Package: pkgtools (via r-universe)

September 19, 2024

Title Code generation and linting functions for R packages

Version 0.0.1

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 7.2.3

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, gert, here, magrittr, pkgload, purrr, readr, remotes, stringi, stringr, tibble, tidyr, rlang, withr, rmarkdown, renv, jsonlite, usethis

Suggests rstudioapi, diffobj

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

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

RemoteRef 0.0.1

RemoteSha 4d56d968f194ea9c432cc29af63e7b60b9af7ebc

Contents

`bump_dev_version` *Update the version of a package, incrementing dev versions.*

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

`delete_backups` *Delete backup files from the current project*

Description

Delete backup files from the current project

Usage

```
delete_backups(pkg = ".")
```

Arguments

`pkg`	the package to delete files from.

Value

nothing

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

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 ()

Value

a list of the

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

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

Value

nothing

fix_unqualified_fns *Fix unqualified functions in active source pane*

Description

Interactively find and replace unqualified, e.g. mutate(...) calls with fully qualified dplyr::mutate(...) calls.

Usage

```
fix_unqualified_fns()
```

Value

nothing - called for side effects

`fix_unqualified_fns_bulk` *Fix errors introduced in package creation by forgetting to qualify namespaces.*

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.

Usage

```
fix_unqualified_fns_bulk(
  pkg = ".",
  rDirectories = c(here::here("R"), here::here("tests/testthat")),
  dry_run = FALSE,
 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, and tests, but not vignettes)
`dry_run`	by default this function will not actually do anything unless this is set to FALSE. However the dry run output can be manually compared with a diff tool to interactively accept changes.
`prioritise`	a list of package names to pick from first

Value

nothing. called for side effects.

`fix_utf8_encoding` *Fixes utf8 encoded characters in source files replaincg them with* `\uXXXX`

Description

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

Usage

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

Arguments

pkg	the package to scan
check	output of a devtools::check() command ()
dry_run	test changes without breaking originals.

Value

nothing

---

install_local *Install package locally using* renv *if available.*

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	path to local directory, or compressed file (tar, zip, tar.gz tar.bz2, tgz2 or tbz)
...	Other arguments passed on to utils::install.packages().
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.
wd	the project root directory of the current project (defaults to here::here())

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 out renv. This we can do using the rebuild = TRUE option of renv::install, e.g.: renv::install(...pkg name/github..., repo = ...r-
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)
```

---

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
...	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

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' = '–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.

args additional r cmd check args

quiet do it without producing messages

Value

a check result

---

set_renv_repos *Adds new repositories to the beginning of an* renv *lockfile*

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

---

unstable *Reload a set of packages that are in development on the local machine*

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`
`...`	Arguments passed on to `remotes::install_local` `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`.

`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

use_standalone *Extended version of* use_standalone *that works with* renv *projects*

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 but it is unwieldy and requires more infrastructure that needed.

Usage

```
use_standalone(repo_spec, file = NULL, ref = NULL, host = NULL)
```

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 gh::gh(). 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.

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 a project as standalone files. From a reproducibility point of view this is sometimes beneficial as the version is hard wired into the analysis project.

The use cases supported by usethis are predicated around R package development but here we extend this behaviour to analysis projects with dependencies managed by renv.

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]`.


[use_package()]: R:use_package()
[rlang (>= 1.0.0), purrr]: R:rlang%20(%3E=%201.0.0),%20purrr
```

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

Compare content of editor with last saved version

Usage

```
what_has_changed()
```

Value

nothing

Index