Package 'srcpkgs'

May 15, 2024

Title R Source Packages Manager

Version 0.1

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 7.3.1

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

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

Imports cli, devtools, pkgload

Suggests knitr, rmarkdown, testthat (>= 3.0.0), 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 2024-05-15 19:50:02 UTC

R topics documented:

`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

```
find_srcpkgs('.')
```

---

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()
```

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

```
pkgs <- get_srcpkgs()
print(pkgs)
```

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

```
# 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)
```

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 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` clear package environment and reset file cache before loading any pieces of the package. This largely equivalent to running `unload()`, however the old namespaces are not completely removed and no `.onUnload()` hooks are called. Use `reset = FALSE` may be faster for large code bases, but is a significantly less accurate approximation.

`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

```
## Not run:
# load and attach a package
pkg_load('mypkg')

# just load, do not attach it (~ loadNamespace())
pkg_load('mypkg', attach = FALSE)

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



## End(Not run)
```

---

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

```
plan <- pkg_unload('mypkg')
```

---

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
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'))
```

`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 use0

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()

Index