Package: filecacher (via r-universe)

September 15, 2024

Type Package

Title File Cacher

Version 0.2.9

Description The main functions in this package are with_cache() and cached_read(). The former is a simple way to cache an R object into a file on disk, using 'cachem'. The latter is a wrapper around any standard read function, but caches both the output and the file list info. If the input file list info hasn't changed, the cache is used; otherwise, the original files are re-read. This can save time if the original operation requires reading from many files, and/or involves lots of processing.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports cachem, glue, here, purrr, rlang, utils, vctrs

RoxygenNote 7.2.3

Suggests arrow, data.table, readr, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

URL https://github.com/orgadish/filecacher

BugReports https://github.com/orgadish/filecacher/issues

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

RemoteUrl https://github.com/orgadish/filecacher

RemoteRef HEAD

RemoteSha b7d02f955d3460e64ed8618b0822b305bd7c5926

Contents

`cached_read` *Read files via cache of file list and contents*

Description

Reads data and save to a local file for easier management and re-reading.

By default, also saves the file info to determine whether the cache is valid, or whether the contents need to be updated because the files have been modified. To skip this, or force reading from scratch, set `skip_file_info=TRUE` or `force=TRUE`, respectively.

If updating is called for, all the files are re-read.

`cached_read_csv()` is a convenience function using a csv read function based on `read_type`.

Usage

```
cached_read(
  files,
  label,
  read_fn,
  cache = NULL,
  type = NULL,
  force = FALSE,
  skip_file_info = FALSE
)

cached_read_csv(
  files,
  label,
  read_type = NULL,
  cache = NULL,
  type = NULL,
  skip_file_info = FALSE,
  force = FALSE
)
```

Arguments

files	A file or files to read with read_fn.
label	A string to use as the name of the file to cache.
read_fn	A function which takes file(s) as its first parameter and reads them. To use a single-input read function such as read.csv() with multiple files, use vectorize_reader(), e.g. read_fn = vectorize_reader(read.csv).
cache	One of the following: • The path to an existing directory to use for caching. If NULL (default) uses the current path, using here::here(). • An existing cache object as generated by file_cache().
type	A string describing the type of cache. Must be NULL or one of 'rds', 'parquet', or 'csv'. If NULL (default), uses 'rds'.
force	If TRUE, forces evaluation even if the cache exists.
skip_file_info	Whether to skip saving and/or checking the file info. Use this when just querying the file system (without opening files) is slow.
read_type	Type of csv read function to use. One of: • "readr": readr::read_csv() • "arrow": vectorize_reader(arrow::read_csv_arrow)() • "data.table": vectorize_reader(data.table::fread)() • "base": vectorize_reader(utils::read.csv)() • NULL (default): uses the first installed.

Value

The result of read_fn(files).

See Also

vectorize_reader() to convert a single-input read function into a multiple-input function.

Examples

```
# Create a temporary directory for the cache.
tf <- tempfile()
dir.create(tf)

# A function that logs when it's called.
read_csv_log <- function(files) {
  message("Reading from file ...")
  return(vectorize_reader(read.csv)(files, stringsAsFactors = TRUE))
}

# `iris` data frame separated into multiple subset files.
iris_files <- system.file("extdata", package = "filecacher") |>
  list.files(pattern = "_only[.]csv$", full.names = TRUE)
```

```
# 1) First time, the message is shown.
iris_files |>
  cached_read("mtcars", read_csv_log, cache = tf) |>
  all.equal(iris)

# 2) Second time, no message is shown since the data is pulled from cache.
iris_files |>
  cached_read("mtcars", read_csv_log, cache = tf) |>
  all.equal(iris)

# 3) If desired, reloading can be forced using `force = TRUE`.
iris_files |>
  cached_read("mtcars", read_csv_log, cache = tf, force = TRUE) |>
  all.equal(iris)


unlink(tf, recursive = TRUE)
```

`dfs_equal` *Compare two data frames (ignoring row order) and ensure they are equal.*

Description

Similar to `dplyr::all_equal(x, y, ignore_row_order=TRUE)`, which is now deprecated.

If either argument is not a data.frame it returns `FALSE`, rather than raise an error.

Usage

```
dfs_equal(target, current)
```

Arguments

`target`	R object.
`current`	other R object, to be compared with `target`.

`file_cache` *Gets or creates a* `cachem` *object for use with other functions.*

Description

Gets or creates a `cachem` object for use with other functions.

Usage

```
file_cache(cache = NULL, type = NULL, ext_prefix = "cache_")
```

Arguments

cache	The path to an existing directory to use for caching. If NULL (default) uses a folder called "cache" in the current path, using here::here(). The folder is created if it does not already exist. **Advanced:** if an existing cachem object is provided, all other parameters are ignored and the object is passed on as is. This functionality is primarily used internally or for testing.
type	A string describing the type of cache. Must be NULL or one of 'rds', 'parquet', or 'csv'. If NULL (default), uses 'rds'.
ext_prefix	The prefix to use with the file extension, e.g. "cache_csv", instead of "csv".

Value

A cachem::cache_disk() object.

See Also

cachem::cache_disk()

Examples

```
# Create a temporary directory for the cache.
tf <- tempfile()
dir.create(tf)

# A dummy function that logs when it's called.
get_df <- function() {
  message("Getting df ...")
  return(mtcars)
}

# Use the resulting object in `with_cache()`.
# 1) The first time, the message is printed.
# 2) The second time, the object is pulled from the cache, with no message.
all.equal(with_cache(get_df(), "df", cache = tf), mtcars)
all.equal(with_cache(get_df(), "df", cache = tf), mtcars)

# `with_cache` is designed to be compatible with piping.
get_df() |>
  with_cache("df", cache = tf) |>
  all.equal(mtcars)


# Advanced: If desired, the `cachem` object methods can be used directly.
cache <- file_cache(tf)
cache$get("df") |> # Get objects previously cached using `with_cache`.
  all.equal(mtcars)
cache$set("df2", mtcars) # Set objects using `$set`.
cache$get("df2") |>
  all.equal(mtcars)
```

```
unlink(tf, recursive = TRUE)
```

`fns_equal` *Check whether two function objects have the same text definition.*

Description

Check whether two function objects have the same text definition.

Usage

```
fns_equal(x, y)
```

Arguments

`x`	First function to compare.
`y`	Second function to compare.

Value

Logical

`get_csv_fns` *Get the CSV read/write function*

Description

Read functions are vectorized.

Usage

```
get_csv_fns(type = NULL)
```

Arguments

`type`	Type of csv read/write functions to get. If `NULL`, returns the first installed.

Value

List of read/write functions.

get_csv_read_fn *Get the first CSV Read function installed*

Description

Get the first CSV Read function installed

Usage

```
get_csv_read_fn(read_type = NULL)
```

Arguments

read_type	Type of csv read function to use. One of: • "readr": `readr::read_csv()` • "arrow": `vectorize_reader(arrow::read_csv_arrow)()` • "data.table": `vectorize_reader(data.table::fread)()` • "base": `vectorize_reader(utils::read.csv)()` • `NULL` (default): uses the first installed.

Value

Function that reads multiple paths to CSVs.

get_file_info *Extract File Information to Indicate if Contents Are Modified.*

Description

Uses `file.info()` to get `size` and `mtime`.

Usage

```
get_file_info(path)
```

Arguments

path	A character vector of one or more paths.

interpret_cache_type *Generate cache parameters from preexisting shorthand types.*

Description

Generate cache parameters from preexisting shorthand types.

Usage

```
interpret_cache_type(type, ext_prefix = "cache_")
```

Arguments

type	A string describing the type of cache. Must be NULL or one of 'rds', 'parquet', or 'csv'. If NULL (default), uses 'rds'.
ext_prefix	The prefix to use with the file extension, e.g. "cache_csv", instead of "csv".

Value

List of read_fn, write_fn, and extension for use with cachem::cache_disk().

vectorize_reader *Vectorize a single-input read function to read multiple files*

Description

The resulting vectorized read function still takes all the arguments of the original function.

Uses purrr::list_rbind() to bind the data frames, which generates a data frame with a superset of the columns from all the files, filling NA where data was not present.

Usage

```
vectorize_reader(read_fn, file_path_to = NULL)
```

Arguments

read_fn	The read function to vectorize. The first argument must be the files to read.
file_path_to	A string, which if provided, is the name of the column containing the file paths in the result. See 'names_to' in purrr::list_rbind().

Value

A version of read_fn that can read multiple paths.

See Also

purrr::list_rbind()

Examples

```
# Convert iris$Species to character to simplify comparison.
iris_chr <- iris
iris_chr$Species <- as.character(iris$Species)


# `iris` data frame separated into multiple subset files.
iris_files <- system.file("extdata", package = "filecacher") |>
  list.files(pattern = "_only[.]csv$", full.names = TRUE)

try(read.csv(iris_files))
vectorize_reader(read.csv)(
  iris_files,
  stringsAsFactors = TRUE
) |>
  all.equal(iris)


if (rlang::is_installed("arrow")) {
  try(arrow::read_csv_arrow(iris_files))
  vectorize_reader(arrow::read_csv_arrow)(
    iris_files
  ) |>
    as.data.frame() |>
    all.equal(iris_chr)
}


if (rlang::is_installed("data.table")) {
  try(data.table::fread(iris_files))
  vectorize_reader(data.table::fread)(
    iris_files,
    stringsAsFactors = TRUE
  ) |>
    as.data.frame() |>
    all.equal(iris)
}
```

`with_cache` *Cache via a file*

Description

If the cache exists, the object is retrieved from the cache. Otherwise, it is evaluated and stored for subsequent retrieval.

Use `force=TRUE` to ensure the object is evaluated and stored anew in the cache.

The object evaluated must be compatible with the cache type. For example, a cache type of 'csv' or 'parquet' requires a `data.frame` or similar type.

Usage

```
with_cache(x, label, cache = NULL, type = NULL, force = FALSE)
```

Arguments

`x`	The object to store in the cache. Must be compatible with the cache type.
`label`	A string to use as the name of the file to cache.
`cache`	One of the following: • The path to an existing directory to use for caching. If `NULL` (default) uses the current path, using `here::here()`. • An existing cache object as generated by `file_cache()`.
`type`	A string describing the type of cache. Must be `NULL` or one of 'rds', 'parquet', or 'csv'. If `NULL` (default), uses 'rds'.
`force`	If `TRUE`, forces evaluation even if the cache exists.

Value

The value of `x`.

Examples

```
# Create a temporary directory for the cache.
tf <- tempfile()
dir.create(tf)

# A dummy function that logs when it's called.
get_df <- function() {
  message("Getting df ...")
  return(mtcars)
}

# Use the resulting object in `with_cache()`.
# 1) The first time, the message is printed.
# 2) The second time, the object is pulled from the cache, with no message.
all.equal(with_cache(get_df(), "df", cache = tf), mtcars)
all.equal(with_cache(get_df(), "df", cache = tf), mtcars)

# `with_cache` is designed to be compatible with piping.
get_df() |>
  with_cache("df", cache = tf) |>
  all.equal(mtcars)


# Advanced: If desired, the `cachem` object methods can be used directly.
```

```
cache <- file_cache(tf)
cache$get("df") |> # Get objects previously cached using `with_cache`.
  all.equal(mtcars)
cache$set("df2", mtcars) # Set objects using `$set`.
cache$get("df2") |>
  all.equal(mtcars)

unlink(tf, recursive = TRUE)
```

Index