Package 'currr'

July 22, 2025

Title Apply Mapping Functions in Frequent Saving

Version 0.1.2

Description Implementations of the family of map() functions with frequent saving of the intermediate results. The contained functions let you start the evaluation of the iterations where you stopped (reading the already evaluated ones from cache), and work with the currently evaluated iterations while remaining ones are running in a background job. Parallel computing is also easier with the workers parameter.

License MIT + file LICENSE

URL https://github.com/MarcellGranat/currr

BugReports https://github.com/MarcellGranat/currr/issues

Depends R (>= 4.1.0)

Imports dplyr, tidyr, readr, stringr, broom, pacman, tibble, clisymbols, job, rstudioapi, scales, parallel, purrr, crayon, stats

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Marcell Granat [aut, cre] (ORCID: <https://orcid.org/0000-0002-4036-1500>)

Maintainer Marcell Granat <granat.marcell@uni-neumann.hu>

Repository CRAN

Date/Publication 2023-02-17 12:20:20 UTC

Contents

`cp_map` *Wrapper function of* `purrr::map`. *Apply a function to each element of a vector, but save the intermediate data after a given number of iterations.*

Description

The map functions transform their input by applying a function to each element of a list or atomic vector and returning an object of the same length as the input. `cp_map` functions work exactly the same way, but creates a secret folder in your current working directory and saves the results if they reach a given checkpoint. This way if you rerun the code, it reads the result from the cache folder and start to evalutate where you finished.

- `cp_map()` always returns a list.
- `map_lgl()`, `map_dbl()` and `map_chr()` return an atomic vector of the indicated type (or die trying). For these functions, `.f` must return a length-1 vector of the appropriate type.

Usage

```
cp_map(.x, .f, ..., name = NULL, cp_options = list())
```

Arguments

`.x`	A list or atomic vector.
`.f`	A function, specified in one of the following ways: • A named function, e.g. `mean`. • An anonymous function, e.g. `\(x) x + 1` or `function(x) x + 1`. • A formula, e.g. `~ .x + 1`. You must use `.x` to refer to the first argument. Only recommended if you require backward compatibility with older versions of R.
`...`	Additional arguments passed on to the mapped function.
`name`	Name for the subfolder in the cache folder. If you do not specify, then `cp_map` uses the name of the function combined with the name of x. This is dangerous, since this generated name can appear multiple times in your code. Also changing x will result a rerun of the code, however you max want to avoid this. (if a subset of .x matches with the cached one and the function is the same, then elements of this subset won't evaluated, rather read from the cache)
`cp_options`	Options for the evaluation: `wait`, `n_checkpoint`, `workers`, `fill`.

- `wait`: An integer to specify that after how many iterations the console shows the intermediate results (default 1). If its value is between 0 and 1, then it is taken as proportions of iterations to wait (example length of .x equals 100, then you get back the result after 50 if you set it to 0.5). Set to `Inf` to get back the results only after full evaluations. If its value is not equal to `Inf` then evaluation is goind in background job.
- `n_chekpoint`: Number of checkpoints, when intermadiate results are saved (default = 100).
- `workers`: Number of CPU cores to use (parallel package called in background). Set to 1 (default) to avoid parallel computing.
- `fill()` When you get back a not fully evaluated result (default `TRUE`). Should the length of the result be the same as .x?

You can set these options also with `options(currr.n_checkpoint = 200)`. Additional options: `currr.unchanged_message` (TRUE/FALSE), `currr.progress_length`

Value

A list.

See Also

Other map variants: `cp_map_chr()`, `cp_map_dbl()`, `cp_map_dfc()`, `cp_map_dfr()`, `cp_map_lgl()`

Examples

```
# Run them on console!
# (functions need writing and reading access to your working directory and they also print)

avg_n <- function(.data, .col, x) {
  Sys.sleep(.01)

  .data |>
    dplyr::pull({{ .col }}) |>
    (\(m) mean(m) * x) ()
}


cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = 2, name = "iris_mean")

 # same function, read from cache
cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = 2, name = "iris_mean")

remove_currr_cache()
```

cp_map_chr *Wrapper function of* purrr::map. *Apply a function to each element of a vector, but save the intermediate data after a given number of iterations.*

Description

The map functions transform their input by applying a function to each element of a list or atomic vector and returning an object of the same length as the input. cp_map functions work exactly the same way, but creates a secret folder in your current working directory and saves the results if they reach a given checkpoint. This way if you rerun the code, it reads the result from the cache folder and start to evalutate where you finished.

- cp_map() always returns a list.
- map_lgl(), map_dbl() and map_chr() return an atomic vector of the indicated type (or die trying). For these functions, .f must return a length-1 vector of the appropriate type.

Usage

```
cp_map_chr(.x, .f, ..., name = NULL, cp_options = list())
```

Arguments

.x	A list or atomic vector.
.f	A function, specified in one of the following ways: • A named function, e.g. mean. • An anonymous function, e.g. \(x) x + 1 or function(x) x + 1. • A formula, e.g. ~ .x + 1. You must use .x to refer to the first argument. Only recommended if you require backward compatibility with older versions of R.
...	Additional arguments passed on to the mapped function.
name	Name for the subfolder in the cache folder. If you do not specify, then cp_map uses the name of the function combined with the name of x. This is dangerous, since this generated name can appear multiple times in your code. Also changing x will result a rerun of the code, however you max want to avoid this. (if a subset of .x matches with the cached one and the function is the same, then elements of this subset won't evaluated, rather read from the cache)
cp_options	Options for the evaluation: wait, n_checkpoint, workers, fill. • wait: An integer to specify that after how many iterations the console shows the intermediate results (default 1). If its value is between 0 and 1, then it is taken as proportions of iterations to wait (example length of .x equals 100, then you get back the result after 50 if you set it to 0.5). Set to Inf to get back the results only after full evaluations. If its value is not equal to Inf then evaluation is goind in background job.

- `n_chekpoint`: Number of checkpoints, when intermadiate results are saved (default = 100).
- `workers`: Number of CPU cores to use (parallel package called in background). Set to 1 (default) to avoid parallel computing.
- `fill()` When you get back a not fully evaluated result (default `TRUE`). Should the length of the result be the same as .x?

You can set these options also with `options(currr.n_checkpoint = 200)`. Additional options: `currr.unchanged_message` (TRUE/FALSE), `currr.progress_length`

Value

A character vector.

See Also

Other map variants: `cp_map_dbl()`, `cp_map_dfc()`, `cp_map_dfr()`, `cp_map_lgl()`, `cp_map()`

Examples

```
# Run them on console!
# (functions need writing and reading access to your working directory and they also print)

avg_n <- function(.data, .col, x) {
  Sys.sleep(.01)

  .data |>
    dplyr::pull({{ .col }}) |>
    (\(m) mean(m) * x) ()
}


cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")

 # same function, read from cache
cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")

remove_currr_cache()
```

`cp_map_dbl` *Wrapper function of* `purrr::map`. *Apply a function to each element of a vector, but save the intermediate data after a given number of iterations.*

Description

The map functions transform their input by applying a function to each element of a list or atomic vector and returning an object of the same length as the input. cp_map functions work exactly the same way, but creates a secret folder in your current working directory and saves the results if they reach a given checkpoint. This way if you rerun the code, it reads the result from the cache folder and start to evalutate where you finished.

- cp_map() always returns a list.
- map_lgl(), map_dbl() and map_chr() return an atomic vector of the indicated type (or die trying). For these functions, .f must return a length-1 vector of the appropriate type.

Usage

```
cp_map_dbl(.x, .f, ..., name = NULL, cp_options = list())
```

Arguments

.x	A list or atomic vector.
.f	A function, specified in one of the following ways: • A named function, e.g. mean. • An anonymous function, e.g. \(x) x + 1 or function(x) x + 1. • A formula, e.g. ~ .x + 1. You must use .x to refer to the first argument. Only recommended if you require backward compatibility with older versions of R.
...	Additional arguments passed on to the mapped function.
name	Name for the subfolder in the cache folder. If you do not specify, then cp_map uses the name of the function combined with the name of x. This is dangerous, since this generated name can appear multiple times in your code. Also changing x will result a rerun of the code, however you max want to avoid this. (if a subset of .x matches with the cached one and the function is the same, then elements of this subset won't evaluated, rather read from the cache)
cp_options	Options for the evaluation: wait, n_checkpoint, workers, fill. • wait: An integer to specify that after how many iterations the console shows the intermediate results (default 1). If its value is between 0 and 1, then it is taken as proportions of iterations to wait (example length of .x equals 100, then you get back the result after 50 if you set it to 0.5). Set to Inf to get back the results only after full evaluations. If its value is not equal to Inf then evaluation is goind in background job. • n_chekpoint: Number of checkpoints, when intermadiate results are saved (default = 100). • workers: Number of CPU cores to use (parallel package called in background). Set to 1 (default) to avoid parallel computing. • fill() When you get back a not fully evaluated result (default TRUE). Should the length of the result be the same as .x? You can set these options also with options(currr.n_checkpoint = 200). Additional options: currr.unchanged_message (TRUE/FALSE), currr.progress_length

Value

A numeric vector.

See Also

Other map variants: cp_map_chr(), cp_map_dfc(), cp_map_dfr(), cp_map_lgl(), cp_map()

Examples

```
# Run them on console!
# (functions need writing and reading access to your working directory and they also print)

avg_n <- function(.data, .col, x) {
  Sys.sleep(.01)

  .data |>
    dplyr::pull({{ .col }}) |>
    (\(m) mean(m) * x) ()
}


cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")

 # same function, read from cache
cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")

remove_currr_cache()
```

---

cp_map_dfc *Wrapper function of* purrr::map. *Apply a function to each element of a vector, but save the intermediate data after a given number of iterations.*

Description

The map functions transform their input by applying a function to each element of a list or atomic vector and returning an object of the same length as the input. cp_map functions work exactly the same way, but creates a secret folder in your current working directory and saves the results if they reach a given checkpoint. This way if you rerun the code, it reads the result from the cache folder and start to evalutate where you finished.

- cp_map() always returns a list.
- map_lgl(), map_dbl() and map_chr() return an atomic vector of the indicated type (or die trying). For these functions, .f must return a length-1 vector of the appropriate type.

Usage

```
cp_map_dfc(.x, .f, ..., name = NULL, cp_options = list())
```

Arguments

`.x`	A list or atomic vector.
`.f`	A function, specified in one of the following ways: • A named function, e.g. `mean`. • An anonymous function, e.g. `\(x) x + 1` or `function(x) x + 1`. • A formula, e.g. `~ .x + 1`. You must use `.x` to refer to the first argument. Only recommended if you require backward compatibility with older versions of R.
`...`	Additional arguments passed on to the mapped function.
`name`	Name for the subfolder in the cache folder. If you do not specify, then `cp_map` uses the name of the function combined with the name of x. This is dangerous, since this generated name can appear multiple times in your code. Also changing x will result a rerun of the code, however you max want to avoid this. (if a subset of .x matches with the cached one and the function is the same, then elements of this subset won't evaluated, rather read from the cache)
`cp_options`	Options for the evaluation: `wait`, `n_checkpoint`, `workers`, `fill`. • `wait`: An integer to specify that after how many iterations the console shows the intermediate results (default `1`). If its value is between 0 and 1, then it is taken as proportions of iterations to wait (example length of .x equals 100, then you get back the result after 50 if you set it to 0.5). Set to `Inf` to get back the results only after full evaluations. If its value is not equal to `Inf` then evaluation is goind in background job. • `n_chekpoint`: Number of checkpoints, when intermadiate results are saved (default = 100). • `workers`: Number of CPU cores to use (parallel package called in background). Set to 1 (default) to avoid parallel computing. • `fill()` When you get back a not fully evaluated result (default `TRUE`). Should the length of the result be the same as .x? You can set these options also with `options(currr.n_checkpoint = 200)`. Additional options: `currr.unchanged_message` (TRUE/FALSE), `currr.progress_length`

Value

A tibble.

See Also

Other map variants: `cp_map_chr()`, `cp_map_dbl()`, `cp_map_dfr()`, `cp_map_lgl()`, `cp_map()`

Examples

```
# Run them on console!
# (functions need writing and reading access to your working directory and they also print)

avg_n <- function(.data, .col, x) {
  Sys.sleep(.01)
```

```
  .data |>
    dplyr::pull({{ .col }}) |>
    (\(m) mean(m) * x) ()
}


cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")

 # same function, read from cache
cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")

remove_currr_cache()
```

`cp_map_dfr` *Wrapper function of* `purrr::map`. *Apply a function to each element of a vector, but save the intermediate data after a given number of iterations.*

Description

The map functions transform their input by applying a function to each element of a list or atomic vector and returning an object of the same length as the input. `cp_map` functions work exactly the same way, but creates a secret folder in your current working directory and saves the results if they reach a given checkpoint. This way if you rerun the code, it reads the result from the cache folder and start to evalutate where you finished.

- `cp_map()` always returns a list.
- `map_lgl()`, `map_dbl()` and `map_chr()` return an atomic vector of the indicated type (or die trying). For these functions, `.f` must return a length-1 vector of the appropriate type.

Usage

```
cp_map_dfr(.x, .f, ..., name = NULL, cp_options = list())
```

Arguments

`.x`	A list or atomic vector.
`.f`	A function, specified in one of the following ways: • A named function, e.g. `mean`. • An anonymous function, e.g. `\(x) x + 1` or `function(x) x + 1`. • A formula, e.g. `~ .x + 1`. You must use `.x` to refer to the first argument. Only recommended if you require backward compatibility with older versions of R.
`...`	Additional arguments passed on to the mapped function.

name — Name for the subfolder in the cache folder. If you do not specify, then cp_map uses the name of the function combined with the name of x. This is dangerous, since this generated name can appear multiple times in your code. Also changing x will result a rerun of the code, however you max want to avoid this. (if a subset of .x matches with the cached one and the function is the same, then elements of this subset won't evaluated, rather read from the cache)

cp_options — Options for the evaluation: wait, n_checkpoint, workers, fill.

- wait: An integer to specify that after how many iterations the console shows the intermediate results (default 1). If its value is between 0 and 1, then it is taken as proportions of iterations to wait (example length of .x equals 100, then you get back the result after 50 if you set it to 0.5). Set to Inf to get back the results only after full evaluations. If its value is not equal to Inf then evaluation is goind in background job.
- n_chekpoint: Number of checkpoints, when intermadiate results are saved (default = 100).
- workers: Number of CPU cores to use (parallel package called in background). Set to 1 (default) to avoid parallel computing.
- fill() When you get back a not fully evaluated result (default TRUE). Should the length of the result be the same as .x?

You can set these options also with options(currr.n_checkpoint = 200). Additional options: currr.unchanged_message (TRUE/FALSE), currr.progress_length

Value

A tibble.

See Also

Other map variants: cp_map_chr(), cp_map_dbl(), cp_map_dfc(), cp_map_lgl(), cp_map()

Examples

```
# Run them on console!
# (functions need writing and reading access to your working directory and they also print)

avg_n <- function(.data, .col, x) {
  Sys.sleep(.01)

  .data |>
    dplyr::pull({{ .col }}) |>
    (\(m) mean(m) * x) ()
}


cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")

 # same function, read from cache
cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")
```

```
remove_currr_cache()
```

`cp_map_lgl` *Wrapper function of* `purrr::map`. *Apply a function to each element of a vector, but save the intermediate data after a given number of iterations.*

Description

The map functions transform their input by applying a function to each element of a list or atomic vector and returning an object of the same length as the input. `cp_map` functions work exactly the same way, but creates a secret folder in your current working directory and saves the results if they reach a given checkpoint. This way if you rerun the code, it reads the result from the cache folder and start to evalutate where you finished.

- `cp_map()` always returns a list.
- `map_lgl()`, `map_dbl()` and `map_chr()` return an atomic vector of the indicated type (or die trying). For these functions, `.f` must return a length-1 vector of the appropriate type.

Usage

```
cp_map_lgl(.x, .f, ..., name = NULL, cp_options = list())
```

Arguments

`.x`	A list or atomic vector.
`.f`	A function, specified in one of the following ways: • A named function, e.g. `mean`. • An anonymous function, e.g. `\(x) x + 1` or `function(x) x + 1`. • A formula, e.g. `~ .x + 1`. You must use `.x` to refer to the first argument. Only recommended if you require backward compatibility with older versions of R.
`...`	Additional arguments passed on to the mapped function.
`name`	Name for the subfolder in the cache folder. If you do not specify, then `cp_map` uses the name of the function combined with the name of x. This is dangerous, since this generated name can appear multiple times in your code. Also changing x will result a rerun of the code, however you max want to avoid this. (if a subset of .x matches with the cached one and the function is the same, then elements of this subset won't evaluated, rather read from the cache)
`cp_options`	Options for the evaluation: `wait`, `n_checkpoint`, `workers`, `fill`. • `wait`: An integer to specify that after how many iterations the console shows the intermediate results (default 1). If its value is between 0 and 1, then it is taken as proportions of iterations to wait (example length of .x equals 100, then you get back the result after 50 if you set it to 0.5). Set to `Inf` to get back the results only after full evaluations. If its value is not equal to `Inf` then evaluation is goind in background job.

- `n_chekpoint`: Number of checkpoints, when intermadiate results are saved (default = 100).
- `workers`: Number of CPU cores to use (parallel package called in background). Set to 1 (default) to avoid parallel computing.
- `fill()` When you get back a not fully evaluated result (default `TRUE`). Should the length of the result be the same as .x?

You can set these options also with `options(currr.n_checkpoint = 200)`. Additional options: `currr.unchanged_message` (TRUE/FALSE), `currr.progress_length`

Value

A logical vector.

See Also

Other map variants: `cp_map_chr()`, `cp_map_dbl()`, `cp_map_dfc()`, `cp_map_dfr()`, `cp_map()`

Examples

```
# Run them on console!
# (functions need writing and reading access to your working directory and they also print)

avg_n <- function(.data, .col, x) {
  Sys.sleep(.01)

  .data |>
    dplyr::pull({{ .col }}) |>
    (\(m) mean(m) * x) ()
}


cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")

 # same function, read from cache
cp_map(.x = 1:10, .f = avg_n, .data = iris, .col = Sepal.Length, name = "iris_mean")

remove_currr_cache()
```

---

remove_currr_cache *Remove currr's intermediate data from the folder.*

Description

Remove currr's intermediate data from the folder.

Usage

```
remove_currr_cache(list = NULL)
```

Arguments

list	A character vector specifying the name of the caches you want to remove (files in .currr.data folder). If empy (default), all caches will be removed.

Value

No return value, called for side effects

---

saving_map *Run a map with the function, but saves after a given number of execution. This is an internal function, you are not supposed to use it manually, but can call for background job inly if exported.*

Description

Run a map with the function, but saves after a given number of execution. This is an internal function, you are not supposed to use it manually, but can call for background job inly if exported.

Usage

```
saving_map(.ids, .f, name, n_checkpoint = 100, currr_folder, ...)
```

Arguments

.ids	Placement of .x to work with.
.f	Called function.
name	Name for saving.
n_checkpoint	Number of checkpoints.
currr_folder	Folder where cache files are stored.
...	Additionals.

Value

No return value, called for side effects

saving_map_nodot	*Run a map with the function, but saves after a given number of execution. This is an internal function, you are not supposed to use it manually, but can call for background job only if exported. This function differs from saving_map, since it does not have a ... input. This is neccessary because job::job fails if ... is not provided for the cp_map call.*

Description

Run a map with the function, but saves after a given number of execution. This is an internal function, you are not supposed to use it manually, but can call for background job only if exported. This function differs from saving_map, since it does not have a ... input. This is neccessary because job::job fails if ... is not provided for the cp_map call.

Usage

```
saving_map_nodot(.ids, .f, name, n_checkpoint = 100, currr_folder)
```

Arguments

.ids	Placement of .x to work with.
.f	Called function.
name	Name for saving.
n_checkpoint	Number of checkpoints.
currr_folder	Folder where cache files are stored.

Value

No return value, called for side effects

Index