# Package 'depcache'

October 13, 2022

**Type** Package

**Title** Cache R Expressions, Taking Their Dependencies into Account

**Imports** codetools, methods

**Version** 0.1-2

**Description** Hash an expression with its dependencies and store its value, reloading it from a file as long as both the expression and its dependencies stay the same.

**License** GPL (>= 3)

**NeedsCompilation** yes

**Author** Ivan Krylov [aut, cre]

**Maintainer** Ivan Krylov <krylov.r00t@gmail.com>

**Repository** CRAN

**Date/Publication** 2022-06-21 08:50:03 UTC

## R topics documented:



---

depcache-package *Cache R Expressions, Taking Their Dependencies into Account*

---

### Description

Hash an expression with its dependencies and store its value, reloading it from a file as long as both the expression and its dependencies stay the same.

## Details

The functions in this package take an expression, walk its code to find its dependencies and calculate a hash of them. If a corresponding file already exists, it is loaded; otherwise, the expression is evaluated and its value is saved in the file. Optionally, this check may be performed every time a variable is accessed.

By default, a subdirectory of the current directory is used to store the cache files.

Index of help topics:

```
cache                   Evaluate an expression and cache its results
depcache-package        Cache R Expressions, Taking Their Dependencies
                        into Account
depcache.options        Caching options
setCached               Cache-tracking assignment
```

## Author(s)

Ivan Krylov

## References

FNV-1a hash: http://www.isthe.com/chongo/tech/comp/fnv/

## See Also

cache, %<-%

## Examples

```
a <- 1
# will evaluate expression
cache({ message('evaluating expression'); a + 1 }) # 2
# will reuse cached value
x %<-% { message('evaluating expression'); a + 1 } # 2
x
a <- 2
# will recalculate the value
x # 3
```

---

cache *Evaluate an expression and cache its results*

---

## Description

This function extracts all dependencies of an R expression, hashes them together with the expression itself and either loads the already-existing file, or evaluates the expression and stores the result in that file.

## Usage

```
cache(expr, extra = NULL, ...)
```

## Arguments

| | |
|---|---|
| `expr` | An expression to evaluate or load from cache, unquoted. |
| `extra` | Any R value that should be considered part of the state deciding whether the expression should be re-computed. For example, if `expr` reads a file, consider using `file.mtime` or `md5sum` to check for changes in it. |
| `...` | Additional options, see `depcache.options`. |

## Details

Currently, the hash is obtained by means of serialisation. In order to make semantically same values have same hashes on a wide range of R versions, the following steps were taken:

- When computing the hash of the serialized data (only the XDR format version 2 or 3 is supported), the first 14 bytes containing the header (including the version of R that serialized the data) are ignored.
- Every function is "rebuilt" from its body before hashing, forcing R to discard the bytecode and the source references from the copy of the function before it's hashed.
- Strings are converted to UTF-8 before hashing.
- All this is done recursively.

The exact algorithm used and the way hash is obtained are implementation details and may eventually change, though not without a good reason.

Other aspects of R data structures are currently not handled:

- Nothing is done about environments. Due to them being reference objects, any fix-up must re-create them from scratch, taking potentially recursive dependencies into account, which is likely expensive.
- Some S4 classes (like reference class implementations) just have different representations in different versions of R and third-party packages. They may mean the same thing, but they serialize to different byte sequences.

## Value

The result of evaluating `expr`, either directly, or loaded from cache.

## See Also

`setCached`

## Examples

```
a <- 1
# will evaluate the expression the first time
cache({ message('evaluating expression'); a + 1 }) # 2
# saved value of the expression will be used
cache({
  message('evaluating expression')
  # even if written a bit differently
  a + 1
}) # 2
a <- -1
# expression evaluated again because dependencies changed
cache({ message('evaluating expression'); a + 1 }) # 0
```

---

`depcache.options` *Caching options*

---

## Description

Control how the dependencies are gathered and hashed to locate the determine the file name to load the cached object from.

## Usage

```
depcache.options(
  defaults = getOption("depcache.version", '0.1'),
  skip = getOption("depcache.skip", NULL),
  dir, compress, local.only, format.version
)
```

## Arguments

| | |
|---|---|
| `defaults` | A string containing the version of **depcache** to get other defaults from. If not set, takes the value from the `depcache.version` option (see `options`), falling back to the current version of the package.<br>To make the caching more reproducible against package updates, call `options(depcache.version = `*something*`)` once at the top of your scripts.<br>Currently, only version `0.1` is accepted. When a new version of the package changes the defaults or adds new settings, the range of the accepted values will expand. |
| `skip` | A character vector of variables to omit from automatically-gathered dependencies. Variables carrying unintended or unimportant state, which would otherwise interfere with obtaining a reproducible hash, should be listed here. This may be useful when a symbol encountered in the expression doesn't signify a variable in |

the evaluation frame (e.g. non-standard evaluation when plotting with **lattice**), or when the variable is being assigned to as part of the expression.

Defaults to the `depcache.skip` option, or `NULL` if unset.

`dir` The directory to store the cache files inside. Defaults to the `depcache.dir` option, or '`.depcache`' in **depcache** version 0.1.

`compress` Passed as the `compress` option to `saveRDS` when saving the cached objects. Defaults to the `depcache.compress` option, or `TRUE` in **depcache** version 0.1.

`local.only` If `TRUE`, only variables available in the same environment where the caching function has been called from are considered as dependencies; parent environments are ignored. Typically, this means taking local variables as parts of the hash that determines the file name, but not loaded packages or `attach`ed datasets. Setting this to `FALSE` may invalidate the cache next time a package or R itself is updated.

Defaults to the `depcache.local.only` option, or `TRUE` in **depcache** version 0.1.

`format.version` Passed as the `version` argument to `saveRDS` and also used when serialising any objects to hash them. Only versions `2` and `3` are supported. Defaults to the `depcache.format.version` option, or `2` in **depcache** version 0.1.

## Details

In all cases, explicitly passed arguments override settings from the `options()`, which override the defaults. Depending on the `defaults` argument or the `depcache.version` option, the defaults may change; setting it explicitly will help your scripts stay forward-compatible.

This function shouldn't be normally called by the user (except, perhaps, to verify the parameters about to be passed to the caching functions), but it is automatically invoked on every call to `cache`, `setCached`, or the use of cache-tracking assignment operators `%<-%` and `%->%`. Any additional options passed to the functions as `...` are handled here, and so are the global `options`.

## Value

A list containing the settings to be used by the caching system.

`dir` The directory used for storage of the cache files.

`compress` Passed to `saveRDS`.

`skip` Variables to skip when hashing the dependencies of the expressions.

`local.only` Whether to ignore non-local dependencies.

`format.version` Passed to `saveRDS` as the `version` argument. Also determines the format version when serialising the variables to hash them.

## See Also

`cache`, `setCached`

## Examples

```
# The output is affected by the user's use of options(...) and the
# current version of the package
options(depcache.local.only = FALSE)
print(depcache.options(format.version = 3))
options(depcache.local.only = TRUE)
print(depcache.options())

# "skip" makes it possible to avoid mistaking arguments evaluated in a
# non-standard way for local variables
speed <- 1
options(depcache.skip = 'speed')
x %<-% { message('fitting the model'); lm(dist ~ speed, cars) }
speed <- 0
# not fitted again despite change in local variable "speed"
summary(x)
```

---

# setCached *Cache-tracking assignment*

---

## Description

Cache expression values and automatically recalculate them when their dependencies change

## Usage

```
symbol %<-% expr
expr %->% symbol
setCached(symbol, expr, extra = NULL, ...)
```

## Arguments

| | |
|---|---|
| symbol | A variable name to associate with the expression, unquoted. |
| expr | The expression to cache, taking dependencies into account. |
| extra | An unquoted expression to be considered an extra part of the state, in addition to the automatically determined dependencies. Will be evaluated every time the variable is accessed to determine whether it should be recalculated. |
| ... | Additional settings, see depcache.options. |

## Details

Sets up the variable *symbol* to automatically recalculate the value of expr any time its dependencies change, using makeActiveBinding and the same mechanisms that power cache.

Initially, expr is loaded from cache or evaluated, and the hash is remembered. When the variable named by *symbol* is accessed, its dependencies are hashed together with expr (this may be done

recursively if the dependencies are themselves active bindings set up the same way). If the hash changes, the value of `expr` is again loaded from `cache` (if available) or evaluated anew.

To prevent infinite loops during dependency calculation, *symbol* is automatically skipped, but a self-dependent `expr` is probably a bad idea anyway.

## Value

Returns the value of `expr`, invisibly. Called for the side effect of creating an active binding with a name specified by `symbol`.

## See Also

`cache`, `makeActiveBinding`

## Examples

```
a <- 1
# will evaluate the expression first
x %<-% { message('evaluating expression "x"'); a + 1 }
x # 2
# will reuse cached value
{
  message('evaluating expression "y"')
  a + 1
  # even if written a bit differently
} %->% y
y # 2
a <- -1
# will evaluate the expression again
x # 0
# will load the new cached value
y # 0
setCached(z, x + y)
a <- 0
z # recalculates all three
```

# Index