Package 'sourcoise'

May 23, 2025

Type Package

Title Source a Script and Cache

Version 0.6.0

Description Provides a function that behaves nearly as base::source() but implements a caching mechanism on disk, project based.
It allows to quasi source() R scripts that gather data but can fail or consume to much time to respond even if nothing new is expected.
It comes with tools to check and execute on demand or when cache is invalid the script.

License MIT + file LICENSE

Encoding UTF-8

URL https://xtimbeau.github.io/sourcoise/,
https://github.com/xtimbeau/sourcoise

Depends R (>= 4.1.0)

Imports fs, qs2, cli, purrr, digest, dplyr, lubridate, tibble,
jsonlite, lobstr, stringr, glue, rprojroot, rlang, scales,
logger

Suggests knitr, insee, memoise, quarto, bench, testthat (>= 3.0.0)

VignetteBuilder quarto

Config/testthat/edition 3

RoxygenNote 7.3.2

SystemRequirements Quarto command line tools
(https://github.com/quarto-dev/quarto-cli).

BugReports https://github.com/xtimbeau/sourcoise/issues

NeedsCompilation no

Author Xavier Timbeau [aut, cre, cph]

Maintainer Xavier Timbeau <xavier.timbeau@sciencespo.fr>

Repository CRAN

Date/Publication 2025-05-23 10:20:02 UTC

Contents

---

set_sourcoise_root   *Force root*

Description

`sourcoise()` mechanism to find root of the project automatically can be bypassed. This function is equivalent to setting the `sourcoise.root` option, except for storage of cache at the level of the file. To allow thus behaviour, root should be set to `NULL`.

Usage

```
set_sourcoise_root(root = NULL, quiet = TRUE)
```

Arguments

root   (default `NULL`, character) path of the root

quiet   (default `TRUE` boolean) displays messages

Value

root set (character)

Examples

```
dir <- tempdir()
set_sourcoise_root(dir)
```

---

sourcoise	*sources R script and caches results on disk*

Description

sourcoise() is used as a drop in replacement for base::source() but caches results on disk. Cache is persistant over sessions.

Usage

```
sourcoise(
  path,
  args = list(),
  track = list(),
  lapse = getOption("sourcoise.lapse"),
  force_exec = getOption("sourcoise.force_exec"),
  prevent_exec = getOption("sourcoise.prevent_exec"),
  metadata = getOption("sourcoise.metadata"),
  wd = getOption("sourcoise.wd"),
  quiet = TRUE,
  inform = FALSE,
  priority = 10
)
```

Arguments

path	(character) path of the script to execute (see details).
args	(list) list of args that can be used in the script (in the form args$xxx).
track	(list) list of files which modification triggers cache invalidation and script execution .
lapse	(character) duration over which cache is invalidated. Could be never (default) x hours, x days, x week, x months, x quarters, x years.
force_exec	(boolean) execute code, disregarding cache valid or invalid.
prevent_exec	(boolean) prevent execution, cache valid or not, returned previous cached data, possibly invalid.
metadata	(boolean) if TRUE sourcoise() returns a list with data is the $data and various meta data (see details).
wd	(character) if project working directory for the execution of script will be the root of the project. If file then it will be the dir of the script (default) If qmd, then working dir will be the dir in which the calling qmd is. Current directory is restored after execution (successful or failed).
quiet	(boolean) mute messages and warnings from script execution.
inform	(boolean) Display logs on console, even if logging is disabled with threshold level "INFO".
priority	(integer) (default 10) can be used as a way to control the order of execution when refreshing data (see sourcoise_refresh())

Details

sourcoise() looks like base::source(). However, there are some minor differences.

First, the script called in sourcoise() must end by a return() or by an object returned. Assignment made in the script won't be kept as sourcoise() is executed locally. Only explicitly reruned object will be returned. So soucoise() is used by assigning its result to something (aa <- sourcoise("mon_script.r) or sourcoise() |> ggplot() ...). Unless specified otherwise with wd parameter, the working directory for the script execution is (temporarly) set to the dir in which is the script. That allows for simple access to companion files and permit to move the script and companion files to another dir or project.

Second, an heuristic is applied to find the script, in the event the path given is incomplete. Whereas it is not advised and comes with a performance cost, this can be useful when there is a change in the structure of the project. The heuristic is simple, the script is searched inside the porject dir and among all hits the closest to the caller is returned.

Third, if an error is triggered by the script, sourcoise() does not fail and return the error and a NULL return. However, if there is a (invalid or valid) cache, the cached data is returned allowing for the script to continue. In that case the error is logged.

Cache is invalidated when : 1 - a cache is not found 2 - the script has been modified 3 - tracked files have been modified 4 - last execution occurred a certain time ago and is considered as expired 5 - execution is forced

Whatever values takes src_in, if the file path starts with a /, then the source file will be interpreted from project root (if any). This is coherent whith naming convention in quarto. Otherwise, the document path wil be used firstly (if any, that is to say executed from quarto, rendering). Finally, working directory will be used. If everything fails, it will try to search in the project directory a corresponding file and will keep the closest from the calling point.

Usually the fisrt call return and cache the results. Results can be aby R object and are serialized and saved using qs2. Subsequent calls, supposing none of cache invalidation are true, are then very quick. No logging is used, data is fecteched from the cache and that's it. For standard size data, used in a table or a graph (< 1Mb roughly), return timing is under 5ms.

lapse parameter is used for invalidation trigger 4. lapse = "1 day" ou lapse="day" for instance will trigger once a day the execution. lapse = "3 days" will do it every 72h. hours, weeks, months, quarters or years are understood time units. MOre complex calendar instructions could be added, but sourcoise_refesh() provides a solution more general and easy to adapt to any use case, as to my knowledge, there is no general mechanism to be warned of data updates.

track is the trigger #3. It is simply a list of files (following path convention defined by scr_in, so either script dir of project dir as reference). If the files in the list are changed then the execution is triggered. It is done with a hash and it is difficult to have a croo plateform hash for excel files. Nevertheless, hash is done on text files with same results of different platforms.

Value

data (list ou ce que le code retourne)

Global options

In order to simplify usage and to avoid complex bugs, some parameters can be set only globally, through options().

- sourcoise.root (character) force root, and bypass soucroise mechanism to find root. Useful when you want to execute sourcoise in a non-project context (see examples). sourcoise.src_in (character) if project stores the cache folder (.sourcoise) at the project root, if file, .sourcoise is stored at the calling point.

- sourcoise.nocache (boolean) no caching, so makes sourcoise less useful, can be used for testing purpose

- sourcoise.log (default "OFF") log threshold (see logger::log_treshold()).

- sourcoise.grow_cache (integer) (default 5 par défaut) cache limit in number of data file kept.

- sourcoise.limit_mb (integer) (default 50) individual cache data files size on disk limit. If above **no caching** occurs.

Metadata

If metadata=TRUE, a list is returned, with some metadatas. Main ones are $data, the data returned, $date, execution date, $timing execution timing, $size of the R object in memory, $data_file, $data_date and $file_size documenting data file path, date size on disk and last modification date, parameters of the call ($track, $wd, $src_in, $args and so on).

force_exec and prevent_exec are parameters that force the script execution (trigger #5) of prevent it (so cache is returned or NULL if no cache). Those 2 parameters can be set for one specific execution, but they are intendend to a global setting through the option sourcoise.force_exec or sourcoise.prevent_exec.

If returned data after execution is not different than previously cached data, then no caching occurs in order to limit the disk use and to avoid keeping an history of the same data files. This implies the possibility of a difference between last execution date and last data modification date. If you are insterested in the moment data was changed, then $data_date is to be preferred.

Working with github

sourcoise() is designed to function with *github*. Cache information is specific to each user (avoiding conflicts) and cached data is named with the hash. Conflicts could occur in the rare case the same script is executed on different machines and that this script return each time a different result (such as a random generator).

See Also

Other sourcoise: sourcoise_clear(), sourcoise_refresh(), sourcoise_reset(), sourcoise_status()

Examples

```
dir <- tempdir()
set_sourcoise_root(dir)
fs::file_copy(
  fs::path_package("sourcoise", "some_data.R"),
  dir,
  overwrite = TRUE)
# Force execution (root is set explicitly here, it is normally deduced from project)
data <- sourcoise("some_data.R", force_exec = TRUE)
```

```
# The second time cache is used
data <- sourcoise("some_data.R")


# Performance and mem test
dir <- tempdir()
set_sourcoise_root(dir)
fs::file_copy(
    fs::path_package("sourcoise", "some_data.R"),
    dir,
    overwrite = TRUE)
bench::mark(
 forced = data <- sourcoise("some_data.R", force_exec = TRUE),
 cached = data <- sourcoise("some_data.R"),
 max_iterations = 1)
```

---

sourcoise_clear   *Cleans sourcoise cache*

Description

removes every json and qs2 files found by sourcoise_status() unless a specific tibble (filtered from sourcoise_status()) is passed as an argument.

Usage

```
sourcoise_clear(
  what = sourcoise_status(root = root, prune = FALSE),
  root = NULL
)
```

Arguments

what   (–) a tibble such as the one obtained by sourcoise_status(), possibly filtered

root   to force root, not recommended (expert use)

Value

list of cleared files, plus a side-effect as specified cache files are deleted (no undo possible)

See Also

Other sourcoise: sourcoise(), sourcoise_refresh(), sourcoise_reset(), sourcoise_status()

Examples

```
dir <- tempdir()
set_sourcoise_root(dir)
fs::file_copy(
    fs::path_package("sourcoise", "some_data.R"),
    dir,
    overwrite = TRUE)
# Force execution
data <- sourcoise("some_data.R", force_exec = TRUE)
# we then clear all caches
sourcoise_clear()
sourcoise_status()
```

---

sourcoise_meta   *Returns sourcoise metadata on a script*

Description

quick acces to metadata of the script, data is not fecthed.

Usage

```
sourcoise_meta(path, args = NULL)
```

Arguments

path	(character) path of the script
args	(named list) arguments of the script if any

Details

- `timing`: time of full script execution
- `date`: date of last full execution
- `size`: size of objects returned (in R memory)
- `args`: args given to sourcoise for the script
- `lapse`: dely before reexecution
- `track`: list of files tracked
- `qmd_file`: list of qmd calling this script
- `log_file`: last log file
- `file_size`: size of data cached on disk
- `data_date`: date of last data save (if no new data when executed, no data is saved)
- `data_file`: path to data cached (as a qs2 data file)
- `file`: path to the json file storing metadata (and .sourcoise dir)

Value

a named list with cache information

Examples

```
dir <- tempdir()
set_sourcoise_root(dir)
fs::file_copy(
   fs::path_package("sourcoise", "some_data.R"),
  dir,
  overwrite = TRUE)
# Force execution (root is set explicitly here, it is normally deduced from project)
data <- sourcoise("some_data.R", force_exec = TRUE)
# Then we access metadata
sourcoise_meta("some_data.R")
```

---

sourcoise_refresh   *Refresh sourcoise cache by executing sources selected*

Description

All scripts (passed to `sourcoise_refresh()`) are executed with logging enabled.

Usage

```
sourcoise_refresh(
  what = NULL,
  force_exec = TRUE,
  unfreeze = TRUE,
  quiet = FALSE,
  init_fn = getOption("sourcoise.init_fn"),
  root = getOption("sourcoise.root"),
  log = "INFO",
  .progress = TRUE
)
```

Arguments

what	(tibble) a tibble as generated by `sourcoise_status()`, possibly filtered, (defaut to `source_status()`)
force_exec	(boolean) (default FALSE) if TRUE code is executed, no matter what is cached
unfreeze	(boolean) (default TRUE) when possible, unfreeze and uncache .qmd files in a quarto project when data used by those .qmd has been refreshed
quiet	(boolean) (default FALSE) no message if TRUE
init_fn	(function) (default NULL) execute a function before sourcing to allow initialization

root	(default NULL) force root to be set, instead of letting the function finding the root, for advanced uses
log	(character) (default "INFO") log levels as in `logger::log_threshold()` (c("OFF", "INFO", ...)), comes with a small performance cost
.progress	(boolean) (default TRUE) displays a progression bar based on previous execution timings

Details

The function returns the list of script executed but its main effect is a side-effect as scripts are executed and caches updates accordingly. Note also that log files reflect execution and track possible errors. Because of logging the execution comes with a loss in performance, which is not an issue if scripts are long to execute.

It is possible to execute `sourcoise_refresh()` without execution forcing (`force_exec=FALSE`) or with it. Forced execution means that the script is executed even if the cache is valid. In the case of non forced execution, execution is triggered by other cache invalidation tests (change in source file, lapse or tacked files).

When scripts are linked to qmds (i.e. when run in a quarto project), it is possible to unfreeeze and uncache those qmds with the option `unfreeze=TRUE`. This allows to refresh the cahe and then render the qmds using the new data.

It is possible to pass to refresh a function that will be executed before every script. This allows to load packages and declare global variables that can be used in each script. If packages are loaded inside the script, then this is not needed.

Parameters registered ins `sourcoise_status()` such as `wd` or `args` are used to execute the script.

Defining a `priority` in `sourcoise()`, will change the order of execution of refresh. This allows to execute first data that will be used then inside another script with `sourcoise()`. When a `sourcoise("something.r", force_exec=TRUE)` is run, inside calls to `sourcoise()` are *not* forced. `priority` is a crude way – yet efficient – to achieve a correct execution with up to date data. Of course, all calls to `sourcoise()` of the same script should set the same level of priority as the last executed will be kept, with no verification done, so left to the responsability of the user.

Value

a list of r scripts (characters) executed, with timing and success and a side effect on caches

See Also

Other sourcoise: [sourcoise](), [sourcoise_clear](), [sourcoise_reset](), [sourcoise_status]()

Examples

```
dir <- tempdir()
set_sourcoise_root(dir)
fs::file_copy(
   fs::path_package("sourcoise", "some_data.R"),
   dir,
   overwrite = TRUE)
# Force execution
```

```
data <- sourcoise("some_data.R", force_exec = TRUE)
# we then refresh all caches
sourcoise_refresh()
```

---

sourcoise_reset   *Resets sourcoise*

Description

Removes all .sourcoise folders found under the project root.

Usage

```
sourcoise_reset(root = NULL)
```

Arguments

root   to force root (expert use)

Value

No return, effect is through removal of .sourcoise folders (this is a side effect, no undo possible)

See Also

Other sourcoise: sourcoise(), sourcoise_clear(), sourcoise_refresh(), sourcoise_status()

Examples

```
dir <- tempdir()
set_sourcoise_root(dir)
fs::file_copy(
   fs::path_package("sourcoise", "some_data.R"),
   dir,
   overwrite = TRUE)
data <- sourcoise("some_data.R", force_exec = TRUE)
sourcoise_reset()
```

---

sourcoise_status *Cache status of sourcoise*

Description

Given the current project, soucoise_status() collects all information about cache (could be project level, file level) and return a tibble with this data.

Usage

```
sourcoise_status(quiet = TRUE, root = NULL, prune = TRUE, clean = FALSE)
```

Arguments

quiet	(boolean) (default TRUE) no messages during execution
root	(string) (default NULL) force root to a defined path, advanced and not recommanded use
prune	(boolean) (default TRUE) clean up status to display only on relevant cache. However, does not clean other cache files.
clean	(boolean) (default FALSE) check if some data files have not json referring to them and cleans if any.

Details

sourcoise_status() reflects what is on the disk (and results indeed from a scan of all cached files and their metadatas). So modifying the result of sourcoise_status() can produce complex bugs when it is passed to sourcoise_refresh() or sourcoise_clean().

Data returned is:

- src: path to the source file (r script)
- date: last execution date
- valid: is cache valid ?
- uid: id of user
- index: index of cache
- timing: last execution timing
- size: size of the R object(s) returned
- lapse: periodic refresh trigger
- wd: wd setting for execution of r script
- args: arguments passed to R script
- json_file: path to the file keeping cache information
- qmd_file: list of path to qmd files calling this script (relevant only for quarto projects)
- src_in: localisaiton of cache option

- data_file: path to data cached
- data_date: date and time of last save of data
- log_file: path to log file, if logging activated
- root: path to the project root, used as reference for all paths
- scr_hash: hash of the source file
- track_hash: hash of the tracked files, if any
- track: list of files tracked
- args_hash: hash of arguments
- data_hash: hash of data cached

Value

tibble of cached files (see details for structure)

See Also

Other sourcoise: sourcoise(), sourcoise_clear(), sourcoise_refresh(), sourcoise_reset()

Examples

```
dir <- tempdir()
set_sourcoise_root(dir)
fs::file_copy(
    fs::path_package("sourcoise", "some_data.R"),
    dir,
    overwrite = TRUE)
# Force execution
data <- sourcoise("some_data.R", force_exec = TRUE)
# status returns the cache status
sourcoise_status()
```

Index