Package 'reproducible'

August 11, 2017

Type Package

Title A Set of Tools that Enhance Reproducibility Beyond Package Management

Description Built on top of 'git2r' and 'archivist', this package aims at making high-level, robust, machine and OS independent tools for making deeply reproducible and reusable content in R. This extends beyond the package management utilities of 'packrat' and 'checkpoint' by including tools for caching, and accessing GitHub repositories.

URL https://github.com/PredictiveEcology/reproducible

Date 2017-08-10

Version 0.1.3

Depends R (>= 3.3.2)

Imports archivist (>= 2.1.2), data.table (>= 1.10.4), devtools, digest, fastdigest, git2r (>= 0.18), magrittr, methods, raster, sp

Suggests hunspell, knitr, rgdal, rmarkdown, testthat

License GPL-3

BugReports https://github.com/PredictiveEcology/reproducible/issues

ByteCompile yes

RoxygenNote 6.0.1

Collate 'cache-helpers.R' 'cache-tools.R' 'robustDigest.R' 'cache.R' 'consistentPaths.R' 'git.R' 'reproducible-package.R'

NeedsCompilation no

Author Eliot J B McIntire [aut, cre], Alex M Chubaty [aut], Her Majesty the Queen in Right of Canada, as represented by the Minister of Natural Resources Canada [cph]

Maintainer Eliot J B McIntire <eliot.mcintire@canada.ca>

Repository CRAN

Date/Publication 2017-08-10 22:08:23 UTC

R topics documented:

`reproducible-package` *The* `reproducible` *package*

Description

Built on top of `git2r` and `archivist`, this package aims at making high-level, robust, machine and OS independent tools for making deeply reproducible and reusable content in R. This extends beyond the package management utilities of `packrat` and `checkpoint` by including tools for caching, and accessing GitHub repositories.

Author(s)

Maintainer: Eliot J B McIntire <eliot.mcintire@canada.ca>

Authors:

- Alex M Chubaty <alexander.chubaty@canada.ca>

Other contributors:

- Her Majesty the Queen in Right of Canada, as represented by the Minister of Natural Resources Canada [copyright holder]

See Also

Useful links:

- https://github.com/PredictiveEcology/reproducible
- Report bugs at https://github.com/PredictiveEcology/reproducible/issues

---

`.addTagsToOutput` *Add tags to object*

Description

This is a generic definition that can be extended according to class. This function and methods should do "deep" copy for archiving purposes.

Usage

```
.addTagsToOutput(object, outputObjects, FUN, preDigestByClass)

## S4 method for signature 'ANY'
.addTagsToOutput(object, outputObjects, FUN, preDigestByClass)
```

Arguments

`object`	Any R object.
`outputObjects`	Optional character vector indicating which objects to return. This is only relevant for `simList` objects
`FUN`	A function
`preDigestByClass`	A list, usually from `.preDigestByClass`

Value

New object with tags attached.

Author(s)

Eliot McIntire

.cacheMessage *Create a custom cache message by class*

Description

This is a generic definition that can be extended according to class.

Usage

```
.cacheMessage(object, functionName)

## S4 method for signature 'ANY'
.cacheMessage(object, functionName)
```

Arguments

object	Any R object.
functionName	A character string indicating the function name

Value

Nothing; called for its messaging side effect.

Author(s)

Eliot McIntire

.checkCacheRepo *Check for cache repository info in ...*

Description

This is a generic definition that can be extended according to class. Normally, checkPath can be called directly, but does not have class-specific methods.

Usage

```
.checkCacheRepo(object, create = FALSE)

## S4 method for signature 'ANY'
.checkCacheRepo(object, create = FALSE)
```

Arguments

object	An R object
create	Logical. If TRUE, then it will create the path for cache.

Value

A character string with a path to a cache repository.

Author(s)

Eliot McIntire

.debugCache *Attach debug info to return for Cache*

Description

Internal use only. Attaches an attribute to the output, usable for debugging the Cache.

Usage

```
.debugCache(obj, preDigest, ...)
```

Arguments

obj	An arbitrary R object.
preDigest	A list of hashes.
...	Dots passed from Cache

Value

The same object as obj, but with 2 attributes set.

Author(s)

Eliot McIntire

.objSizeInclEnviros *Determine object size of all objects inside environments*

Description

This is a generic definition that can be extended according to class.

Usage

```
.objSizeInclEnviros(object)

## S4 method for signature 'ANY'
.objSizeInclEnviros(object)

## S4 method for signature 'environment'
.objSizeInclEnviros(object)
```

Arguments

object	Any R object.

Value

A numeric, the result of object.size for all objects in environments.

Author(s)

Eliot McIntire

.preDigestByClass *Any miscellaneous things to do before* .robustDigest *and after* FUN *call*

Description

The default method for preDigestByClass and simply returns NULL. There may be methods in other packages.

Usage

```
.preDigestByClass(object)

## S4 method for signature 'ANY'
.preDigestByClass(object)
```

Arguments

object	Any R object.

Value

A list with elements that will likely be used in .postProcessing

Author(s)

Eliot McIntire

`.prepareFileBackedRaster` *Copy the file-backing of a file-backed Raster* object*

Description

Rasters are sometimes file-based, so the normal save and copy and assign mechanisms in R don't work for saving, copying and assigning. This function creates an explicit file copy of the file that is backing the raster, and changes the pointer (i.e., `filename(object)`) so that it is pointing to the new file.

Usage

```
.prepareFileBackedRaster(obj, repoDir = NULL,
  compareRasterFileLength = 1e+06, ...)
```

Arguments

`obj`	The raster object to save to the repository.
`repoDir`	Character denoting an existing directory in which an artifact will be saved.
`compareRasterFileLength`	Numeric. Optional. When there are Rasters, that have file-backed storage, this is passed to the length arg in `digest` when determining if the Raster file is already in the database. Note: uses `digest` for file-backed Raster. Default 1e6. Passed to `.prepareFileBackedRaster`.
`...`	passed to `archivist::saveToRepo`

Value

A raster object and its newly located file backing. Note that if this is a legitimate archivist repository, the new location will be a subdirectory called '`rasters/`' of '`repoDir/`'. If this is not a repository, the new location will be within `repoDir`.

Author(s)

Eliot McIntire

.prepareOutput *Make any modifications to object recovered from cacheRepo*

Description

This is a generic definition that can be extended according to class.

Usage

```
.prepareOutput(object, cacheRepo, ...)

## S4 method for signature 'RasterLayer'
.prepareOutput(object, cacheRepo, ...)

## S4 method for signature 'ANY'
.prepareOutput(object, cacheRepo, ...)
```

Arguments

object	Any R object
cacheRepo	A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
...	Arguments of FUN function .

Value

The object, modified

Author(s)

Eliot McIntire

.sortDotsUnderscoreFirst *Sort a any named object with dotted names first*

Description

Internal use only. This exists so Windows and Linux machines can have the same order after a sort.

Usage

```
.sortDotsUnderscoreFirst(obj)
```

Arguments

obj	An arbitrary R object for which a names function returns a character vector.

Value

The same object as obj, but sorted with .objects first.

Author(s)

Eliot McIntire

Examples

```
items <- c(A = "a", Z = "z", `.D` = ".d", `_W` = "_w")
.sortDotsUnderscoreFirst(items)
```

.tagsByClass *Add extra tags to an archive based on class*

Description

This is a generic definition that can be extended according to class.

Usage

```
.tagsByClass(object)

## S4 method for signature 'ANY'
.tagsByClass(object)
```

Arguments

object	Any R object.

Value

A character vector of new tags.

Author(s)

Eliot McIntire

Cache *Cache method that accommodates environments, S4 methods, Rasters*

Description

Cache method that accommodates environments, S4 methods, Rasters

Usage

```
Cache(FUN, ..., notOlderThan = NULL, objects = NULL, outputObjects = NULL,
  algo = "xxhash64", cacheRepo = NULL, compareRasterFileLength = 1e+06,
  userTags = c(), digestPathContent = FALSE, omitArgs = NULL,
  classOptions = list(), debugCache = character(), sideEffect = FALSE,
  makeCopy = FALSE, quick = FALSE)

## S4 method for signature 'ANY'
Cache(FUN, ..., notOlderThan = NULL, objects = NULL,
  outputObjects = NULL, algo = "xxhash64", cacheRepo = NULL,
  compareRasterFileLength = 1e+06, userTags = c(),
  digestPathContent = FALSE, omitArgs = NULL, classOptions = list(),
  debugCache = character(), sideEffect = FALSE, makeCopy = FALSE,
  quick = FALSE)
```

Arguments

FUN	A function to be called.
...	Arguments of FUN function .
notOlderThan	load an artifact from the database only if it was created after notOlderThan.
objects	Character vector of objects to be digested. This is only applicable if there is a list, environment or simList with named objects within it. Only this/these objects will be considered for caching, i.e., only use a subset of the list, environment or simList objects.
outputObjects	Optional character vector indicating which objects to return. This is only relevant for simList objects
algo	The algorithms to be used; currently available choices are md5, which is also the default, sha1, crc32, sha256, sha512, xxhash32, xxhash64 and murmur32.
cacheRepo	A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
compareRasterFileLength	Numeric. Optional. When there are Rasters, that have file-backed storage, this is passed to the length arg in digest when determining if the Raster file is already in the database. Note: uses digest for file-backed Raster. Default 1e6. Passed to .prepareFileBackedRaster.
userTags	A character vector with Tags. These Tags will be added to the repository along with the artifact.

digestPathContent
: Logical. Should arguments that are of class Path (see examples below) have their name digested (FALSE; default), or their file contents (TRUE).

omitArgs
: Optional character string of arguments in the FUN to omit from the digest.

classOptions
: Optional list. This will pass into .robustDigest for specific classes. Should be options that the .robustDigest knows what to do with.

debugCache
: Character or Logical. Either "complete" or "quick" (uses partial matching, so "c" or "q" work). TRUE is equivalent to "complete". If "complete", then the returned object from the Cache function will have two attributes, debugCache1 and debugCache2, which are the entire list(...) and that same object, but after all .robustDigest calls, at the moment that it is digested using fastdigest, respectively. This attr(mySimOut, "debugCache2") can then be compared to a subsequent call and individual items within the object attr(mySimOut, "debugCache1") can be compared. If "quick", then it will return the same two objects directly, without evalutating the FUN(...).

sideEffect
: Logical. Check if files to be downloaded are found locally in the cacheRepo prior to download and try to recover from a copy (makeCopy must have been set to TRUE the first time Cache was run). Default is FALSE. *NOTE: this argument is experimental and may change in future releases.*

makeCopy
: Logical. If sideEffect = TRUE, and makeCopy = TRUE, a copy of the downloaded files will be made and stored in the cacheRepo to speed up subsequent file recovery in the case where the original copy of the downloaded files are corrupted or missing. Currently only works when set to TRUE during the first run of Cache. Default is FALSE. *NOTE: this argument is experimental and may change in future releases.*

quick
: Logical. If sideEffect = TRUE, setting this to TRUE, will hash the file's metadata (i.e., filename and file size) instead of hashing the contents of the file(s). If set to FALSE (default), the contents of the file(s) are hashed. *NOTE: this argument is experimental and may change in future releases.*

Details

Caching R objects using cache has four important limitations:

1. the archivist package detects different environments as different;
2. it also does not detect S4 methods correctly due to method inheritance;
3. it does not detect objects that have file-base storage of information (specifically RasterLayer-class objects);
4. the default hashing algorithm is relatively slow.

This version of the Cache function accommodates those four special, though quite common, cases by:

1. converting any environments into list equivalents;
2. identifying the dispatched S4 method (including those made through inheritance) before hashing so the correct method is being cached;

3. by hashing the linked file, rather than the Raster object. Currently, only file-backed `Raster*` objects are digested (e.g., not `ff` objects, or any other R object where the data are on disk instead of in RAM);
4. using `fastdigest` internally when the object is in RAM, which can be up to ten times faster than `digest`. Note that file-backed objects are still hashed using `digest`.

If `Cache` is called within a SpaDES module, then the cached entry will automatically get 3 extra `userTags`: `eventTime`, `eventType`, and `moduleName`. These can then be used in `clearCache` to selectively remove cached objects by `eventTime`, `eventType` or `moduleName`.

`Cache` will add a tag to the artifact in the database called `accessed`, which will assign the time that it was accessed, either read or write. That way, artifacts can be shown (using `showCache`) or removed (using `clearCache`) selectively, based on their access dates, rather than only by their creation dates. See example in `clearCache`. `Cache` (uppercase C) is used here so that it is not confused with, and does not mask, the `archivist::cache` function.

Value

As with `cache`, returns the value of the function call or the cached version (i.e., the result from a previous call to this same cached function with identical arguments).

Filepaths

If a function has a path argument, there is some ambiguity about what should be done. Possibilities include:

1. hash the string as is (this will be very system specific, meaning a `Cache` call will not work if copied between systems or directories);
2. hash the `basename(path)`;
3. hash the contents of the file.

If paths are passed in as is (i.e,. character string), the result will not be predictable. Instead, one should use the wrapper function `asPath(path)`, which sets the class of the string to a `Path`, and one should decide whether one wants to digest the content of the file (using `digestPathContent = TRUE`), or just the filename (`(digestPathContent = FALSE)`). See examples.

Stochasticity

In general, it is expected that caching will only be used when stochasticity is not relevant, or if a user has achieved sufficient stochasticity (e.g., via sufficient number of calls to `experiment`) such that no new explorations of stochastic outcomes are required. It will also be very useful in a reproducible workflow.

Note

As indicated above, several objects require pre-treatment before caching will work as expected. The function `.robustDigest` accommodates this. It is an S4 generic, meaning that developers can produce their own methods for different classes of objects. Currently, there are methods for several types of classes. See `.robustDigest`.

See `.robustDigest` for other specifics for other classes.

Author(s)

Eliot McIntire

See Also

cache, .robustDigest

Examples

```
library(raster)

tmpDir <- file.path(tempdir(), "reproducible_examples", "Cache")

## Example 1: basic cache use
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")

showCache(tmpDir, userTags = c("objectName"))
showCache(tmpDir, userTags = c("^a$")) # regular expression ... "a" exactly
showCache(tmpDir, userTags = c("runif")) # show only cached objects made during runif call

clearCache(tmpDir, userTags = c("runif")) # remove only cached objects made during runif call
showCache(tmpDir) # only those made during rnorm call

clearCache(tmpDir)

## Example 2: using the "accessed" tag
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")

# access it again, but "later"
Sys.sleep(1)
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
wholeCache <- showCache(tmpDir)

# keep only items accessed "recently" (i.e., only objectName:a)
onlyRecentlyAccessed <- showCache(tmpDir, userTags = max(wholeCache[tagKey == "accessed"]$tagValue))

# inverse join with 2 data.tables ... using: a[!b]
# i.e., return all of wholeCache that was not recently accessed
toRemove <- unique(wholeCache[!onlyRecentlyAccessed], by = "artifact")$artifact
clearCache(tmpDir, toRemove) # remove ones not recently accessed
showCache(tmpDir) # still has more recently accessed

clearCache(tmpDir)

## Example 3: using keepCache
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")

# keep only those cached items from the last 24 hours
oneDay <- 60 * 60 * 24
```

```
keepCache(tmpDir, after = Sys.time() - oneDay)

# Keep all Cache items created with an rnorm() call
keepCache(tmpDir, userTags = "rnorm")

# Remove all Cache items that happened within a rnorm() call
clearCache(tmpDir, userTags = "rnorm")

showCache(tmpDir) ## empty

## Example 4: searching for multiple objects in the cache

# default userTags is "and" matching; for "or" matching use |
ranNumsA <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:b")

# show all objects (runif and rnorm in this case)
showCache(tmpDir)

# show objects that are both runif and rnorm
# (i.e., none in this case, because objecs are either or, not both)
showCache(tmpDir, userTags = c("runif", "rnorm")) ## empty

# show objects that are either runif or rnorm ("or" search)
showCache(tmpDir, userTags = "runif|rnorm")

# keep only objects that are either runif or rnorm ("or" search)
keepCache(tmpDir, userTags = "runif|rnorm")

clearCache(tmpDir)

## Example 5: using caching to speed up rerunning expensive computations
ras <- raster(extent(0, 100, 0, 100), res = 1,
              vals = sample(1:5, replace = TRUE, size = 1e4),
              crs = "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84")

# A slow operation, like GIS operation
notCached <- suppressWarnings(
  # project raster generates warnings when run non-interactively
  projectRaster(ras, crs = crs(ras), res = 5, cacheRepo = tmpDir)
)

cached <- suppressWarnings(
  # project raster generates warnings when run non-interactively
  Cache(projectRaster, ras, crs = crs(ras), res = 5, cacheRepo = tmpDir)
)

# second time is much faster
reRun <- suppressWarnings(
  # project raster generates warnings when run non-interactively
  Cache(projectRaster, ras, crs = crs(ras), res = 5, cacheRepo = tmpDir)
)
```

```
# recovered cached version is same as non-cached version
all.equal(notCached, reRun) ## TRUE

## Example 6: working with file paths

# if passing a character string, it will take 2 complete passes to before
#  a cached copy is used when it is a save event (read or load is different)
obj <- 1:10
fname <- tempfile(fileext = ".RData")
Cache(saveRDS, obj, file = fname, cacheRepo = tmpDir)
Cache(saveRDS, obj, file = fname, cacheRepo = tmpDir)
Cache(saveRDS, obj, file = fname, cacheRepo = tmpDir) # cached copy is loaded

# however, using asPath(), cached version retrieved after being run once
fname1 <- tempfile(fileext = ".RData")
Cache(saveRDS, obj, file = asPath(fname1), cacheRepo = tmpDir)
Cache(saveRDS, obj, file = asPath(fname1), cacheRepo = tmpDir) # cached copy is loaded

clearCache(tmpDir)

## cleanup
unlink(c("filename.rda", "filename1.rda"))
unlink(dirname(tmpDir), recursive = TRUE)
```

---

cache *Deprecated functions*

Description

Deprecated functions

Usage

```
cache(cacheRepo = NULL, FUN, ..., notOlderThan = NULL, objects = NULL,
  outputObjects = NULL, algo = "xxhash64")

## S4 method for signature 'ANY'
cache(cacheRepo = NULL, FUN, ..., notOlderThan = NULL,
  objects = NULL, outputObjects = NULL, algo = "xxhash64")
```

Arguments

cacheRepo	A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
FUN	A function to be called.
...	Arguments of FUN function .
notOlderThan	load an artifact from the database only if it was created after notOlderThan.

objects	Character vector of objects to be digested. This is only applicable if there is a list, environment or simList with named objects within it. Only this/these objects will be considered for caching, i.e., only use a subset of the list, environment or simList objects.
outputObjects	Optional character vector indicating which objects to return. This is only relevant for simList objects
algo	The algorithms to be used; currently available choices are md5, which is also the default, sha1, crc32, sha256, sha512, xxhash32, xxhash64 and murmur32.

---

checkoutVersion *Clone, fetch, and checkout from GitHub.com repositories*

Description

In reproducible research, not only do packages and R version have to be consistent, but also specific versions of version controlled scripts. This function allows a simple way to create an exactly copy locally of a git repository. It can use ssh keys (including GitHub deploy keys) or GitHub Personal Access Tokens.

Usage

```
checkoutVersion(repo, localRepoPath = ".", cred = "", ...)
```

Arguments

repo	Repository address in the format username/repo[/subdir][@ref|#pull]. Alternatively, you can specify subdir and/or ref using the respective parameters (see below); if both are specified, the values in repo take precedence.
localRepoPath	Character string. The path into which the git repo should be cloned, fetched, and checked out from.
cred	Character string. Either the name of the environment variable that contains the GitHub PAT or filename of the GitHub private key file.
...	Additional arguments passed to git2r functions.

Value

Invisibly returns a repository class object, defined in git_repository-class

Author(s)

Eliot McIntire and Alex Chubaty

Examples

```
## Not run:
  tmpDir <- tempfile("")
  dir.create(tmpDir)
  repo <- "PredictiveEcology/reproducible"

  ## get latest from master branch
  localRepo <- checkoutVersion("PredictiveEcology/reproducible",
                               localRepoPath = tmpDir)
  git2r::summary(localRepo)
  unlink(tmpDir, recursive = TRUE)

  ## get latest from development branch
  localRepo <- checkoutVersion(paste0(repo, "@", "development"), localRepoPath = tmpDir)
  git2r::summary(localRepo)
  unlink(tmpDir, recursive = TRUE)

  ## get a particular commit by sha
  sha <- "8179e1910e7c617fdeacad0f9d81323e6aad57c3"
  localRepo <- checkoutVersion(paste0(repo, "@", sha), localRepoPath = tmpDir)
  git2r::summary(localRepo)
  unlink(tmpDir, recursive = TRUE)

  rm(localRepo, repo)

## End(Not run)
```

---

checkPath *Check filepath*

Description

Checks the specified filepath for formatting consistencies, such as trailing slashes, etc.

Usage

```
checkPath(path, create)

## S4 method for signature 'character,logical'
checkPath(path, create)

## S4 method for signature 'character,missing'
checkPath(path)

## S4 method for signature '`NULL`,ANY'
checkPath(path)

## S4 method for signature 'missing,ANY'
checkPath()
```

Arguments

path	A character string corresponding to a filepath.
create	A logical indicating whether the path should be created if it doesn't exist. Default is FALSE.

Value

Character string denoting the cleaned up filepath.

See Also

file.exists, dir.create.

Examples

```
## normalize file paths
paths <- list("./aaa/zzz",
              "./aaa/zzz/",
              ".//aaa//zzz",
              ".//aaa//zzz/",
              ".\\aaa\\zzz",
              ".\\aaa\\zzz\\",
              file.path(".", "aaa", "zzz"))

checked <- normPath(paths)
length(unique(checked)) ## 1; all of the above are equivalent

## check to see if a path exists
tmpdir <- file.path(tempdir(), "example_checkPath")

dir.exists(tmpdir) ## FALSE
tryCatch(checkPath(tmpdir, create = FALSE), error = function(e) FALSE) ## FALSE

checkPath(tmpdir, create = TRUE)
dir.exists(tmpdir) ## TRUE

unlink(tmpdir, recursive = TRUE)
```

---

clearCache *Examining and modifying the cache*

Description

These are convenience wrappers around archivist package functions. They allow the user a bit of control over what is being cached.

Usage

```
clearCache(x, userTags = character(), after, before, ...)

## S4 method for signature 'ANY'
clearCache(x, userTags = character(), after, before, ...)

showCache(x, userTags = character(), after, before, ...)

## S4 method for signature 'ANY'
showCache(x, userTags = character(), after, before, ...)

keepCache(x, userTags = character(), after, before, ...)

## S4 method for signature 'ANY'
keepCache(x, userTags = character(), after, before, ...)
```

Arguments

x	A simList or a directory containing a valid archivist repository
userTags	Character vector. If used, this will be used in place of the after and before. Specifying one or more userTag here will clear all objects that match those tags. Matching is via regular expresssion, meaning partial matches will work unless strict beginning (^) and end ($) of string characters are used. Matching will be against any of the 3 columns returned by showCache(), i.e., artifact, tagValue or tagName. Also, length userTags > 1, then matching is by 'and'. For 'or' matching, use | in a single character string. See examples.
after	A time (POSIX, character understandable by data.table). Objects cached after this time will be shown or deleted.
before	A time (POSIX, character understandable by data.table). Objects cached before this time will be shown or deleted.
...	Other arguments. Currently unused.
	If neither after or before are provided, nor userTags, then all objects will be removed. If both after and before are specified, then all objects between after and before will be deleted. If userTags is used, this will override after or before.

Details

clearCache remove items from the cache based on their userTag or times values.

keepCache remove all cached items *except* those based on certain userTags or times values.

showCache display the contents of the cache.

Value

Will clear all (or that match userTags, or between after or before) objects from the repository located at cachePath of the sim object, if sim is provided, or located in cacheRepo. Also returns a data.table invisibly of the removed items.

See Also

splitTagsLocal.

Examples

```
library(raster)

tmpDir <- file.path(tempdir(), "reproducible_examples", "Cache")

## Example 1: basic cache use
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")

showCache(tmpDir, userTags = c("objectName"))
showCache(tmpDir, userTags = c("^a$")) # regular expression ... "a" exactly
showCache(tmpDir, userTags = c("runif")) # show only cached objects made during runif call

clearCache(tmpDir, userTags = c("runif")) # remove only cached objects made during runif call
showCache(tmpDir) # only those made during rnorm call

clearCache(tmpDir)

## Example 2: using the "accessed" tag
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")

# access it again, but "later"
Sys.sleep(1)
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
wholeCache <- showCache(tmpDir)

# keep only items accessed "recently" (i.e., only objectName:a)
onlyRecentlyAccessed <- showCache(tmpDir, userTags = max(wholeCache[tagKey == "accessed"]$tagValue))

# inverse join with 2 data.tables ... using: a[!b]
# i.e., return all of wholeCache that was not recently accessed
toRemove <- unique(wholeCache[!onlyRecentlyAccessed], by = "artifact")$artifact
clearCache(tmpDir, toRemove) # remove ones not recently accessed
showCache(tmpDir) # still has more recently accessed

clearCache(tmpDir)

## Example 3: using keepCache
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")

# keep only those cached items from the last 24 hours
oneDay <- 60 * 60 * 24
keepCache(tmpDir, after = Sys.time() - oneDay)

# Keep all Cache items created with an rnorm() call
keepCache(tmpDir, userTags = "rnorm")
```

```
# Remove all Cache items that happened within a rnorm() call
clearCache(tmpDir, userTags = "rnorm")

showCache(tmpDir) ## empty

## Example 4: searching for multiple objects in the cache

# default userTags is "and" matching; for "or" matching use |
ranNumsA <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:b")

# show all objects (runif and rnorm in this case)
showCache(tmpDir)

# show objects that are both runif and rnorm
# (i.e., none in this case, because objecs are either or, not both)
showCache(tmpDir, userTags = c("runif", "rnorm")) ## empty

# show objects that are either runif or rnorm ("or" search)
showCache(tmpDir, userTags = "runif|rnorm")

# keep only objects that are either runif or rnorm ("or" search)
keepCache(tmpDir, userTags = "runif|rnorm")

clearCache(tmpDir)

## Example 5: using caching to speed up rerunning expensive computations
ras <- raster(extent(0, 100, 0, 100), res = 1,
              vals = sample(1:5, replace = TRUE, size = 1e4),
              crs = "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84")

# A slow operation, like GIS operation
notCached <- suppressWarnings(
  # project raster generates warnings when run non-interactively
  projectRaster(ras, crs = crs(ras), res = 5, cacheRepo = tmpDir)
)

cached <- suppressWarnings(
  # project raster generates warnings when run non-interactively
  Cache(projectRaster, ras, crs = crs(ras), res = 5, cacheRepo = tmpDir)
)

# second time is much faster
reRun <- suppressWarnings(
  # project raster generates warnings when run non-interactively
  Cache(projectRaster, ras, crs = crs(ras), res = 5, cacheRepo = tmpDir)
)

# recovered cached version is same as non-cached version
all.equal(notCached, reRun) ## TRUE

## Example 6: working with file paths
```

```
# if passing a character string, it will take 2 complete passes to before
#  a cached copy is used when it is a save event (read or load is different)
obj <- 1:10
fname <- tempfile(fileext = ".RData")
Cache(saveRDS, obj, file = fname, cacheRepo = tmpDir)
Cache(saveRDS, obj, file = fname, cacheRepo = tmpDir)
Cache(saveRDS, obj, file = fname, cacheRepo = tmpDir) # cached copy is loaded

# however, using asPath(), cached version retrieved after being run once
fname1 <- tempfile(fileext = ".RData")
Cache(saveRDS, obj, file = asPath(fname1), cacheRepo = tmpDir)
Cache(saveRDS, obj, file = asPath(fname1), cacheRepo = tmpDir) # cached copy is loaded

clearCache(tmpDir)

## cleanup
unlink(c("filename.rda", "filename1.rda"))
unlink(dirname(tmpDir), recursive = TRUE)
library(raster)

tmpDir <- file.path(tempdir(), "reproducible_examples", "Cache")

## Example 1: basic cache use
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")

showCache(tmpDir, userTags = c("objectName"))
showCache(tmpDir, userTags = c("^a$")) # regular expression ... "a" exactly
showCache(tmpDir, userTags = c("runif")) # show only cached objects made during runif call

clearCache(tmpDir, userTags = c("runif")) # remove only cached objects made during runif call
showCache(tmpDir) # only those made during rnorm call

clearCache(tmpDir)

## Example 2: using the "accessed" tag
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")

# access it again, but "later"
Sys.sleep(1)
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
wholeCache <- showCache(tmpDir)

# keep only items accessed "recently" (i.e., only objectName:a)
onlyRecentlyAccessed <- showCache(tmpDir, userTags = max(wholeCache[tagKey == "accessed"]$tagValue))

# inverse join with 2 data.tables ... using: a[!b]
# i.e., return all of wholeCache that was not recently accessed
toRemove <- unique(wholeCache[!onlyRecentlyAccessed], by = "artifact")$artifact
clearCache(tmpDir, toRemove) # remove ones not recently accessed
showCache(tmpDir) # still has more recently accessed
```

```
clearCache(tmpDir)

## Example 3: using keepCache
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")

# keep only those cached items from the last 24 hours
oneDay <- 60 * 60 * 24
keepCache(tmpDir, after = Sys.time() - oneDay)

# Keep all Cache items created with an rnorm() call
keepCache(tmpDir, userTags = "rnorm")

# Remove all Cache items that happened within a rnorm() call
clearCache(tmpDir, userTags = "rnorm")

showCache(tmpDir) ## empty

## Example 4: searching for multiple objects in the cache

# default userTags is "and" matching; for "or" matching use |
ranNumsA <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:b")

# show all objects (runif and rnorm in this case)
showCache(tmpDir)

# show objects that are both runif and rnorm
# (i.e., none in this case, because objecs are either or, not both)
showCache(tmpDir, userTags = c("runif", "rnorm")) ## empty

# show objects that are either runif or rnorm ("or" search)
showCache(tmpDir, userTags = "runif|rnorm")

# keep only objects that are either runif or rnorm ("or" search)
keepCache(tmpDir, userTags = "runif|rnorm")

clearCache(tmpDir)

## Example 5: using caching to speed up rerunning expensive computations
ras <- raster(extent(0, 100, 0, 100), res = 1,
              vals = sample(1:5, replace = TRUE, size = 1e4),
              crs = "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84")

# A slow operation, like GIS operation
notCached <- suppressWarnings(
  # project raster generates warnings when run non-interactively
  projectRaster(ras, crs = crs(ras), res = 5, cacheRepo = tmpDir)
)

cached <- suppressWarnings(
  # project raster generates warnings when run non-interactively
```

```
  Cache(projectRaster, ras, crs = crs(ras), res = 5, cacheRepo = tmpDir)
)

# second time is much faster
reRun <- suppressWarnings(
  # project raster generates warnings when run non-interactively
  Cache(projectRaster, ras, crs = crs(ras), res = 5, cacheRepo = tmpDir)
)

# recovered cached version is same as non-cached version
all.equal(notCached, reRun) ## TRUE

## Example 6: working with file paths

# if passing a character string, it will take 2 complete passes to before
#  a cached copy is used when it is a save event (read or load is different)
obj <- 1:10
fname <- tempfile(fileext = ".RData")
Cache(saveRDS, obj, file = fname, cacheRepo = tmpDir)
Cache(saveRDS, obj, file = fname, cacheRepo = tmpDir)
Cache(saveRDS, obj, file = fname, cacheRepo = tmpDir) # cached copy is loaded

# however, using asPath(), cached version retrieved after being run once
fname1 <- tempfile(fileext = ".RData")
Cache(saveRDS, obj, file = asPath(fname1), cacheRepo = tmpDir)
Cache(saveRDS, obj, file = asPath(fname1), cacheRepo = tmpDir) # cached copy is loaded

clearCache(tmpDir)

## cleanup
unlink(c("filename.rda", "filename1.rda"))
unlink(dirname(tmpDir), recursive = TRUE)
```

`clearStubArtifacts` *Clear erroneous archivist artifacts*

Description

Clear stub artifacts resulting when an archive object is being saved at the same time as another process doing the same thing.

Usage

```
clearStubArtifacts(repoDir = NULL)

## S4 method for signature 'ANY'
clearStubArtifacts(repoDir = NULL)
```

Arguments

repoDir	A character denoting an existing directory of the repository for which metadata will be returned. If NULL (default), it will use the repoDir specified in archivist::setLocalRepo.

Value

Invoked for its side effect on the repoDir.

Author(s)

Eliot McIntire

Examples

```
tmpDir <- file.path(tempdir(), "reproducible_examples", "clearStubArtifacts")

lapply(c(runif, rnorm), function(f) {
  reproducible::Cache(f, 10, cacheRepo = tmpDir)
})

# clear out any stub artifacts
clearStubArtifacts(tmpDir)

# cleanup
clearCache(tmpDir)
unlink(tmpDir, recursive = TRUE)
```

Copy *Recursive copying of nested environments, and other "hard to copy" objects*

Description

When copying environments and all the objects contained within them, there are no copies made: it is a pass-by-reference operation. Sometimes, a deep copy is needed, and sometimes, this must be recursive (i.e., environments inside environments).

Usage

```
Copy(object, filebackedDir = tempdir(), ...)

## S4 method for signature 'ANY'
Copy(object, filebackedDir = tempdir(), ...)

## S4 method for signature 'data.table'
Copy(object, filebackedDir = tempdir(), ...)
```

```
## S4 method for signature 'environment'
Copy(object, filebackedDir = tempdir(), ...)

## S4 method for signature 'list'
Copy(object, filebackedDir = tempdir(), ...)

## S4 method for signature 'data.frame'
Copy(object, filebackedDir = tempdir(), ...)

## S4 method for signature 'Raster'
Copy(object, filebackedDir = tempdir(), ...)
```

Arguments

`object`	An R object (likely containing environments) or an environment.
`filebackedDir`	A directory to copy any files that are backing R objects, currently only valid for `Raster` classes. Defaults to `tempdir()`, which is unlikely to be very useful.
`...`	Only used for custom Methods

Author(s)

Eliot McIntire

See Also

`.robustDigest`

Examples

```
e <- new.env()
e$abc <- letters
e$one <- 1L
e$lst <- list(W = 1:10, X = runif(10), Y = rnorm(10), Z = LETTERS[1:10])
ls(e)

# 'normal' copy
f <- e
ls(f)
f$one
f$one <- 2L
f$one
e$one ## uh oh, e has changed!

# deep copy
e$one <- 1L
g <- Copy(e)
ls(g)
g$one
g$one <- 3L
```

```
g$one
f$one
e$one
```

---

copyFile *Copy a file using* Robocopy *on Windows and* rsync *on Linux/macOS*

Description

This will copy an individual file faster using Robocopy on Windows, and using rsync on macOS and Linux.

Usage

```
copyFile(from = NULL, to = NULL, useRobocopy = TRUE, overwrite = TRUE,
  delDestination = FALSE, create = TRUE, silent = FALSE)
```

Arguments

from	The source file.
to	The new file.
useRobocopy	For Windows, this will use a system call to Robocopy which appears to be much faster than the internal file.copy function. Uses /MIR flag. Default TRUE.
overwrite	Passed to file.copy
delDestination	Logical, whether the destination should have any files deleted, if they don't exist in the source. This is /purge.
create	Passed to checkLazyDir.
silent	Should a progress be printed.

Author(s)

Eliot McIntire and Alex Chubaty

Examples

```
tmpDirFrom <- file.path(tempdir(), "example_fileCopy_from")
tmpDirTo <- file.path(tempdir(), "example_fileCopy_to")
tmpFile <- tempfile("file", tmpDirFrom, ".csv")
dir.create(tmpDirFrom)
f1 <- normalizePath(tmpFile, mustWork = FALSE)
f2 <- normalizePath(file.path(tmpDirTo, basename(tmpFile)), mustWork = FALSE)

write.csv(data.frame(a = 1:10, b = runif(10), c = letters[1:10]), f1)
copyFile(f1, f2)
file.exists(f2) ## TRUE
identical(read.csv(f1), read.csv(f2)) ## TRUE
```

```
unlink(tmpDirFrom, recursive = TRUE)
unlink(tmpDirTo, recursive = TRUE)
```

---

normPath *Normalize filepath*

Description

Checks the specified filepath for formatting consistencies: 1) use slash instead of backslash; 2) do tilde etc. expansion; 3) remove trailing slash.

Usage

```
normPath(path)

## S4 method for signature 'character'
normPath(path)

## S4 method for signature 'list'
normPath(path)

## S4 method for signature '`NULL`'
normPath(path)

## S4 method for signature 'missing'
normPath()
```

Arguments

path	A character vector of filepaths.

Value

Character vector of cleaned up filepaths.

Examples

```
## normalize file paths
paths <- list("./aaa/zzz",
              "./aaa/zzz/",
              ".//aaa//zzz",
              ".//aaa//zzz/",
              ".\\aaa\\zzz",
              ".\\aaa\\zzz\\",
              file.path(".", "aaa", "zzz"))

checked <- normPath(paths)
```

```
length(unique(checked)) ## 1; all of the above are equivalent

## check to see if a path exists
tmpdir <- file.path(tempdir(), "example_checkPath")

dir.exists(tmpdir) ## FALSE
tryCatch(checkPath(tmpdir, create = FALSE), error = function(e) FALSE) ## FALSE

checkPath(tmpdir, create = TRUE)
dir.exists(tmpdir) ## TRUE

unlink(tmpdir, recursive = TRUE)
```

`Path-class` *Coerce a character string to a class "Path"*

Description

Allows a user to specify that their character string is indeed a filepath. Thus, methods that require only a filepath can be dispatched correctly.

Usage

```
asPath(obj)

## S3 method for class 'character'
asPath(obj)
```

Arguments

`obj`	A character string to convert to a `Path`.

Details

It is often difficult or impossible to know algorithmically whether a character string corresponds to a valid filepath. In the case where it is en existing file, `file.exists` can work. But if it does not yet exist, e.g., for a `save`, it is difficult to know whether it is a valid path before attempting to save to the path.

This is primarily useful for achieving repeatability with Caching. Essentially, when Caching, it is likely that character strings should be digested verbatim, i.e., it must be an exact copy for the Cache mechanism to detect a candidate for recovery from the cache. Paths, are different. While they are character strings, there are many ways to write the same path. Examples of identical meaning, but different character strings are: path exanding of ~ vs. not, double back slash vs. single forward slash, relative path vs. absolute path. All of these should be assessed for their actual file or directory location, NOT their character string. By converting all character string that are actual file or directory paths with this function, then `Cache` will correctly assess the location, NOT the character string representation.

Examples

```
tmpf <- tempfile(fileext = ".csv")
file.exists(tmpf)    ## FALSE
tmpfPath <- asPath(tmpf)
is(tmpf, "Path")     ## FALSE
is(tmpfPath, "Path") ## TRUE
```

Index