Package 'sourcoise'

January 7, 2026

Type Package

Title Source a Script and Cache

Version 1.1.0

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

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

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

Depends R (>= 4.2.0)

Imports cli, digest, dplyr, fs, glue, jsonlite, knitr, lobstr, logger, lubridate, memoise, purrr, qs2, RcppSimdJson, rlang, rprojroot, stringr, tibble, tidyr, utils

Suggests quarto, bench, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

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

NeedsCompilation no

Author Xavier Timbeau [aut, cre, cph]

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

Repository CRAN

Date/Publication 2026-01-07 19:50:02 UTC

Contents

`set_sourcoise_root` *Set the Root Directory for Sourcoise*

Description

This function allows you to manually set the root directory for the sourcoise package, bypassing the automatic root detection mechanism used by `sourcoise()`. Setting the root directory affects where sourcoise looks for files and stores cache data.

Usage

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

Arguments

`root`	Path to the desired root directory. If `NULL` (default), sourcoise will attempt to automatically detect the project root. Can be an absolute or relative path.
`quiet`	Logical value indicating whether to suppress messages during root detection. Default is `TRUE` (messages suppressed).

Details

By default, sourcoise automatically detects the project root. This function is equivalent to setting the `sourcoise.root` option directly, except when dealing with file-level cache storage. To enable file-level cache storage behavior, set root to `NULL`.

Value

The root path that was set (character string), invisibly returned by `try_find_root()`.

Examples

```
# Set root to a temporary directory
dir <- tempdir()
set_sourcoise_root(dir)

# Reset to automatic detection
set_sourcoise_root(NULL)

# Set root with messages enabled
set_sourcoise_root(dir, quiet = FALSE)
```

`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 and can be shared through github.

Usage

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

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. Default to `NULL`wich means no change in tracking.
`lapse`	(character) duration over which cache is invalidated. Could be `NULL` (ie no change), `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".

Details

sourcoise() looks like base::source(). However, there are some 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 returned 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 attempts to retrun a cache, even invalid. However, if there is no (invalid or valid) cache, then an error is trigerred.

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 the option sourcoise.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 returns and caches the results. Results can be any R object and are serialized and saved using qs2. Subsequent calls, supposing none of cache invalidation conditions 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 10ms on a decent computer.

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. When lapse is defined for a script it will be used as long as a new lapse parameter is provided (such as "never" if you whish to stop expiration control) 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 from websites.

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. As for lapse, tracked files are accumulated over execution and a null track will not change the list of tracked files. To untrack use the sourcoise_untrack() function. Track is done with a hash and it is impossible to have a cross 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 file size is above the limit **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

`sourcoise_untrack()` `sourcoise_lapse()` `sourcoise_status()` `sourcoise_refresh()`

Other sourcoise: `sourcoise_clear()`, `sourcoise_clear_all()`, `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(what2keep = "all", root = NULL)
```

Arguments

what2keep	(–) a string (such as "last", the default or "nothing" clears all or "all" removes only non sourcoise files) or a tibble such as the one obtained by `sourcoise_status()`, possibly filtered for the files you whish to keep
root	to force root, not recommended (expert use only)

Value

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

See Also

Other sourcoise: `sourcoise()`, `sourcoise_clear_all()`, `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_clear_all` *Cleans sourcoise cache*

Description

removes every json and qs2 files found by `sourcoise_status()`.

Usage

```
sourcoise_clear_all(root = NULL)
```

Arguments

root	to force root, not recommended (expert use only)

Value

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

See Also

Other sourcoise: `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
data <- sourcoise("some_data.R", force_exec = TRUE)
# we then clear all caches
sourcoise_clear_all()
sourcoise_status()
```

---

sourcoise_lapse *Change Cache Lapse Policy*

Description

Updates the lapse policy metadata for cached files associated with a given path. The lapse policy determines when cached results should expire.

Usage

```
sourcoise_lapse(path, lapse = "never", root = getOption("sourcoise.root"))
```

Arguments

`path`	Character string specifying the file path whose cache metadata should be updated.
`lapse`	Character string specifying the lapse policy. Default is "never". Common values include "never", "daily", "weekly", or custom time periods.
`root`	Character string specifying the root directory for the cache. Defaults to `getOption("sourcoise.root")`.

Details

The function locates all cache entries for the specified path, filters to the most recent entry for each argument hash, and updates the lapse policy metadata only for entries where the current lapse value differs from the specified value.

Value

Invisibly returns the results of writing metadata for each updated cache entry. Returns a message string if no files are found or no changes are needed.

Examples

```
## Not run:
# Set cache to expire daily
sourcoise_lapse("scripts/analysis.R", lapse = "daily")

# Set cache to never expire (default)
sourcoise_lapse("scripts/model.R", lapse = "never")

## End(Not run)
```

`sourcoise_meta` *Get Sourcoise Metadata for a Script*

Description

Retrieves metadata about a cached script without fetching the actual data. This function provides quick access to information about script execution, cache status, and related files.

Usage

```
sourcoise_meta(path, args = NULL, root = NULL, quiet = FALSE)
```

Arguments

`path`	Path to the script file (character). Can be an absolute or relative path.
`args`	Named list of arguments that were passed to the script, if any. Default is `NULL`. This is used to identify the specific cached version when the script was executed with different argument sets.
`root`	(defaut `NULL`) the root of the project (you'd better rely on sourcoise for that one)
`quiet`	(defaut `FALSE`) should we say something ?

Value

A named list containing cache metadata with the following elements:

ok Cache status indicator: "cache ok&valid", "invalid cache", or "no cache data"

timing Execution time of the full script (duration)

date Date and time of the last full execution

size Size of objects returned, measured in R memory

args Arguments given to sourcoise for the script

lapse Delay interval before reexecution is triggered

track List of files being tracked for changes

qmd_file List of Quarto (.qmd) files calling this script

log_file Path to the last log file

file_size Size of cached data on disk

data_date Date of last data save (note: if no new data is generated during execution, no new data file is saved)

data_file Path to the cached data file (stored as .qs2 format)

json_file Path to the JSON file storing metadata (located in .sourcoise directory)

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)

# Access metadata without loading the cached data
meta <- sourcoise_meta("some_data.R")
print(meta$timing)  # View execution time
print(meta$ok)      # Check cache status
```

---

sourcoise_priority *Change Priority of Cached Files*

Description

Updates the priority metadata for cached files associated with a given path. Only affects cache entries where the priority differs from the specified value.

Usage

```
sourcoise_priority(path, priority = 10, root = getOption("sourcoise.root"))
```

Arguments

`path`	Character string specifying the file path whose cache metadata should be updated.
`priority`	Numeric priority value to set. Default is 10. Lower values indicate higher priority.
`root`	Character string specifying the root directory for the cache. Defaults to `getOption("sourcoise.root")`.

Details

The function locates all cache entries for the specified path, filters to the most recent entry for each argument hash, and updates the priority metadata only for entries where the current priority differs from the specified value.

Value

Invisibly returns the results of writing metadata for each updated cache entry. Returns a message string if no files are found or no changes are needed.

Examples

```
## Not run:
# Set priority to 5 for cached results of a script
sourcoise_priority("scripts/analysis.R", priority = 5)

# Use default priority of 10
sourcoise_priority("scripts/model.R")

## End(Not run)
```

---

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"),
  priotirize = TRUE,
  log = "INFO",
  .progress = TRUE
)
```

Arguments

`what`	(tibble) a tibble as generated by `sourcoise_status()`, possibly filtered, (defaut to `source_status()`). What can also be a vector of strings to filter srouces files by name.

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
priotirize	(boolean) (defaut TRUE) will set priority based on pattern of execution
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 can be set automatically using priotirize option. After execution of one refresh, by setting higher priority to more used files.

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_clear_all(), 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_clear_all()`, `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(short = TRUE, quiet = TRUE, root = NULL, prune = TRUE)
```

Arguments

`short`	(boolean) (deafault `TRUE`) return a simplified tibble
`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.

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`: localisation 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_clear_all()`, `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()
```

---

sourcoise_untrack *Remove Tracking from Cached Files*

Description

Removes tracking metadata from cached files associated with a given path by setting the track field to an empty list.

Usage

```
sourcoise_untrack(path, root = getOption("sourcoise.root"))
```

Arguments

path	Character string specifying the file path whose cache metadata should be updated.
root	Character string specifying the root directory for the cache. Defaults to `getOption("sourcoise.root")`.

Details

NOte that tracked fies are accumulated when specified in `track` argument of `soucoise()`. This function allows to reset the list.

The function locates all cache entries for the specified path, filters to entries that currently have tracking enabled (non-empty priority field), and removes tracking by setting the track field to an empty list. Only affects the most recent cache entry for each argument hash.

Value

Invisibly returns the results of writing metadata for each updated cache entry. Returns a message string if no files are found or no tracked files exist.

Examples

```
## Not run:
# Remove tracking from cached results
sourcoise_untrack("scripts/analysis.R")

## End(Not run)
```

Index