Package 'srcpkgs'

April 20, 2026

Title R Source Packages Manager

Version 0.2.1

Description Manage a collection/library of R source packages. Discover, document, load, test source packages. Enable to use those packages as if they were actually installed. Quickly reload only what is needed on source code change. Run tests and checks in parallel.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/kforner/srcpkgs

BugReports https://github.com/kforner/srcpkgs/issues

Imports cli, clitable, devtools, pkgload, testthat, stats, utils

Suggests knitr, rmarkdown, withr

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Karl Forner [aut, cre, cph]

Maintainer Karl Forner <karl.forner@gmail.com>

Repository CRAN

Date/Publication 2026-04-20 13:00:02 UTC

Contents

srcpkgs-package 2
find_srcpkgs 3
get_srcpkgs 3
hack_r_loaders 4
pkgs_check 5
pkgs_deps 6
pkgs_install 7
pkgs_test 8

pkg_check 9
pkg_list_attached 10
pkg_load 10
pkg_roxygenise 12
pkg_test 13
pkg_unload 14
reset 15
settings 15
setup_and_get_dummy_srcpkg 16
srcpkgs 17
unhack_r_loaders 17

Index **19**

`srcpkgs-package` *srcpkgs: R Source Packages Manager*

Description

Manage a collection/library of R source packages. Discover, document, load, test source packages. Enable to use those packages as if they were actually installed. Quickly reload only what is needed on source code change. Run tests and checks in parallel.

Features

`srcpkgs` main objective is to ease development on any project that uses a collection of R source packages (a library). It is able to figure out which dependencies are source packages, and is able to quickly detect changes in any of the used source packages.

Author(s)

Maintainer: Karl Forner <karl.forner@gmail.com> [copyright holder]

See Also

Useful links:

- https://github.com/kforner/srcpkgs
- Report bugs at https://github.com/kforner/srcpkgs/issues

find_srcpkgs *finds all available source packages starting from the project root*

Description

N.B: the *hidden* files and directories are ignored. In general, this function is not used directly, instead you should use `get_srcpkgs()`

Usage

```
find_srcpkgs(
  root = get_project_root(),
  srcpkgs_paths = find_srcpkgs_paths(root, prune = prune),
  prune = TRUE
)
```

Arguments

`root`	directory from where to search for source packages
`srcpkgs_paths`	paths to the source packages folders
`prune`	whether to report packages contained inside another package (e.g. in tests/)

Value

a "srcpkgs" object (or NULL if none found), a named list of "srcpkg" objects, that essentially are devtools "package" objects. The list is named after the package names.

Examples

```
pkg <- setup_and_get_dummy_srcpkg()
pkgs <- find_srcpkgs(dirname(pkg$path))
print(pkgs)
```

get_srcpkgs *get the current source packages list*

Description

The first call to this function will trigger the initialization of the package ((cf `reset()`). Since it is used by mostly all user-facing load-related functions, this enables a runtime initialization, as opposed to a load-time initialization. So for example you may load `srcpkgs`, then change the current directory to your project. Then the first load will setup the settings.

Usage

```
get_srcpkgs(filter = NULL)
```

Arguments

filter	a pattern to filter the source packages

Details

For optimization, the paths to discovered source packages are cached (cf `reset()` and `settings()`. This function will reparse the DESCRIPTION for any change. If you add or delete a source package, you must reset the source package paths using `reset()`

This function is useful for troubleshooting, to understand what are the source packages discovered and managed by `srcpkgs`

Value

the source packages as a "scrpkgs" object, cf `find_srcpkgs()`, or NULL if none

Examples

```
# setup a srcpkg. We need reset because it is not discoverable from the current directory
pkg <- setup_and_get_dummy_srcpkg()
reset(dirname(pkg$path))

print(get_srcpkgs())
```

---

hack_r_loaders *instruments the R loaders to make them aware of source packages*

Description

hacks `library()` and `loadNamespace()` using the base R tracer function `trace()`. `library(pkg)` will basically call `pkg_load(pkg)` if the source package `pkg` is managed by **srcpkgs**

Usage

```
hack_r_loaders()
```

Details

N.B: usually you do not need to call that function explicitly. The function is reentrant.

Value

no return value, called for side-effects

Package startup

At package startup (actually `.OnAttach()`), `hack_r_loaders()` will be automatically called to hack the R loaders, UNLESS this is inhibited via the option `srcpkgs.inhibit_r_loaders_hack` or the environment variable `SRCPKGS.INHIBIT_R_LOADERS_HACK`. You may set any value like TRUE, "TRUE", 1 or "1".

See Also

unhack_r_loaders()

Examples

```
## Not run:
# hack library
hack_r_loaders()

# unhack
unhack_r_loaders()

# prevent automatic hacking when srcpkgs is loaded
options(srcpkgs.inhibit_r_loaders_hack=TRUE)
# or
Sys.setenv(SRCPKGS.INHIBIT_R_LOADERS_HACK="1")
library(srcpkgs)

## End(Not run)
```

---

pkgs_check *checks a list of source packages*

Description

checks a list of source packages

Usage

```
pkgs_check(
  pkgids = names(filter_srcpkgs(src_pkgs, filter)),
  src_pkgs = get_srcpkgs(),
  filter = NULL,
  lib = ".check",
  quiet = FALSE,
  fail_on_error = FALSE,
  ...
)
```

Arguments

pkgids	a list of package ids (names, paths or object), or a srcpkgs object. Also accept a singleton package object
src_pkgs	a collection of source packages as a srckgs object.
filter	filter out the packages to check using this pattern
lib	directory where to install and find installed pkgs
quiet	whether to be quiet/silent

fail_on_error	whether to die if there is at least an error or warning in the checks
...	passed to pkg_check

Value

the results as a pkgs_test object

---

pkgs_deps *computes the dependencies of some (source) packages*

Description

computes the dependencies of some (source) packages

Usage

```
pkgs_deps(
  pkgids,
  src_pkgs = get_srcpkgs(),
  source = TRUE,
  installed = TRUE,
  imports = TRUE,
  depends = TRUE,
  suggests = TRUE,
  reverse = FALSE
)
```

Arguments

pkgids	a list of package ids (names, paths or object), or a srcpkgs object. Also accept a singleton package object
src_pkgs	a collection of source packages as a srckgs object.
source	whether to report source packages
installed	whether to report installed (non-source) packages
imports	whether to only consider imports dependencies
depends	whether to only consider depends dependencies
suggests	whether to only consider suggests dependencies
reverse	whether to compute reverse dependencies instead

Value

the dependencies, as a character vector, topologically sorted

Examples

```
pkg <- setup_and_get_dummy_srcpkg()
deps_src <- pkgs_deps(pkg, installed = FALSE)
deps_inst <- pkgs_deps(pkg, source = FALSE)
print(get_srcpkgs())
deps_rev <- pkgs_deps(pkg, reverse = TRUE, suggests = FALSE)
```

---

pkgs_install *installs a list of source packages*

Description

- A source package can not be installed if its dependencies are not.
- Will not reinstall packages if they are up-to-date
- will roxygenise packages if needed

Usage

```
pkgs_install(
  pkgids,
  lib,
  src_pkgs = get_srcpkgs(),
  only_deps = FALSE,
  quiet = TRUE,
  ...
)
```

Arguments

pkgids	a list of package ids (names, paths or object), or a srcpkgs object. Also accept a singleton package object
lib	directory where to install and find installed pkgs
src_pkgs	a collection of source packages as a srckgs object.
only_deps	whether not to include pkgids, only their dependencies.
quiet	whether to be quiet/silent
...	passed to devtools::install()

Value

the names of the packages actually installed

pkgs_test *tests a list of source packages*

Description

tests a list of source packages

Usage

```
pkgs_test(
  pkgids = names(filter_srcpkgs(src_pkgs, filter)),
  src_pkgs = get_srcpkgs(),
  filter = NULL,
  quiet = TRUE,
  ...
)
```

Arguments

pkgids	a list of package ids (names, paths or object), or a srcpkgs object. Also accept a singleton package object
src_pkgs	a collection of source packages as a srckgs object.
filter	filter out the packages to test using this pattern
quiet	whether to be quiet/silent
...	passed to pkg_test

Value

the results as a pkgs_test object

Examples

```
## create a dummy collection of srcpkgs by replicating the dummy srcpkg
 pkg <- setup_and_get_dummy_srcpkg()
 pkgs <- srcpkgs(list(pkg, pkg))

 res <- pkgs_test(pkgs, error_on = "never")
 print(res)
```

pkg_check *tests a package - runs R CMD check*

Description

This function will check a source package.

Usage

```
pkg_check(
  pkgid,
  src_pkgs = get_srcpkgs(),
  lib = ".check",
  roxygen = TRUE,
  quiet = FALSE,
  error_on = "error",
  check_system_clock = FALSE,
  ...
)
```

Arguments

pkgid	a package name, path or package object
src_pkgs	a collection of source packages as a srckgs object.
lib	directory where to install and find installed pkgs
roxygen	whether to roxygenize
quiet	whether to be quiet/silent
error_on	passed to devtools::check()
check_system_clock	if FALSE, disable the _R_CHECK_SYSTEM_CLOCK_ check. This check sometimes fail because of firewalls...
...	passed to devtools::check()

Value

the results as a pkg_test object, or NULL if no tests found

pkg_list_attached *lists the packages that are attached, i.e. present in the R search() path*

Description

lists the packages that are attached, i.e. present in the R search() path

Usage

```
pkg_list_attached()
```

Value

the names of attached package name as a character vector

Examples

```
print(sort(pkg_list_attached()))
```

pkg_load *loads or reloads if needed a source package, taking care of its dependencies*

Description

N.B: the defaults are different from `devtools::load_all()`: the helpers are not loaded, only the functions tagged as *exported* are actually exported. The intended goal is to make it as similar to the behaviour of the R loaders.

Usage

```
pkg_load(
  pkgid,
  src_pkgs = get_srcpkgs(),
  attach = TRUE,
  suggests = FALSE,
  roxygen = TRUE,
  helpers = FALSE,
  export_all = FALSE,
  quiet = FALSE,
  dry_run = FALSE,
  ...
)
```

Arguments

pkgid	a package name, path or package object
src_pkgs	a collection of source packages as a srckgs object.
attach	Whether to attach a package environment to the search path. If FALSE load_all() behaves like loadNamespace(). If TRUE (the default), it behaves like library(). If FALSE, the export_all, export_imports, and helpers arguments have no effect.
suggests	whether to load suggested packages. if TRUE, the suggested are processed like imports
roxygen	whether to automatically roxygenise packages (if needed)
helpers	if TRUE loads **testthat** test helpers.
export_all	If TRUE (the default), export all objects. If FALSE, export only the objects that are listed as exports in the NAMESPACE file.
quiet	whether to be quiet/silent
dry_run	whether not to actually execute any action having side-effects
...	Arguments passed on to devtools::load_all
	path Path to a package, or within a package.
	reset **[Deprecated]** This is no longer supported because preserving the namespace requires unlocking its environment, which is no longer possible in recent versions of R.
	recompile DEPRECATED. force a recompile of DLL from source code, if present. This is equivalent to running pkgbuild::clean_dll() before load_all()

Details

This the workhorse function of the package, called by library() and loadNamespace() when hacked (cf hack_r_loaders().

This function will check that all dependent packages are up-to-date, and document and reload them as needed.

To be able to properly load a package, its dependent source packages must be loaded in proper order. i.e. if A–>B–>C, the load order must be C, B, A

Value

the load plan as a data frame, or NULL if there is nothing to do.

Examples

```
root <- tempfile()
pkg <- setup_and_get_dummy_srcpkg(root)
reset(root)
# load and attach a package
pkg_load(pkg)
```

```
# just load, do not attach it (~ loadNamespace())
pkg_unload(pkg)
pkg_load(pkg, attach = FALSE)

# do some changes, to a source package or any of its depencies or dependents
pkg_unload(pkg)
plan <- pkg_load(pkg, dry_run = TRUE)
# then you can inspect the plan actions
```

---

pkg_roxygenise *roxygenize a source package if needed*

Description

- if the package has not changed (based on the md5sum file), does nothing
- otherwise roxygenise the package using roxygen2::roxygenise
- and update and save the new md5sum file

Usage

```
pkg_roxygenise(pkg_path, force = FALSE, quiet = FALSE, ...)
```

Arguments

pkg_path	the package path, as a character
force	if force(d), do not use the md5-based system to detect package changes
quiet	whether to be quiet/silent
...	passed to devtools::document()

Details

- N.B: has the side-effect of loading the package

Value

if the roxygenation has been performed

Examples

```
pkg <- setup_and_get_dummy_srcpkg()
pkg_roxygenise(pkg$path)
```

pkg_test *tests a package - runs its unit tests*

Description

This function will test a source package using `testthat`, making sure the package and its source package dependencies are up-to-date and loaded

Usage

```
pkg_test(
  pkgid,
  filter = NULL,
  src_pkgs = get_srcpkgs(),
  export_all = TRUE,
  quiet = TRUE,
  ...
)
```

Arguments

`pkgid`	a package name, path or package object
`filter`	filter in the tests to run. cf `testthat::test_dir()`
`src_pkgs`	a collection of source packages as a `srckgs` object.
`export_all`	passed to `pkg_load()`. Enables the test functions to easily access to non-exported functions. Caveat: If the pkg is already loaded and up-to-date with export_all=FALSE, it will not work.
`quiet`	whether to be quiet/silent
`...`	passed to `testthat::test_dir()`

Value

the results as a `pkg_test` object, which is an empty listL if no tests were found

Examples

```
root <- tempfile()
pkg <- setup_and_get_dummy_srcpkg(root)
reset(root)
res <- pkg_test(pkg)
print(res)
```

pkg_unload *unloads a package, unloading its dependent packages if needed*

Description

To be able to unload properly a package, all the packages that depend even indirectly on it should be unloaded first.

Usage

```
pkg_unload(
  pkg_or_name,
  src_pkgs = get_srcpkgs(),
  dry_run = FALSE,
  loaded = loadedNamespaces(),
  quiet = FALSE
)
```

Arguments

pkg_or_name	a package name or object ("package" or "srcpkg")
src_pkgs	a collection of source packages as a srckgs object.
dry_run	whether not to actually execute any action having side-effects
loaded	the loaded packages, useful for testing.
quiet	whether to be quiet/silent

Details

N.B: this function also works for non source packages.

Value

a data frame of the unloaded package names, and whether they were attached, invisibly or NULL if the package is not loaded

Examples

```
root <- tempfile()
pkg <- setup_and_get_dummy_srcpkg(root)
reset(root)
pkg_load(pkg)

pkg_unload(pkg)
```

reset *resets the* srcpkgs *settings*

Description

With this function, you can reset or set precisely the settings.

Usage

```
reset(root = find_project_root(), srcpkgs_paths = find_srcpkgs_paths(root))
```

Arguments

root	directory from where to search for source packages
srcpkgs_paths	paths to the source packages folders

Value

the settings (cf settings()) invisibly

Examples

```
# reset to appropriate defaults based on your current directory
old <- reset()

# explictly set the project root
reset(root = tempdir())

# explictly set the source package paths (very unlikely)
reset(srcpkgs_paths = c('pkgs/mypkg1', 'pkgs/mypkg2'))

# restore previous settings
reset(root = old$root, srcpkgs_paths = old$srcpkgs_paths)
```

settings *informs about the settings currently used by* srcpkgs

Description

informs about the settings currently used by srcpkgs

Usage

```
settings()
```

Value

a named list of:

- initialized: whether the settings are initialized (as triggered by `get_srcpkgs()`)
- root: the project root
- srcpkgs_paths: the paths of the source packages to manage
- hack_r_loaders_installed: whether the R loaders are hacked
- hack_r_loaders_enabled: whether the R loaded hack is in action (internal use)

Examples

```
print(settings())
```

---

setup_and_get_dummy_srcpkg

installs the dummy srcpkg in a temp location

Description

Intended for testing and to write examples

Usage

```
setup_and_get_dummy_srcpkg(dest = tempfile())
```

Arguments

`dest`	where to install the dummy srcpkg

Value

the package as a `srcpkg` object

Examples

```
pkg <- setup_and_get_dummy_srcpkg()
print(pkg)
```

srcpkgs *creates a new "srcpkgs" object*

Description

creates a new "srcpkgs" object

Usage

```
srcpkgs(pkgs = lapply(paths, devtools::as.package), paths = NULL)
```

Arguments

pkgs	an existing srcpkgs object (no op), or a list of source package-like objects
paths	a list of source package paths as a character vector or list

Value

a srcpkgs object (a list named after the package names)

Examples

```
# build dummy source packages
pkg1 <- setup_and_get_dummy_srcpkg()
pkg2 <- pkg1
pkg2$package <- "dummy.srcpkg2"

print(srcpkgs(list(pkg1, pkg2)))
print(srcpkgs(paths = pkg1$path))
```

unhack_r_loaders *untraces library() and loadNamespace()*

Description

The function is reentrant.

Usage

```
unhack_r_loaders()
```

Value

no return value, called for side-effects

See Also

hack_r_loaders()

Examples

```
## Not run:
unhack_r_loaders()

## End(Not run)
```

Index

devtools::document(), *12*
devtools::load_all, *11*
devtools::load_all(), *10*

find_srcpkgs, 3
find_srcpkgs(), *4*

get_srcpkgs, 3
get_srcpkgs(), *3*, *16*

hack_r_loaders, 4
hack_r_loaders(), *11*, *18*

library(), *11*
loadNamespace(), *11*

pkg_check, 9
pkg_list_attached, 10
pkg_load, 10
pkg_load(), *13*
pkg_roxygenise, 12
pkg_test, 13
pkg_unload, 14
pkgbuild::clean_dll(), *11*
pkgs_check, 5
pkgs_deps, 6
pkgs_install, 7
pkgs_test, 8

reset, 15
reset(), *3, 4*

settings, 15
settings(), *4*, *15*
setup_and_get_dummy_srcpkg, 16
srcpkgs, 17
srcpkgs-package, 2

unhack_r_loaders, 17
unhack_r_loaders(), *5*