Package: capsule (via r-universe)

July 7, 2024

Title A streamlined inversion of `renv`

Version 0.4.2

Description A `capsule` is a stable local package library that you consciously choose to execute code within. Think of it as representing 'production', while your normal interactive R session represents 'development'.

License MIT + file LICNSE

Encoding UTF-8

LazyData true

Imports callr, renv (>= 0.7.0.125), withr, using (>= 0.4.0), fs, jsonlite, stats, utils

Remotes anthonynorth/using

RoxygenNote 7.1.2

Roxygen list(markdown = TRUE)

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

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

RemoteUrl https://github.com/milesmcbain/capsule

RemoteRef master

RemoteSha 401d0c98adc329c17d0bb129069c9ec220a26646

Contents

`any_local_behind_lockfile` *check if any local packages are behind lockfile*

Description

A wrapper for get_local_behind_lockfile that returns TRUE if any dependencies found in `dep_source_paths` are behind the lockfile version in `lockfile_path`

Usage

```
any_local_behind_lockfile(
  lockfile_path = "./renv.lock",
  dep_source_paths = NULL
)
```

Arguments

`lockfile_path`	a length one character vector path of the lockfile for
`dep_source_paths`	a character vector of file paths to extract package dependencies from. If NULL (default) the whole local library is compared.

Value

TRUE if dev packages are behind lockfile, FALSE otherwise.

See Also

Other comparisons: `compare_local_to_lockfile()`, `get_local_behind_lockfile()`

capshot *Quickly generate an renv compliant lock file*

Description

These functions generate json lockfiles that can be restored from using `capsule` or `renv`.

Usage

```
capshot(
  dep_source_paths = "./packages.R",
  lockfile_path = "./renv.lock",
  minify = FALSE
)

capshot_str(dep_source_paths = "./packages.R", minify = FALSE)
```

Arguments

`dep_source_paths`	files to scan for project dependencies to write to the lock file.
`lockfile_path`	output path for the lock file.
`minify`	a boolean value indicicating if lockfile JSON should have whitespace removed to shrink footprint.

Details

Unlike `create()` this function does not populate a local library. It writes a lock file using dependencies found in files in `dep_source_paths`. Package dependency information is mined from DESCRIPTION files using the current `.libPaths()`.

These functions do not use `{renv}` machinery and so may produce different results. They have been re-implmented for speed, so that they can be integrated into automated pipelines that build projects or documents.

Value

`lockfile_path`. Writes lockfile as a side effect.

Functions

- `capshot_str`: a variation that returns lockfile json as a character vector for further use.

`compare_local_to_lockfile` *compare the local R library with the lockfile*

Description

Get a summary dataframe comparing package versions in the lockfile with versions in the local R library (.libPaths()) or capsule library (./renv).

Usage

```
compare_local_to_lockfile(lockfile_path = "./renv.lock")

compare_capsule_to_lockfile(lockfile_path = "./renv.lock")
```

Arguments

`lockfile_path`	a length one character vector path of the lockfile for

Value

a summary dataframe of version differences

Functions

- `compare_capsule_to_lockfile`: compares the renv libray to the lockfile

See Also

Other comparisons: `any_local_behind_lockfile()`, `get_local_behind_lockfile()`

Other comparisons: `any_local_behind_lockfile()`, `get_local_behind_lockfile()`

`create` *create*

Description

Create a capsule library context to run code in

Usage

```
create(dep_source_paths = "./packages.R", lockfile_path = "./renv.lock")
```

Arguments

`dep_source_paths`	files to find package dependencies in.

Details

Dependencies to be encapsulated are detected from files you nominate in `dep_source_paths`. Good practice would be to have a single dependencies R file that contains all library() calls - hence this makes an explicit assertion of your dependencies. This way spurious usages of pkg:: for packages not stated as dependencies will cause errors that can be caught.

Value

nothing. Creates a capsule as a side effect.

Author(s)

Miles McBain

`delete` *delete*

Description

Delete the capsule

Usage

```
delete()
```

Details

Removes the lockfile and library, in the case that you made a mistake or no longer want to use capsule.

Value

nothing

Author(s)

Miles McBain

`delete_local_lib` *delete_local_lib*

Description

Delete the capsule's local library

Usage

```
delete_local_lib()
```

Details

This helper is provided to help you recover from mistakes or test building the library from a lockfile you have generated.

Value

nothing

Author(s)

Miles McBain

See Also

`delete()`

`delete_lockfile` *delete_lockfile*

Description

Delete the capule's lockfile

Usage

```
delete_lockfile()
```

Details

This helper is provided to help you recover from mistakes or test extracting dependencies.

Value

nothing

Author(s)

Miles McBain

`detect_dependencies` *Detect dependencies in nominated R or Rmd files.*

Description

Get the names of R packages referred to in `file_path`. `file_path` can be a vector of paths if you need, although I advise keeping dependency calls in a single file for R projects.

Usage

```
detect_dependencies(file_path)
```

Arguments

`file_path`	the file(s) to detect dependencies in.

Details

This is a thin wrapper around `[renv::dependencies()]` that includes support for the `[using::pkg()]` style specification via `[using::detect_dependencies()]`

Value

a character vector of package names

`dev_mirror_lockfile` *Mirror lockfile in local library*

Description

Install packages contained in the lockfile that are either missing from the local library or at a lower version number.

Usage

```
dev_mirror_lockfile(
  lockfile_path = "./renv.lock",
  dep_source_paths = NULL,
  prompt = interactive()
)
```

Arguments

`lockfile_path`	lockfile to be compared to local environment.
`dep_source_paths`	R files to search for dependencies, the set of packages to be updated is limited to these dependencies (and their dependencies).
`prompt`	ask for confirmation after displaying list to be installed and before isntalling?

Details

Packages are installed at the lockfile version. Packages in the local library that are ahead of the the local library are not touched.

So this function ensures that the local development environment is **at least** at the lockfile version of all packages, not **equal to**.

To find differences between the local library and the lockfile versions use `compare_local_to_lockfile()`.

Value

names of the packages updated or to be updated (if install did not proceed) invisibly.

`get_local_behind_lockfile` *get packckes behind lockfile*

Description

return information on packages in your main R library (`.libPaths()`) or capsule library (`./renv`) that are behind the lockfile versions (at `lockfile_path`).

Usage

```
get_local_behind_lockfile(
  lockfile_path = "./renv.lock",
  dep_source_paths = NULL
)

get_capsule_behind_lockfile(
  lockfile_path = "./renv.lock",
  dep_source_paths = NULL
)
```

Arguments

`lockfile_path`	a length one character vector path of the lockfile for
`dep_source_paths`	a character vector of file paths to extract package dependencies from. If NULL (default) the whole local library is compared.

Details

if dep_source_paths is supplied only dependencies declared in these files are returned.

Information is returned about packages that are behind in your development environment, so you can update them to the capsule versions if you wish.

A warning is thrown in the case that pacakges have the same version but different remote SHA. E.g. A package in one library is from GitHub and in the other library is from CRAN. Or Both packages are from GitHub, have the same version but different SHAs.

Value

a summary dataframe of package version differences.

Functions

- get_capsule_behind_lockfile: get packages in the renv library that are behind the lockfile

See Also

Other comparisons: any_local_behind_lockfile(), compare_local_to_lockfile()

Other comparisons: any_local_behind_lockfile(), compare_local_to_lockfile()

Examples

```
## Not run:
get_local_behind_capsule(
  dep_source_paths = "./packages.R",
  lockfile_path = "./renv.lock"
)

## End(Not run)
```

---

recreate *recreate*

Description

Recreate a capsule with new dependencies

Usage

```
recreate(dep_source_paths = "./packages.R")
```

Arguments

dep_source_paths
: a character vector of project source files to extract dependencies from.

Details

After some development work has been completed, Use this function to update the capsule environment to match the dependency versions in your development environment.

Similarly to `create()`, you are expected to supply a vector of files in your project to extract dependencies from. Things work best when this is a single file containing only dependency related code.

Value

nothing. The capsule is regenerated as a side effect.

Author(s)

Miles McBain

See Also

`create()`

`repl` *repl*

Description

Open a REPL within the capsule

Usage

```
repl()
```

Details

Uses an experimental feature from `callr` to attach a new process repl to your current interactive session. That REPL evaluates code within the context of your capsule.

To exit the process send the use the interrupt signal in the REPL e.g. Control-C, or, ess-interrupt, or the 'stop' button in rstudio.

Depending on your R editor, overtaking your REPL with a new process may cause strang behaviour, like the loss of autocompletions.

Value

nothing.

Author(s)

Miles McBain

reproduce_lib *reproduce_lib*

Description

Reproduce the capsule library from the lockfile

Usage

```
reproduce_lib()
```

Details

If you have cloned a project that contains a lockfile you can actually just use `run()` to execute commands in the capsule and have the library built automatically. If that is not convenient, this will explicitly create the capsule library from the lockfile.

Value

nothing.

Author(s)

Miles McBain

run_callr *run functions in the capsule*

Description

run function in a new process in the capsule

Usage

```
run_callr(func, show = TRUE, ...)

run(code)
```

Arguments

`func`	a function to run in the capsule context, as per the `callr::r()` interface.
`...`	additional arguments passed to `callr::r()`
`code`	the body of function to be run in the capsule context. See Details.

Details

Execute the supplied function in the context of the capsule library using `callr::r`. This ensures code is run in a new R process that will not be contaminated by the state of the interactive development environment.

Value

output of `func`

Lockfile

At a minimum, an `renv` lockfile must be present in the current working directory. The capsule library will be generated from the lockfile if it does not exist. Use `create()` to make the lockfile.

Details

`run` is a more convenient interface to `run_callr`, which inserts the `code` argument into the body of a function, to be run in a child R process. The code is passed through to the function body using non-standard evaluation. If edge cases arise due to non-standard evaluation, prefer `run_callr`.

Author(s)

Miles McBain

See Also

`callr::r()` for detailed calling semantics, `create()` to make the lockfile. `run()` for a lighter weight alternative.

Examples

```
## Not run:
run_callr(function() {library(tidyverse)})
run(library(tidyverse))
By default rmarkdown::render looks into the .GlobalEnv:
run_session(rmarkdown::render("./analysis.Rmd"))

## End(Not run)
```

`run_rscript` *Run an R script in a new process in the capsule*

Description

Execute the supplied R script in the context of the capsule library using `callr::rscript()`. This ensures the script is executed in a new R process that will not be contaminated by the state of the interactive development environment and will use the R packages and versions in the capsule.

Usage

```
run_rscript(path, ..., show = TRUE)
```

Arguments

path	The path to the R script
...	Arguments passed on to callr::rscript

script Path of the script to run.

cmdargs Command line arguments.

libpath The library path.

repos The repos option. If NULL, then no repos option is set. This options is only used if user_profile or system_profile is set FALSE, as it is set using the system or the user profile.

stdout Optionally a file name to send the standard output to.

stderr Optionally a file name to send the standard error to. It may be the same as stdout, in which case standard error is redirected to standard output. It can also be the special string "2>&1", in which case standard error will be redirected to standard output.

poll_connection Whether to have a control connection to the process. This is used to transmit messages from the subprocess to the parent.

echo Whether to echo the complete command run by rcmd.

callback A function to call for each line of the standard output and standard error from the child process. It works together with the show option; i.e. if show = TRUE, and a callback is provided, then the output is shown of the screen, and the callback is also called.

block_callback A function to call for each block of the standard output and standard error. This callback is not line oriented, i.e. multiple lines or half a line can be passed to the callback.

spinner Whether to show a calming spinner on the screen while the child R session is running. By default it is shown if show = TRUE and the R session is interactive.

system_profile Whether to use the system profile file.

user_profile Whether to use the user's profile file. If this is "project", then only the profile from the working directory is used, but the R_PROFILE_USER environment variable and the user level profile are not. See also "Security considerations" below.

env Environment variables to set for the child process.

timeout Timeout for the function call to finish. It can be a base::difftime object, or a real number, meaning seconds. If the process does not finish before the timeout period expires, then a system_command_timeout_error error is thrown. Inf means no timeout.

wd Working directory to use for running the command. Defaults to the current working directory.

fail_on_status Whether to throw an R error if the command returns with a non-zero status code. By default no error is thrown.

color Whether to use terminal colors in the child process, assuming they are active in the parent process.

show Logical, whether to show the standard output on the screen while the child process is running. Note that this is independent of the stdout and stderr arguments. The standard error is not shown currently.

Value

Invisibly returns the result of callr::rscript()

See Also

run_callr() for more details relevant to run_rscript(), callr::r() for detailed calling semantics, create() to make the lockfile. run() for a lighter weight alternative.

---

run_session *run_session*

Description

run code in the context of the capsule in the current R session

Usage

```
run_session(code)
```

Arguments

code an expression to run in the context of the capsule library.

Details

Execute the supplied function in the context of the capsule library, by changing the R library paths it searches.

In almost all cases, run or run_callr which do effectively the same thing, are preferred. This is because the code argument can cause packages to be attached, and thus not read from the capsule library.

For example if code was drake::r_make() this would cause drake, to compatibility issues.

Use this function when you have R code that modifies the .GlobalEnv, and you want to inspect it at the end, or you want to actively debug with #' browser() or recover(). Even then it may be preferrable to use capsule::repl() to do debugging.

Value

output of code

Lockfile

At a minimum, an `renv` lockfile must be present in the current working directory. The capsule library will be generated from the lockfile if it does not exist. Use `create()` to make the lockfile.

Author(s)

Miles McBain

See Also

`create()` to make the lockfile. `run_callr()` and `run()` for safer versions.

Examples

```
## Not run:
run(library())
run(search())
 capsule::run({
   search()
   message("hello")
 })

## End(Not run)
```

---

whinge *Complain if the local R library has packages that are behind the lockfile versions*

Description

Useful for keeping teams loosely in sync on package versions. A warning can be tolerated until updating at a convenient time. For example if placed in the packages.R file of a `{tflow}` project.

Usage

```
whinge(whinge_fun = warning, lockfile_path = "./renv.lock")
```

Arguments

`whinge_fun`	the function to use to have a whinge about packages, e.g. message, warning, stop, etc.
`lockfile_path`	the path to the project lockfile

Details

The message is hardcoded, but the whinge_fun that takes the message is customisable.

Value

output of whinge_fun, most likely nothing.

Index