an error if cancelled.

Examples

```
if (FALSE) {

  old = paste0("line ",c(2:4,6:12))
  new = paste0("line ",c(1:50))

  merged_result <- merge_code(old, new)
  cat("Merged Content:\n")
  cat(paste0(merged_result, collapse = "\n"),"\n")

}
```

migrate_to_pins

Migrate package data to a github hosted pin board

Description

This function makes multiple changes to the package. It adds packaged data files from the data directory to a pkgdown github published pin board and constructs a minimal set of functions with the same names as the data to access the data from the pinboard. It finds any references to the dataset in the code directories (data-raw,R,vignettes,test) and replaces them with a function call of the same name.

Usage

```
migrate_to_pins(pkg = ".", board = "data", migrate = NULL)
```

Arguments

pkg	the package (defaults to current)
board	the board name (this is generally not exposed to user)
migrate	either "*" for all data files, a list of names or if null the user will be asked interactively.

Value

nothing

qcheck

*Check the package structure without running any code***Description**

Check the package structure without running any code

Usage

```
qcheck(pkg = ".", ..., args = "", quiet = FALSE)
```

Arguments

`pkg` the path of the package to check

`...` Named arguments passed on to `devtools::check`

`document` By default (NULL) will document if your installed roxygen2 version matches the version declared in the DESCRIPTION file. Use TRUE or FALSE to override the default.

`build_args` Additional arguments passed to R CMD build

`...` Additional arguments passed on to `pkgbuild::build()`.

`manual` If FALSE, don't build and check manual (`--no-manual`).

`cran` if TRUE (the default), check using the same settings as CRAN uses. Because this is a moving target and is not uniform across all of CRAN's machine, this is on a "best effort" basis. It is more complicated than simply setting `--as-cran`.

`remote` Sets `_R_CHECK_CRAN_INCOMING_REMOTE_` env var. If TRUE, performs a number of CRAN incoming checks that require remote access.

`incoming` Sets `_R_CHECK_CRAN_INCOMING_` env var. If TRUE, performs a number of CRAN incoming checks.

`force_suggests` Sets `_R_CHECK_FORCE_SUGGESTS_`. If FALSE (the default), check will proceed even if all suggested packages aren't found.

`run_dont_test` Sets `--run-donttest` so that examples surrounded in `\donttest{}` are also run. When `cran = TRUE`, this only affects R 3.6 and earlier; in R 4.0, code in `\donttest{}` is always run as part of CRAN submission.

`env_vars` Environment variables set during R CMD check

`check_dir` Path to a directory where the check is performed. If this is not NULL, then the a temporary directory is used, that is cleaned up when the returned object is garbage collected.

`cleanup` **[Deprecated]** See `check_dir` for details.

`vignettes` If FALSE, do not build or check vignettes, equivalent to using `args = '--ignore-vignettes'` or `'--no-build-vignettes'`.

`error_on` Whether to throw an error on R CMD check failures. Note that the check is always completed (unless a timeout happens), and the error is only thrown after completion. If "never", then no errors are thrown. If "error",

then only ERROR failures generate errors. If "warning", then WARNING failures generate errors as well. If "note", then any check failure generated an error. Its default can be modified with the RCMDCHECK_ERROR_ON environment variable. If that is not set, then "never" is used.

path Path to built package.

args additional r cmd check args

quiet do it without producing messages

Value

a check result

roxy_tag_parse.roxy_tag_dparam
Parser for @dparam tags

Description

Replaces the default behaviour of a @param tag and documents default parameters automatically.

Usage

```
## S3 method for class 'roxy_tag_dparam'
roxy_tag_parse(x)
```

Arguments

x A tag

Value

a roxy_tag object with the val field set to the parsed value

Examples

```
# This provides support to `roxygen2` and only gets executed in the context
# of `devtools::document()`. There is no interactive use of this function.
```

```
roxy_tag_parse.roxy_tag_unit
    Parser for @unit tags
```

Description

The @unit tag can be used in roxygen2 documentation of a function to define unit testing code. This code will be written to a testthat file.

Usage

```
## S3 method for class 'roxy_tag_unit'
roxy_tag_parse(x)
```

Arguments

x A tag

Value

a roxy_tag object with the val field set to the parsed value

Examples

```
# This provides support to `roxygen2` and only gets executed in the context
# of `devtools::document()`. There is no interactive use of this function.
```

```
roxy_tag_rd.roxy_tag_dparam
    Support for @dparam tags
```

Description

Replaces the default behaviour of a @param tag and documents default parameters automatically.

Usage

```
## S3 method for class 'roxy_tag_dparam'
roxy_tag_rd(x, base_path, env)
```

Arguments

x The tag

base_path Path to package root directory.

env Environment in which to evaluate code (if needed)

Value

an `roxygen2::rd_section` (see `roxygen2` documentation)

Examples

```
# An example function definition:
fn_definition <- "
#' This is a title
#'
#' This is the description.
#'
#' @param x this parameter specifies a default
#'
f <- function(x = c("hello", "world")) {
  return("test")
}
"
```

```
# For this example we manually parse the function specification in `fn_definition`
# creating a .Rd block - normally this is done by `roxygen2` which then
# writes this to an .Rd file. This function is not intended to be used
# outside of a call to `devtools::document`.

tmp = roxygen2::parse_text(fn_definition)
print(tmp[[1]])
```

roxy_tag_rd.roxy_tag_unit

Support for @unit tags

Description

The `@unit` tag can be used in `roxygen2` documentation of a function to define unit testing code. This code will be written to a `testthat` file.

Usage

```
## S3 method for class 'roxy_tag_unit'
roxy_tag_rd(x, base_path, env)
```

Arguments

<code>x</code>	The tag
<code>base_path</code>	Path to package root directory.
<code>env</code>	Environment in which to evaluate code (if needed)

Value

an `roxygen2::rd_section` (see `roxygen2` documentation)

Unit tests

```
# for testing
warning("should be suppressed")
```

Examples

```
# An example function definition:
fn_definition <- "
#' This is a title
#'
#' This is the description.
#'
#' @md
#' @unit
#' # unit testing code here:
#' testthat::expect_equal(f(), \"test\")
#' @export
f <- function() {
  return(\"test\")
}
"

# For this example we manually parse the function specification in `fn_definition`
# creating a .Rd block - normally this is done by `roxygen2` which then
# writes this to an .Rd file. This function is not intended to be used
# outside of a call to `devtools::document`.

fn_definition = gsub("@", "@", fn_definition)
tmp = roxygen2::parse_text(fn_definition)
print(tmp[[1]])
```

run_commented_code	<i>Execute commented out code</i>
--------------------	-----------------------------------

Description

RStudio does this for examples but nothing else this allows singly commented code or code in other tags e.g. `@unit` to be executed

Usage

```
run_commented_code()
```

Details

This is installed as an RStudio add in and can be given a keyboard shortcut through Tools > Modify Keyboard Shortcuts...

Value

nothing

set_renv_repos	<i>Adds new repositories to the beginning of an renv lockfile</i>
----------------	---

Description

Sets custom repositories (e.g. r-universe repositories) in a renv lockfile to override CRAN repositories. This is a persistent change can be undone by manual editing of the lockfile.

Usage

```
set_renv_repos(..., .wd = here::here())
```

Arguments

...	a named list of repository urls
.wd	the working directory (defaults to here::here())

Value

nothing

standalone_snapshot	<i>Generate a hash based expectation for a standalone object</i>
---------------------	--

Description

testthat does not use hash based checking anymore, in favour of snapshots. These don't work well in tests of standalones because the file will move projects independently of the test directory. This function lets you quickly create a check based on a gold standard

Usage

```
standalone_snapshot(obj, .clip = interactive(), .as = deparse(substitute(obj)))
```

Arguments

obj	the gold standard to test
.clip	copy the result to the clipboard (needs clipr)
.as	what name to use for the comparison

Value

a code snippet that can be pasted into the unit test case

style_text	<i>Style code in pkgtools style</i>
------------	-------------------------------------

Description

This is adapted from <https://github.com/gadenbuie/grkstyle> It differs in that it will break apart long lines regardless of their initial line breaks to try and fit them into a set width. It does not unbreak lines.

Usage

```
style_text(text, ..., transformers = pkgtools_style_transformer(...))
```

Arguments

text	A character vector with text to style.
...	Named arguments passed on to styler::style_text
	... Arguments passed on to the style function, see tidyverse_style() for the default argument.
style	A function that creates a style guide to use, by default tidyverse_style . Not used further except to construct the argument transformers. See style_guides() for details.
include_roxygen_examples	Whether or not to style code in roxygen examples.
base_indention	Integer scalar indicating by how many spaces the whole output text should be indented. Note that this is not the same as splitting by line and add a base_indention spaces before the code in the case multi-line strings are present. See 'Examples'.
	Named arguments passed on to pkgtools_style_transformer
width	the max width of the code (excludes indent hence approximate)
transformers	A set of transformer functions. This argument is most conveniently constructed via the style argument and See 'Examples'.

Value

the formatted code

switch_expect_equals *Switch expression for equality based test case.*

Description

Switch expression for equality based test case.

Usage

switch_expect_equals()

Value

nothing

switch_standalone_snapshot
 Switch global variable for hash based test case.

Description

Switch global variable for hash based test case.

Usage

switch_standalone_snapshot()

Value

nothing

sync_standalone_to_master
 Synchronise master version of standalone file with editor contents

Description

This file syncs the current content of the rstudio editor window with a local master standalone file, which is assumed to be in a Git directory. This allows for changes to be propagated back to the master when in active development. The master copy is committed before changes.

Usage

sync_standalone_to_master(git_dir = fs::path_home("Git"))

Arguments

`git_dir` absolute or relative path to git base directory (e.g. ~/Git)

Value

nothing used for side effects

undo	<i>Undo a bulk pkgtools operation</i>
------	---------------------------------------

Description

Reverts to last state committed by pkgtools, and stashes any unstaged changes.

Usage

```
undo(pkg = ".")
```

Arguments

`pkg` the package

Value

nothing

unstable	<i>Reload a set of packages that are in development on the local machine</i>
----------	--

Description

Vignette building uses a new session. Any changes in current project or dependent locally developed projects are not tested unless the packages are all installed using `devtools::install_local(...)`. This causes problems when developing multiple packages in parallel.

Usage

```
unstable(
  path = ".",
  ...,
  force = TRUE,
  upgrade = "never",
  quiet = TRUE,
  load_lib = TRUE
)
```

Arguments

path	the package local development repository path. This assumes you have all your other package code in a sibling directory, e.g. ~/Git/pkg1, ~/Git/pkg2
...	Named arguments passed on to <code>remotes::install_local</code>
subdir	subdirectory within url bundle that contains the R package.
dependencies	Which dependencies do you want to check? Can be a character vector (selecting from "Depends", "Imports", "LinkingTo", "Suggests", or "Enhances"), or a logical vector. TRUE is shorthand for "Depends", "Imports", "LinkingTo" and "Suggests". NA is shorthand for "Depends", "Imports" and "LinkingTo" and is the default. FALSE is shorthand for no dependencies (i.e. just check this package, not its dependencies). The value "soft" means the same as TRUE, "hard" means the same as NA. You can also specify dependencies from one or more additional fields, common ones include: • Config/Needs/website - for dependencies used in building the pkgdown site. • Config/Needs/coverage for dependencies used in calculating test coverage.
upgrade	Should package dependencies be upgraded? One of "default", "ask", "always", or "never". "default" respects the value of the R_REMOTES_UPGRADE environment variable if set, and falls back to "ask" if unset. "ask" prompts the user for which out of date packages to upgrade. For non-interactive sessions "ask" is equivalent to "always". TRUE and FALSE are also accepted and correspond to "always" and "never" respectively.
force	Force installation, even if the remote state has not changed since the previous install.
quiet	If TRUE, suppress output.
build	If TRUE build the package before installing.
build_opts	Options to pass to R CMD build, only used when build is TRUE.
build_manual	If FALSE, don't build PDF manual ('-no-manual').
build_vignettes	If FALSE, don't build package vignettes ('-no-build-vignettes').
repos	A character vector giving repositories to use.
type	Type of package to update.
...	Other arguments passed on to <code>utils::install.packages()</code> .
force	Force installation, even if the remote state has not changed since the previous install.
upgrade	Should package dependencies be upgraded? One of "default", "ask", "always", or "never". "default" respects the value of the R_REMOTES_UPGRADE environment variable if set, and falls back to "ask" if unset. "ask" prompts the user for which out of date packages to upgrade. For non-interactive sessions "ask" is equivalent to "always". TRUE and FALSE are also accepted and correspond to "always" and "never" respectively.
quiet	If TRUE, suppress output.
load_lib	load the package using a library command

Details

This function assumes the path variable is a path to a package which is under version control in a Git directory. Other dependencies to this package may also be under development in sibling directories. The aim is to install the current version of the target package and all locally held dependencies that have changed on the local disk compared to the locally installed version.

This function scans the current package and first order dependencies, looking for local development directories for any packages imported. Looks for changes in files in local development directories of package and first order dependencies versus files currently installed in r-library. If it finds any differences it checks if there is a version change of the package, bumps the version number of the development package, and installs it locally. After installation it restarts R.

Any recent file change in development directories triggers a dev version bump and local package installation. After a call to `unstable()` any dependencies in your local dev environment are up to date.

If `unstable` is called from within a non package project which is using `renv` then rather than installing locally using `devtools` the package is built and deployed locally in the `renv` local package directory (`<proj root>/renv/local`) and installed from there. The `renv` local packages are placed under version control. At the moment it is a manual job to tidy this up once the development package is finalised and deployed

Value

nothing

update_standalone	<i>Update or create a standalone metadata block in the Rstudio editor.</i>
-------------------	--

Description

This function helps identify package imports and maintains standalone file metadata. It operates on the code in the Rstudio active window and suggests changes to be made to keep the standalone file metadata up to date. It is intended be used interactively.

Usage

```
update_standalone(repo = NULL, license = NULL, dependencies = NULL)
```

Arguments

repo	the github repository e.g. organisation/project
license	a license URL (defaults to http://unlicense.org)
dependencies	a optional list of standalone filenames in the same repository

Value

nothing, updates the RStudio editor content

use_standalone	<i>Extended version of use_standalone that works with analysis projects</i>
----------------	---

Description

usethis::use_standalone is a package development tool used in r-lib to share useful functions between packages without creating a hard dependency on them. This is also useful in data analysis projects where no package infrastructure exists but you want to reuse common functions (e.g. plot themes) between analysis projects. Developing a package containing these shared functions and deploying to CRAN or r-universe is possible but it is unwieldy and requires more infrastructure that needed.

Usage

```
use_standalone(
  repo_spec,
  file = NULL,
  ref = NULL,
  host = NULL,
  git_dir = fs::path_home("Git")
)
```

Arguments

repo_spec	A string identifying the GitHub repo in one of these forms: • Plain OWNER/REPO spec • Browser URL, such as "https://github.com/OWNER/REPO" • HTTPS Git URL, such as "https://github.com/OWNER/REPO.git" • SSH Git URL, such as "git@github.com:OWNER/REPO.git"
file	Name of standalone file. The standalone- prefix and file extension are optional. If omitted, will allow you to choose from the standalone files offered by that repo.
ref	The name of a branch, tag, or commit. By default, the file at path will be copied from its current state in the repo's default branch. This is extracted from repo_spec when user provides a URL.
host	GitHub host to target, passed to the .api_url argument of <code>gh::gh()</code>. If repo_spec is a URL, host is extracted from that. If unspecified, gh defaults to "https://api.github.com", although gh's default can be customised by setting the GITHUB_API_URL environment variable. For a hypothetical GitHub Enterprise instance, either "https://github.acme.com/api/v3" or "https://github.acme.com" is acceptable.
git_dir	The directory to the users collection of Git projects

Details

Using a standalone file we can develop these functions in a basic git repository with no deployment (with or without package infrastructure), and import them into an analysis project as standalone files. From a reproducibility point of view this is sometimes beneficial if the functions in question are fairly dynamic as the version is hard wired into the analysis project.

The use cases supported by `usethis::use_standalone` are predicated around R package development but here we extend this behaviour to analysis projects with dependencies managed by `renv`, or not managed at all. If an analysis project is being managed by `renv` then using a standalone file will install missing `renv` dependencies and snapshot the project. If no `renv` is detected a check is written to the `.RProfile` file which will produce a message about missing dependencies when the project is opened.

In a non package project directives to try and source all standalone files are added to the `.RProfile` file, so that standalone file functions are immediately available.

If this is not working you may need to set the repository to HTTP/2: `git config --global http.version HTTP/2 && git p`

Supported fields

A standalone file has YAML frontmatter that provides additional information, such as where the file originates from and when it was last updated. Here is an example:

```
---
repo: r-lib/rlang
file: standalone-types-check.R
last-updated: 2023-03-07
license: https://unlicense.org
dependencies: standalone-obj-type.R
imports: rlang (>= 1.1.0)
---
```

Two of these fields are consulted by `use_standalone()`:

- **dependencies:** A file or a list of files in the same repo that the standalone file depends on. These files are retrieved automatically by `use_standalone()`.
- **imports:** A package or list of packages that the standalone file depends on. A minimal version may be specified in parentheses, e.g. `rlang (>= 1.0.0)`. These dependencies are passed to `use_package()` to ensure they are included in the `Imports:` field of the `DESCRIPTION` file.

Note that lists are specified with standard YAML syntax, using square brackets, for example: `imports: [rlang (>= 1.0.0), purrr]`.

Examples

```
## Not run:
use_standalone("r-lib/rlang", file = "types-check")
use_standalone("r-lib/rlang", file = "types-check", ref = "standalone-dep")

## End(Not run)
```

`what_has_changed`*Compare content of editor with last saved version*

Description

I frequently end up getting an unsaved change in RStudio conflicting with an update on disk. Often as a result of a global find and replace. This helps determine which version to keep, by bringing up a 2 way diff of editor content and disk content. The result of the merge is placed back in the editor.

Usage

```
what_has_changed()
```

Value

nothing

Index

* **edit**

- find_and_replace, 3
- merge_code, 9
- run_commented_code, 15
- standalone_snapshot, 16
- switch_expect_equals, 18
- switch_standalone_snapshot, 18
- what_has_changed, 24

* **renv**

- document, 3
- install_local, 7
- set_renv_repos, 16
- unstable, 19

* **roxy**

- roxy_tag_parse.roxy_tag_dparam, 12
- roxy_tag_parse.roxy_tag_unit, 13
- roxy_tag_rd.roxy_tag_dparam, 13
- roxy_tag_rd.roxy_tag_unit, 14

* **standalone**

- sync_standalone_to_master, 18
- update_standalone, 21
- use_standalone, 22

* **usethis**

- bump_dev_version, 2
- fix_check, 4
- fix_dependencies, 4
- fix_global_variables, 5
- fix_non_standard_files, 5
- fix_unqualified_fns, 6
- fix_unqualified_fns_bulk, 6
- fix_utf8_encoding, 7
- migrate_to_pins, 10
- qcheck, 11
- undo, 19

bump_dev_version, 2

devtools::check, 11

document, 3

find_and_replace, 3

fix_check, 4

fix_dependencies, 4

fix_global_variables, 5

fix_non_standard_files, 5

fix_unqualified_fns, 6

fix_unqualified_fns_bulk, 6

fix_utf8_encoding, 7

gh::gh(), 22

install_bioc, 9

install_bitbucket, 9

install_cran, 9

install_dev, 9

install_git, 9

install_github, 9

install_gitlab, 9

install_local, 7

install_svn, 9

install_url, 9

install_version, 9

merge_code, 9

migrate_to_pins, 10

pkgbuild::build(), 11

pkgtools_style_transformer, 17

qcheck, 11

remotes::install_local, 8, 20

roxy_tag_parse.roxy_tag_dparam, 12

roxy_tag_parse.roxy_tag_unit, 13

roxy_tag_rd.roxy_tag_dparam, 13

roxy_tag_rd.roxy_tag_unit, 14

run_commented_code, 15

set_renv_repos, 16

standalone_snapshot, 16

style_guides(), 17

`style_text`, [17](#)
`styler::style_text`, [17](#)
`switch_expect_equals`, [18](#)
`switch_standalone_snapshot`, [18](#)
`sync_standalone_to_master`, [18](#)

`tidyverse_style`, [17](#)
`tidyverse_style()`, [17](#)

`undo`, [19](#)
`unstable`, [19](#)
`update_standalone`, [21](#)
`use_package()`, [23](#)
`use_standalone`, [22](#)
`utils::install.packages()`, [8](#), [20](#)

`what_has_changed`, [24](#)