Package 'reproducible'

September 26, 2021

Type Package

Title A Set of Tools that Enhance Reproducibility Beyond Package Management

Description Collection of high-level, machine- and OS-independent tools for making deeply reproducible and reusable content in R. The two workhorse functions are Cache and prepInputs; these allow for: nested caching, robust to environments, and objects with environments (like functions); and data retrieval and processing in continuous workflow environments. In all cases, efforts are made to make the first and subsequent calls of functions have the same result, but vastly faster at subsequent times by way of checksums and digesting. Several features are still under active development, including cloud storage of cached objects, allowing for sharing between users. Several advanced options are available, see ?reproducibleOptions.

SystemRequirements 'unrar' (Linux/macOS) or '7-Zip' (Windows) to work with '.rar' files.

URL https://reproducible.predictiveecology.org, https://github.com/PredictiveEcology/reproducible

Date 2021-09-26

Version 1.2.8

Depends R (>= 3.6)

Imports data.table (>= 1.10.4), DBI, digest, fpCompare, gdalUtilities, glue, magrittr, methods, Require, raster, RSQLite, rlang, sp (>= 1.4-2), utils

Suggests covr, crayon, fastdigest, fasterize, future, gdalUtils, googledrive, httr, lwgeom, qs, knitr, parallel, RCurl (>= 1.95-4.8), rgdal, rgeos, rmarkdown, sf, stats, terra, testthat

Encoding UTF-8

Language en-CA

License GPL-3

VignetteBuilder knitr, rmarkdown

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

ByteCompile yes

RoxygenNote 7.1.1

Collate 'DBI.R' 'cache-helpers.R' 'cache-internals.R' 'cache-tools.R' 'robustDigest.R' 'cache.R' 'checksums.R' 'cloud.R' 'cloudOld.R' 'convertPaths.R' 'copy.R' 'download.R' 'gis.R' 'git.R' 'helpers.R' 'objectSize.R' 'options.R' 'packages.R' 'pipe.R' 'postProcess.R' 'preProcess.R' 'prepInputs.R' 'reexports.R' 'reproducible-deprecated.R' 'reproducible-package.R' 'search.R' 'spatialObjects-class.R' 'zzz.R'

NeedsCompilation no

Author Eliot J B McIntire [aut, cre] (<https://orcid.org/0000-0002-6914-8316>), Alex M Chubaty [aut] (<https://orcid.org/0000-0001-7146-8135>), Tati Micheletti [ctb] (<https://orcid.org/0000-0003-4838-8342>), Ceres Barros [ctb] (<https://orcid.org/0000-0003-4036-977X>), Ian Eddy [ctb] (<https://orcid.org/0000-0001-7397-2116>), 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 2021-09-26 16:40:04 UTC

R topics documented:

reproducible-package *The* reproducible *package*

Description

This package aims at making high-level, robust, machine and OS independent tools for making deeply reproducible and reusable content in R. The core user functions are `Cache` and `prepInputs`. Each of these is built around many core and edge cases required to have deeply reproducible code.

Main Tools

There are many elements within the reproducible package. However, there are currently three main ones that are critical for reproducible research. The key element for reproducible research is that the code must always return the same content every time it is run, but it must be vastly faster the 2nd, 3rd, 4th etc, time it is run. That way, the entire code sequence for a project of arbitrary size can be run *from the start* every time.

`Cache`**:** A robust wrapper for any function, including those with environments, disk-backed storage (currently on `Raster`) class), operating-system independent, whose first time called will execute the function, second time will compare the inputs to a database of entries, and recover the first result if inputs are identical. If `options("reproducible.useMemoise" = TRUE)`, the third time will be very fast as it will recover the answer from RAM.

`prepInputs`**:** Download, or load objects, and possibly post-process them. The main advantage to using this over more direct routes is that it will automatically build checksums tables, use `Cache` internally where helpful, and possibly run a variety of post-processing actions. This means this function can also itself be cached for even more speed. This allows all project data to be stored in custom cloud locations or in their original online data repositories, `without altering` code between the first, second, third, etc., times the code is run.

Package options

See `reproducibleOptions` for a complete description of package `options` to configure behaviour.

Author(s)

Maintainer: Eliot J B McIntire <achubaty@for-cast.ca> (ORCID)

Authors:

- Alex M Chubaty <achubaty@for-cast.ca> (ORCID)

Other contributors:

- Tati Micheletti <tati.micheletti@gmail.com> (ORCID) [contributor]
- Ceres Barros <cbarros@mail.ubc.ca> (ORCID) [contributor]
- Ian Eddy <ian.eddy@canada.com> (ORCID) [contributor]
- Her Majesty the Queen in Right of Canada, as represented by the Minister of Natural Resources Canada [copyright holder]

See Also

Useful links:

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

---

`.addChangedAttr` *Add an attribute to an object indicating which named elements change*

Description

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

Usage

```
.addChangedAttr(object, preDigest, origArguments, ...)

## S4 method for signature 'ANY'
.addChangedAttr(object, preDigest, origArguments, ...)
```

Arguments

`object`	Any R object returned from a function
`preDigest`	The full, element by element hash of the input arguments to that same function, e.g., from `.robustDigest`
`origArguments`	These are the actual arguments (i.e., the values, not the names) that were the source for `preDigest`
`...`	Anything passed to methods.

Value

The object, modified

Author(s)

Eliot McIntire

Examples

```
a <- 1
.addChangedAttr(a) # does nothing because default method is just a pass through
```

.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 list, environment (or similar) 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,
  fromMemoise = getOption("reproducible.useMemoise", TRUE),
  verbose = getOption("reproducible.verbose", 1)
)

## S4 method for signature 'ANY'
.cacheMessage(
  object,
  functionName,
  fromMemoise = getOption("reproducible.useMemoise", TRUE),
  verbose = getOption("reproducible.verbose", 1)
)
```

Arguments

`object`	Any R object.
`functionName`	A character string indicating the function name
`fromMemoise`	Logical. If `TRUE`, the message will be about recovery from memoised copy
`verbose`	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., `options('reproducible.verbose' = 0)` to reduce to `minimal`

Value

Nothing; called for its messaging side effect.

Author(s)

Eliot McIntire

Examples

```
a <- 1
.cacheMessage(a, "mean")
```

`.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,
  verbose = getOption("reproducible.verbose", 1)
)

## S4 method for signature 'ANY'
.checkCacheRepo(
  object,
  create = FALSE,
  verbose = getOption("reproducible.verbose", 1)
)
```

Arguments

`object`	An R object
`create`	Logical. If TRUE, then it will create the path for cache.
`verbose`	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., `options('reproducible.verbose' = 0)` to reduce to `minimal`

Value

A character string with a path to a cache repository.

Author(s)

Eliot McIntire

Examples

```
a <- "test"
.checkCacheRepo(a) # no cache repository supplied
```

.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

.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

Examples

```
a <- 1
.preDigestByClass(a) # returns NULL in the simple case here.
```

`.prefix` *Add a prefix or suffix to the basename part of a file path*

Description

Prepend (or postpend) a filename with a prefix (or suffix). If the directory name of the file cannot be ascertained from its path, it is assumed to be in the current working directory.

Usage

```
.prefix(f, prefix = "")

.suffix(f, suffix = "")
```

Arguments

`f`	A character string giving the name/path of a file.
`prefix`	A character string to prepend to the filename.
`suffix`	A character string to postpend to the filename.

Author(s)

Jean Marchal and Alex Chubaty

Examples

```
# file's full path is specified (i.e., dirname is known)
myFile <- file.path("~/data", "file.tif")
.prefix(myFile, "small_")    ## "/home/username/data/small_file.tif"
.suffix(myFile, "_cropped") ## "/home/username/data/myFile_cropped.shp"

# file's full path is not specified
.prefix("myFile.shp", "small")    ## "./small_myFile.shp"
.suffix("myFile.shp", "_cropped") ## "./myFile_cropped.shp"
```

`.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,
  overwrite = FALSE,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)
```

Arguments

`obj`	The raster object to save to the repository.
`repoDir`	Character denoting an existing directory in which an artifact will be saved.
`overwrite`	Logical. Should the raster be saved to disk, overwriting existing file.
`drv`	an object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).
`conn`	A DBIConnection object, as returned by `dbConnect()`.
`...`	Not used

Value

A raster object and its newly located file backing. Note that if this is a legitimate Cache 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

Examples

```
library(raster)
# make a cache repository
a <- Cache(rnorm, 1)

r <- raster(extent(0,10,0,10), vals = 1:100)

# write to disk manually -- will be in tempdir()
r <- writeRaster(r, file = tempfile())

# copy it to the cache repository
r <- .prepareFileBackedRaster(r, tempdir())

r # now in "rasters" subfolder of tempdir()
```

.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 'Raster'
.prepareOutput(
  object,
  cacheRepo,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)

## 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 passed to FUN
drv	an object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).
conn	A DBIConnection object, as returned by dbConnect().

Value

The object, modified

Author(s)

Eliot McIntire

Examples

```
a <- 1
.prepareOutput(a) # does nothing

b <- "NULL"
.prepareOutput(b) # converts to NULL

library(raster)
r <- raster(extent(0,10,0,10), vals = 1:100)

# write to disk manually -- will be in tempdir()
r <- writeRaster(r, file = tempfile())

# copy it to the cache repository
r <- .prepareOutput(r, tempdir())
```

.removeCacheAtts *Remove attributes that are highly varying*

Description

Remove attributes that are highly varying

Usage

```
.removeCacheAtts(x, passByReference = TRUE)
```

Arguments

x	Any arbitrary R object that could have attributes
passByReference	Logical. If TRUE, the default, this uses data.table::setattr to remove several attributes that are unnecessary for digesting, specifically tags, .Cache and call

.requireNamespace *Provide standard messaging for missing package dependencies*

Description

This provides a standard message format for missing packages, e.g., detected via `requireNamespace`.

Usage

```
.requireNamespace(
  pkg = "methods",
  minVersion = NULL,
  stopOnFALSE = FALSE,
 messageStart = paste0(pkg, if (!is.null(minVersion)) paste0("(>=", minVersion, ")"),
    " is required. Try: ")
)
```

Arguments

`pkg`	Character string indicating name of package required
`minVersion`	Character string indicating minimum version of package that is needed
`stopOnFALSE`	Logical. If `TRUE`, this function will create an error (i.e., `stop`) if the function returns `FALSE`; otherwise it simply returns `FALSE`
`messageStart`	A character string with a prefix of message to provide

.setSubAttrInList *Set subattributes within a list by reference*

Description

This uses `data.table::setattr`, but in the case where there is only a single element within a list attribute.

Usage

```
.setSubAttrInList(object, attr, subAttr, value)
```

Arguments

`object`	An arbitrary object
`attr`	The attribute name (that is a list object) to change
`subAttr`	The list element name to change
`value`	The new value

`.sortDotsUnderscoreFirst` *Sort or order any named object with dotted names and underscores first*

Description

Internal use only. This exists so Windows, Linux, and Mac machines can have the same order after a sort. It will put dots and underscores first (with the sort key based on their second character, see examples. It also sorts lower case before upper case.

Usage

```
.sortDotsUnderscoreFirst(obj)

.orderDotsUnderscoreFirst(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", `_C` = "_C")
.sortDotsUnderscoreFirst(items)

# dots & underscore (using 2nd character), then all lower then all upper
items <- c(B = "Upper", b = "lower", A = "a", `.D` = ".d", `_C` = "_C")
.sortDotsUnderscoreFirst(items)

# with a vector
.sortDotsUnderscoreFirst(c(".C", "_B", "A")) # _B is first
```

.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

Examples

```
.tagsByClass(character()) # Nothing interesting. Other packages will make methods
```

assessDataType *Assess the appropriate raster layer data type*

Description

Can be used to write prepared inputs on disk.

This is a convenience function around `assessDataType(ras,type = "GDAL")`

Usage

```
assessDataType(ras, type = "writeRaster")

## S3 method for class 'Raster'
assessDataType(ras, type = "writeRaster")

## S3 method for class 'RasterStack'
assessDataType(ras, type = "writeRaster")

## Default S3 method:
assessDataType(ras, type = "writeRaster")

assessDataTypeGDAL(ras)
```

Arguments

ras	The RasterLayer or RasterStack for which data type will be assessed.
type	Character. "writeRaster" (default) or "GDAL" to return the recommended data type for writing from the raster and gdalUtils packages, respectively, or "projectRaster" to return recommended resampling type.

Value

The appropriate data type for the range of values in ras. See dataType for details.

The appropriate data type for the range of values in ras for using GDAL. See dataType for details.

Author(s)

Eliot McIntire

Ceres Barros

Ian Eddy

Eliot McIntire

Eliot McIntire, Ceres Barros, Ian Eddy, and Tati Micheletti

Examples

```
## LOG1S
library(raster)
ras <- raster(ncol = 10, nrow = 10)
ras[] <- rep(c(0,1),50)
assessDataType(ras)

ras[] <- rep(c(TRUE,FALSE),50)
assessDataType(ras)

ras[] <- c(NA, NA, rep(c(0,1),49))
assessDataType(ras)
```

```
ras <- raster(ncol = 10, nrow = 10)
ras[] <- c(0, NaN, rep(c(0,1),49))
assessDataType(ras)


## INT1S
ras[] <- -1:98
assessDataType(ras)

ras[] <- c(NA, -1:97)
assessDataType(ras)

## INT1U
ras <- raster(ncol = 10, nrow = 10)
ras[] <- 1:100
assessDataType(ras)

ras[] <- c(NA, 2:100)
assessDataType(ras)

## INT2U
ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = 64000, max = 65000))
assessDataType(ras)

## INT2S
ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = -32767, max = 32767))
assessDataType(ras)

ras[54] <- NA
assessDataType(ras)

## INT4U
ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = 0, max = 500000000))
assessDataType(ras)

ras[14] <- NA
assessDataType(ras)

## INT4S
ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = -200000000, max = 200000000))
assessDataType(ras)

ras[14] <- NA
assessDataType(ras)

## FLT4S
ras <- raster(ncol = 10, nrow = 10)
ras[] <- runif(100, min = -10, max = 87)
assessDataType(ras)
```

```
ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = -3.4e+26, max = 3.4e+28))
assessDataType(ras)

ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = 3.4e+26, max = 3.4e+28))
assessDataType(ras)

ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = -3.4e+26, max = -1))
assessDataType(ras)

## FLT8S
ras <- raster(ncol = 10, nrow = 10)
ras[] <- c(-Inf, 1, rep(c(0,1),49))
assessDataType(ras)

ras <- raster(ncol = 10, nrow = 10)
ras[] <- c(Inf, 1, rep(c(0,1),49))
assessDataType(ras)

ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = -1.7e+30, max = 1.7e+308))
assessDataType(ras)

ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = 1.7e+30, max = 1.7e+308))
assessDataType(ras)

ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = -1.7e+308, max = -1))
assessDataType(ras)

# stack
ras <- raster(ncol = 10, nrow = 10)
ras[] <- rep(c(0,1),50)
ras1 <- raster(ncol = 10, nrow = 10)
ras1[] <- round(runif(100, min = -1.7e+308, max = -1))
sta <- stack(ras, ras1)
assessDataType(sta)
library(raster)

## Byte
ras <- raster(ncol = 10, nrow = 10)

ras[] <- 1:100
assessDataTypeGDAL(ras)

ras[] <- c(NA, 2:100)
assessDataTypeGDAL(ras)

##Int16
```

```
ras <- raster(ncol = 10, nrow = 10)

ras <- setValues(ras, -1:98)
assessDataTypeGDAL(ras)

ras[] <- c(NA, -1:97)
assessDataTypeGDAL(ras)

ras[] <- round(runif(100, min = -32767, max = 32767))
assessDataTypeGDAL(ras)

## UInt16
ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = 64000, max = 65000))
assessDataTypeGDAL(ras)


## UInt32
ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = 0, max = 500000000))
assessDataTypeGDAL(ras)

ras[14] <- NA
assessDataTypeGDAL(ras)

## Int32
ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = -200000000, max = 200000000))
assessDataTypeGDAL(ras)

ras[14] <- NA
assessDataTypeGDAL(ras)

## Float32
ras <- raster(ncol = 10, nrow = 10)
ras[] <- runif(100, min = -10, max = 87)
assessDataTypeGDAL(ras)

ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = -3.4e+26, max = 3.4e+28))
assessDataTypeGDAL(ras)

ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = 3.4e+26, max = 3.4e+28))
assessDataTypeGDAL(ras)

ras <- raster(ncol = 10, nrow = 10)
ras[] <- round(runif(100, min = -3.4e+26, max = -1))
assessDataTypeGDAL(ras)
```

---

basename2 *A version of* base::basename *that is* NULL *resistant*

Description

Returns NULL if x is NULL, otherwise, as basename.

Usage

```
basename2(x)
```

Arguments

x	A character vector of paths

Value

Same as basename

Cache *Cache method that accommodates environments, S4 methods, Rasters, & nested caching*

Description

A function that can be used to wrap around other functions to cache function calls for later use. This is normally most effective when the function to cache is slow to run, yet the inputs and outputs are small. The benefit of caching, therefore, will decline when the computational time of the "first" function call is fast and/or the argument values and return objects are large. The default setting (and first call to Cache) will always save to disk. The 2nd call to the same function will return from disk, unless options("reproducible.useMemoise" = TRUE), then the 2nd time will recover the object from RAM and is normally much faster (at the expense of RAM use).

Usage

```
Cache(
  FUN,
  ...,
  notOlderThan = NULL,
  .objects = NULL,
  .cacheExtra = NULL,
  outputObjects = NULL,
  algo = "xxhash64",
  cacheRepo = NULL,
  length = getOption("reproducible.length", Inf),
  compareRasterFileLength,
  userTags = c(),
  digestPathContent,
  omitArgs = NULL,
  classOptions = list(),
  debugCache = character(),
```

```
  sideEffect = FALSE,
  makeCopy = FALSE,
  quick = getOption("reproducible.quick", FALSE),
  verbose = getOption("reproducible.verbose", 1),
  cacheId = NULL,
  useCache = getOption("reproducible.useCache", TRUE),
  useCloud = FALSE,
  cloudFolderID = getOption("reproducible.cloudFolderID", NULL),
  showSimilar = getOption("reproducible.showSimilar", FALSE),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL)
)

## S4 method for signature 'ANY'
Cache(
  FUN,
  ...,
  notOlderThan = NULL,
  .objects = NULL,
  .cacheExtra = NULL,
  outputObjects = NULL,
  algo = "xxhash64",
  cacheRepo = NULL,
  length = getOption("reproducible.length", Inf),
  compareRasterFileLength,
  userTags = c(),
  digestPathContent,
  omitArgs = NULL,
  classOptions = list(),
  debugCache = character(),
  sideEffect = FALSE,
  makeCopy = FALSE,
  quick = getOption("reproducible.quick", FALSE),
  verbose = getOption("reproducible.verbose", 1),
  cacheId = NULL,
  useCache = getOption("reproducible.useCache", TRUE),
  useCloud = FALSE,
  cloudFolderID = getOption("reproducible.cloudFolderID", NULL),
  showSimilar = getOption("reproducible.showSimilar", FALSE),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL)
)
```

Arguments

FUN	Either a function or an unevaluated function call (e.g., using quote.
...	Arguments passed to FUN
notOlderThan	A time. Load an object from the Cache if it was created after this.

`.objects` Character vector of objects to be digested. This is only applicable if there is a list, environment (or similar) 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 similar objects. In the case of nested list-type objects, this will only be applied outermost first.

`.cacheExtra` A an arbitrary R object that will be included in the 'CacheDigest', but otherwise not passed into the `FUN`.

`outputObjects` Optional character vector indicating which objects to return. This is only relevant for list, environment (or similar) objects

`algo` The algorithms to be used; currently available choices are `md5`, which is also the default, `sha1`, `crc32`, `sha256`, `sha512`, `xxhash32`, `xxhash64`, `murmur32`, `spookyhash` and `blake3`.

`cacheRepo` A repository used for storing cached objects. This is optional if `Cache` is used inside a SpaDES module.

`length` Numeric. If the element passed to Cache is a `Path` class object (from e.g., `asPath(filename)`) or it is a `Raster` with file-backing, then this will be passed to `digest::digest`, essentially limiting the number of bytes to digest (for speed). This will only be used if `quick = FALSE`. Default is `getOption("reproducible.length")`, which is set to `Inf`.

`compareRasterFileLength`
Being deprecated; use `length`.

`userTags` A character vector with descriptions of the Cache function call. These will be added to the Cache so that this entry in the Cache can be found using `userTags` e.g., via showCache.

`digestPathContent`
Being deprecated. Use `quick`.

`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 `digest`, 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 or path. Determines where the function will look for new files following function completion. See Details. *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 or character. If TRUE, no disk-based information will be assessed, i.e., only memory content. See Details section about quick in Cache.

verbose — Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal

cacheId — Character string. If passed, this will override the calculated hash of the inputs, and return the result from this cacheId in the cacheRepo. Setting this is equivalent to manually saving the output of this function, i.e., the object will be on disk, and will be recovered in subsequent This may help in some particularly finicky situations where Cache is not correctly detecting unchanged inputs. This will guarantee the object will be identical each time; this may be useful in operational code.

useCache — Logical, numeric or "overwrite" or "devMode". See details.

useCloud — Logical. See Details.

cloudFolderID — A googledrive dribble of a folder, e.g., using drive_mkdir(). If left as NULL, the function will create a cloud folder with name from last two folder levels of the cacheRepo path, : paste0(basename(dirname(cacheRepo)),"_",basename(cacheRepo)). This cloudFolderID will be added to options("reproducible.cloudFolderID"), but this will not persist across sessions. If this is a character string, it will treat this as a folder name to create or use on GoogleDrive.

showSimilar — A logical or numeric. Useful for debugging. If TRUE or 1, then if the Cache does not find an identical archive in the cacheRepo, it will report (via message) the next most similar archive, and indicate which argument(s) is/are different. If a number larger than 1, then it will report the N most similar archived objects.

drv — an object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).

conn — A DBIConnection object, as returned by dbConnect().

Details

There are other similar functions in the R universe. This version of Cache has been used as part of a robust continuous workflow approach. As a result, we have tested it with many "non-standard" R objects (e.g., RasterLayer objects) and environments, which tend to be challenging for caching as they are always unique.

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. Uses `digest` (formerly fastdigest, which does not translate between operating systems). This is used for file-backed objects as well.
5. Cache will save arguments passed by user in a hidden environment. Any nested Cache functions will use arguments in this order 1) actual arguments passed at each Cache call, 2) any inherited arguments from an outer Cache call, 3) the default values of the Cache function. See section on *Nested Caching*.

Caching R objects using `archivist::cache` has five 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-based storage of information (specifically `RasterLayer-class` objects);
4. the default hashing algorithm is relatively slow.
5. heavily nested function calls may want Cache arguments to propagate through

As part of the SpaDES ecosystem of R packages, `Cache` can be used within SpaDES modules. If it is, 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 `archivist::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).

Nested Caching

Commonly, Caching is nested, i.e., an outer function is wrapped in a `Cache` function call, and one or more inner functions are also wrapped in a `Cache` function call. A user *can* always specify arguments in every Cache function call, but this can get tedious and can be prone to errors. The normal way that *R* handles arguments is it takes the user passed arguments if any, and default arguments for all those that have no user passed arguments. We have inserted a middle step. The order or precedence for any given `Cache` function call is 1. user arguments, 2. inherited arguments, 3. default arguments. At this time, the top level `Cache` arguments will propagate to all inner functions unless each individual `Cache` call has other arguments specified, i.e., "middle" nested `Cache` function calls don't propagate their arguments to further "inner" `Cache` function calls. See example.

`userTags` is unique of all arguments: its values will be appended to the inherited `userTags`.

quick

The quick argument is attempting to sort out an ambiguity with character strings: are they file paths or are they simply character strings. When quick = TRUE, Cache will treat these as character strings; when quick = FALSE, they will be attempted to be treated as file paths first; if there is no file, then it will revert to treating them as character strings. If user passes a character vector to this, then this will behave like omitArgs: quick = "file" will treat the argument "file" as character string.

The most often encountered situation where this ambiguity matters is in arguments about filenames: is the filename an input pointing to an object whose content we want to assess (e.g., a file-backed raster), or an output (as in saveRDS) and it should not be assessed. If only run once, the output file won't exist, so it will be treated as a character string. However, once the function has been run once, the output file will exist, and Cache(...) will assess it, which is incorrect. In these cases, the user is advised to use quick = "TheOutputFilenameArgument" to specify the argument whose content on disk should not be assessed, but whose character string should be assessed (distinguishing it from omitArgs = "TheOutputFilenameArgument", which will not assess the file content nor the character string).

This is relevant for objects of class character, Path and Raster currently. For class character, it is ambiguous whether this represents a character string or a vector of file paths. If it is known that character strings should not be treated as paths, then quick = TRUE will be much faster, with no loss of information. If it is file or directory, then it will digest the file content, or basename(object). For class Path objects, the file's metadata (i.e., filename and file size) will be hashed instead of the file contents if quick = TRUE. If set to FALSE (default), the contents of the file(s) are hashed. If quick = TRUE, length is ignored. Raster objects are treated as paths, if they are file-backed.

Caching Speed

Caching speed may become a critical aspect of a final product. For example, if the final product is a shiny app, rerunning the entire project may need to take less then a few seconds at most. There are 3 arguments that affect Cache speed: quick, length, and algo. quick is passed to .robustDigest, which currently only affects Path and Raster* class objects. In both cases, quick means that little or no disk-based information will be assessed.

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 quick = FALSE), or just the filename ((quick = TRUE)). 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.

useCache

Logical or numeric. If `FALSE` or `0`, then the entire Caching mechanism is bypassed and the function is evaluated as if it was not being Cached. Default is `getOption("reproducible.useCache"))`, which is `TRUE` by default, meaning use the Cache mechanism. This may be useful to turn all Caching on or off in very complex scripts and nested functions. Increasing levels of numeric values will cause deeper levels of Caching to occur. Currently, only implemented in `postProcess`: to do both caching of inner `cropInputs`, `projectInputs` and `maskInputs`, and caching of outer `postProcess`, use `useCache = 2`; to skip the inner sequence of 3 functions, use `useCache = 1`. For large objects, this may prevent many duplicated save to disk events.

If `"overwrite"` (which can be set with `options("reproducible.useCache" = "overwrite"))`, then the function invoke the caching mechanism but will purge any entry that is matched, and it will be replaced with the results of the current call.

If `"devMode"`: The point of this mode is to facilitate using the Cache when functions and datasets are continually in flux, and old Cache entries are likely stale very often. In 'devMode', the cache mechanism will work as normal if the Cache call is the first time for a function OR if it successfully finds a copy in the cache based on the normal Cache mechanism. It *differs* from the normal Cache if the Cache call does *not* find a copy in the 'cacheRepo', but it does find an entry that matches based on 'userTags'. In this case, it will delete the old entry in the 'cacheRepo' (identified based on matching 'userTags'), then continue with normal 'Cache'. For this to work correctly, 'userTags' must be unique for each function call. This should be used with caution as it is still experimental. Currently, if `userTags` are not unique to a single entry in the cacheRepo, it will default to the behaviour of `useCache = TRUE` with a message. This means that `"devMode"` is most useful if used from the start of a project.

`useCloud`

This is a way to store all or some of the local Cache in the cloud. Currently, the only cloud option is Google Drive, via **googledrive**. For this to work, the user must be or be able to be authenticated with `googledrive::drive_auth`. The principle behind this `useCloud` is that it will be a full or partial mirror of a local Cache. It is not intended to be used independently from a local Cache. To share objects that are in the Cloud with another person, it requires 2 steps. 1) share the `cloudFolderID$id`, which can be retrieved by `getOption("reproducible.cloudFolderID")$id` after at least one Cache call has been made. 2) The other user must then set their `cacheFolderID` in a `Cache\(...,reproducible.cloudFold = \"the ID here\"\)` call or set their option manually `options\(\"reproducible.cloudFolderID\" = \"the ID here\"\)`.

If `TRUE`, then this Cache call will download (if local copy doesn't exist, but cloud copy does exist), upload (local copy does or doesn't exist and cloud copy doesn't exist), or will not download nor upload if object exists in both. If `TRUE` will be at least 1 second slower than setting this to `FALSE`, and likely even slower as the cloud folder gets large. If a user wishes to keep "high-level" control, set this to `getOption("reproducible.useCloud",FALSE)` or `getOption("reproducible.useCloud",TRUE)`

(if the default behaviour should be FALSE or TRUE, respectively) so it can be turned on and off with this option. NOTE: *This argument will not be passed into inner/nested Cache calls.*)

sideEffect

If sideEffect is not FALSE, then metadata about any files that added to sideEffect will be added as an attribute to the cached copy. Subsequent calls to this function will assess for the presence of the new files in the sideEffect location. If the files are identical (quick = FALSE) or their file size is identical (quick = TRUE), then the cached copy of the function will be returned (and no files changed). If there are missing or incorrect files, then the function will re-run. This will accommodate the situation where the function call is identical, but somehow the side effect files were modified. If sideEffect is logical, then the function will check the cacheRepo; if it is a path, then it will check the path. The function will assess whether the files to be downloaded are found locally prior to download. If it fails the local test, then it will try to recover from a local copy if (makeCopy had been set to TRUE the first time the function was run. Currently, local recovery will only work ifmakeCOpy was set to TRUE the first time Cache was run). Default is FALSE.

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

showCache, clearCache, keepCache, CacheDigest, movedCache, .robustDigest, pipe

Examples

```
tmpDir <- file.path(tempdir())

# Basic use
ranNumsA <- Cache(rnorm, 10, 16, cacheRepo = tmpDir)

# All same
ranNumsB <- Cache(rnorm, 10, 16, cacheRepo = tmpDir) # recovers cached copy
ranNumsD <- Cache(quote(rnorm(n = 10, 16)), cacheRepo = tmpDir) # recovers cached copy

##############################################
# experimental devMode
##############################################
opt <- options("reproducible.useCache" = "devMode")
clearCache(tmpDir, ask = FALSE)
centralTendency <- function(x)
  mean(x)
```

```
funnyData <- c(1, 1, 1, 1, 10)
uniqueUserTags <- c("thisIsUnique", "reallyUnique")
ranNumsB <- Cache(centralTendency, funnyData, cacheRepo = tmpDir,
                  userTags = uniqueUserTags) # sets new value to Cache
showCache(tmpDir) # 1 unique artifact -- cacheId is 8be9cf2a072bdbb0515c5f0b3578f474

# During development, we often redefine function internals
centralTendency <- function(x)
  median(x)
# When we rerun, we don't want to keep the "old" cache because the function will
#   never again be defined that way. Here, because of userTags being the same,
#   it will replace the entry in the Cache, effetively overwriting it, even though
#   it has a different cacheId
ranNumsD <- Cache(centralTendency, funnyData, cacheRepo = tmpDir, userTags = uniqueUserTags)
showCache(tmpDir) # 1 unique artifact -- cacheId is bb1195b40c8d37a60fd6004e5d526e6b

# If it finds it by cacheID, doesn't matter what the userTags are
ranNumsD <- Cache(centralTendency, funnyData, cacheRepo = tmpDir, userTags = "thisIsUnique")

options(opt)

# For more in depth uses, see vignette
## Not run:
  # To use Postgres, set environment variables with the required credentials
  if (requireNamespace("RPostgres")) {
    Sys.setenv(PGHOST = "server.url")
    Sys.setenv(PGPORT = 5432)
    Sys.setenv(PGDATABASE = "mydatabase")
    Sys.setenv(PGUSER = "mydbuser")
    Sys.setenv(PGPASSWORD = "mysecurepassword")

    conn <- DBI::dbConnect(RPostgres::Postgres())
    options("reproducible.conn" = conn)

    # Will use postgres for cache data table, and tempdir() for saved R objects
    Cache(rnorm, 1, cacheRepo = tempdir())
  }

  browseVignettes(package = "reproducible")

## End(Not run)
```

`CacheDBFile` *A collection of low level tools for Cache*

Description

These are not intended for normal use.

Usage

```
CacheDBFile(
  cachePath = getOption("reproducible.cachePath"),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL)
)

CacheStorageDir(cachePath = getOption("reproducible.cachePath"))

CacheStoredFile(
  cachePath = getOption("reproducible.cachePath"),
  hash,
  format = getOption("reproducible.cacheSaveFormat", "rds")
)

CacheDBTableName(
  cachePath = getOption("reproducible.cachePath"),
  drv = getOption("reproducible.drv", RSQLite::SQLite())
)

CacheIsACache(
  cachePath = getOption("reproducible.cachePath"),
  create = FALSE,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL)
)
```

Arguments

cachePath	A path describing the directory in which to create the database file(s)
drv	an object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).
conn	A DBIConnection object, as returned by dbConnect().
hash	The cacheId or otherwise digested hash value, as character string.
format	The text string representing the file extension used normally by different save formats; currently only "rds" or "qs". Defaults to getOption("reproducible.cacheSaveFormat","rd
create	Logical. Currently only affects non RQSLite default drivers. If this is TRUE and there is no Cache database, the function will create one.

Details

CacheStoredFile returns the file path to the file with the specified hash value.

CacheStoredFile returns the file path to the file with the specified hash value.

CacheIsACache returns a logical of whether the specified cachePath is actually a functioning cache.

CacheDigest *The exact digest function that* Cache *uses*

Description

This can be used by a user to pre-test their arguments before running Cache, for example to determine whether there is a cached copy.

Usage

```
CacheDigest(
  objsToDigest,
  algo = "xxhash64",
  calledFrom = "Cache",
  quick = FALSE,
  ...
)
```

Arguments

objsToDigest	A list of all the objects (e.g., arguments) to be digested
algo	The algorithms to be used; currently available choices are md5, which is also the default, sha1, crc32, sha256, sha512, xxhash32, xxhash64, murmur32, spookyhash and blake3.
calledFrom	a Character string, length 1, with the function to compare with. Default is "Cache". All other values may not produce robust CacheDigest results.
quick	Logical or character. If TRUE, no disk-based information will be assessed, i.e., only memory content. See Details section about quick in Cache.
...	passed to .robustDigest; this is generally empty except for advanced use.

Value

A list of length 2 with the outputHash, which is the digest that Cache uses for cacheId and also preDigest, which is the digest of each sub-element in objsToDigest.

Examples

```
## Not run:
  a <- Cache(rnorm, 1)
  CacheDigest(list(rnorm, 1))


## End(Not run)
```

checkAndMakeCloudFolderID *Check for presence of* checkFolderID *(for* Cache(useCloud)*)*

Description

Will check for presence of a cloudFolderID and make a new one if one not present on Google Drive, with a warning.

Usage

```
checkAndMakeCloudFolderID(
  cloudFolderID = getOption("reproducible.cloudFolderID", NULL),
  cacheRepo = NULL,
  create = FALSE,
  overwrite = FALSE,
  verbose = getOption("reproducible.verbose", 1),
  team_drive = NULL
)
```

Arguments

cloudFolderID	The google folder ID where cloud caching will occur.
cacheRepo	A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
create	Logical. If TRUE, then the cloudFolderID will be created. This should be used with caution as there are no checks for overwriting. See googledrive::drive_mkdir. Default FALSE.
overwrite	Logical. Passed to googledrive::drive_mkdir.
verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal
team_drive	Logical indicating whether to check team drives.

checkGDALVersion *Check whether the system has a minimum version of GDAL available*

Description

Check whether the system has a minimum version of GDAL available

Usage

```
checkGDALVersion(version)
```

Arguments

version	The minimum GDAL version to check for.

Value

Logical.

Author(s)

Eliot McIntire and Alex Chubaty

Examples

```
## Not run:
  checkGDALVersion("2.0")

## End(Not run)
```

---

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

Description

Defunct.

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 is 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 git_repository class object, defined in **git2r**.

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)
```

`Checksums` *Calculate checksum*

Description

Verify (and optionally write) checksums. Checksums are computed using `.digest`, which is simply a wrapper around `digest::digest`.

Usage

```
Checksums(
  path,
  write,
  quickCheck = FALSE,
  checksumFile = file.path(path, "CHECKSUMS.txt"),
  files = NULL,
  verbose = getOption("reproducible.verbose", 1),
```

```
  ...
)

## S4 method for signature 'character,logical'
Checksums(
  path,
  write,
  quickCheck = FALSE,
  checksumFile = file.path(path, "CHECKSUMS.txt"),
  files = NULL,
  verbose = getOption("reproducible.verbose", 1),
  ...
)

## S4 method for signature 'character,missing'
Checksums(
  path,
  write,
  quickCheck = FALSE,
  checksumFile = file.path(path, "CHECKSUMS.txt"),
  files = NULL,
  verbose = getOption("reproducible.verbose", 1),
  ...
)
```

Arguments

path	Character string giving the directory path containing CHECKSUMS.txt file, or where it will be written if checksumFile = TRUE.
write	Logical indicating whether to overwrite CHECKSUMS.txt. Default is FALSE, as users should not change this file. Module developers should write this file prior to distributing their module code, and update accordingly when the data change.
quickCheck	Logical. If TRUE, then this will only use file sizes, rather than a digest::digest hash. This is generally faster, but will be *much* less robust.
checksumFile	The filename of the checksums file to read or write to. The default is 'CHECKSUMS.txt' located at file.path(path,module,"data",checksumFile). It is likely not a good idea to change this, and should only be used in cases such as Cache, which can evaluate if the checksumFile has changed.
files	An optional character string or vector of specific files to checksum. This may be very important if there are many files listed in a CHECKSUMS.txt file, but only a few are to be checksummed.
verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal
...	Passed to digest and write.table. For digest, the notable argument is algo. For write.table, the notable argument is append.

Value

A `data.table` with columns: `result`, `expectedFile`, `actualFile`, `checksum.x`, `checksum.y`, `algorithm.x`, `algorithm.y`, `filesize.x`, `filesize.y` indicating the result of comparison between local file (`x`) and expectation based on the `CHECKSUMS.txt` file.

Note

In version 1.2.0 and earlier, two checksums per file were required because of differences in the checksum hash values on Windows and Unix-like platforms. Recent versions use a different (faster) algorithm and only require one checksum value per file. To update your 'CHECKSUMS.txt' files using the new algorithm, see https://github.com/PredictiveEcology/SpaDES/issues/295#issuecomment-246513405.

Author(s)

Alex Chubaty

Examples

```
## Not run:
moduleName <- "my_module"
modulePath <- file.path("path", "to", "modules")

## verify checksums of all data files
Checksums(moduleName, modulePath)

## write new CHECKSUMS.txt file

# 1. verify that all data files are present (and no extra files are present)
list.files(file.path(modulePath, moduleName, "data"))

# 2. calculate file checksums and write to file (this will overwrite CHECKSUMS.txt)
Checksums(moduleName, modulePath, write = TRUE)

## End(Not run)
```

---

clearCache *Examining and modifying the cache*

Description

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

Usage

```
clearCache(
  x,
  userTags = character(),
  after = NULL,
  before = NULL,
  ask = getOption("reproducible.ask"),
  useCloud = FALSE,
  cloudFolderID = getOption("reproducible.cloudFolderID", NULL),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)

## S4 method for signature 'ANY'
clearCache(
  x,
  userTags = character(),
  after = NULL,
  before = NULL,
  ask = getOption("reproducible.ask"),
  useCloud = FALSE,
  cloudFolderID = getOption("reproducible.cloudFolderID", NULL),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)

cc(secs, ...)

showCache(
  x,
  userTags = character(),
  after = NULL,
  before = NULL,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)

## S4 method for signature 'ANY'
showCache(
  x,
  userTags = character(),
  after = NULL,
  before = NULL,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
```

```
  ...
)

keepCache(
  x,
  userTags = character(),
  after = NULL,
  before = NULL,
  ask = getOption("reproducible.ask"),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)

## S4 method for signature 'ANY'
keepCache(
  x,
  userTags = character(),
  after = NULL,
  before = NULL,
  ask = getOption("reproducible.ask"),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)
```

Arguments

x	A simList or a directory containing a valid Cache repository. Note: For compatibility with Cache argument, cacheRepo can also be used instead of x, though x will take precedence.
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 expression, 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.
ask	Logical. If FALSE, then it will not ask to confirm deletions using clearCache or keepCache. Default is TRUE
useCloud	Logical. If TRUE, then every object that is deleted locally will also be deleted in the cloudFolderID, if it is non-NULL

cloudFolderID	A googledrive dribble of a folder, e.g., using drive_mkdir(). If left as NULL, the function will create a cloud folder with name from last two folder levels of the cacheRepo path, : paste0(basename(dirname(cacheRepo)),"_",basename(cacheRepo)). This cloudFolderID will be added to options("reproducible.cloudFolderID"), but this will not persist across sessions. If this is a character string, it will treat this as a folder name to create or use on GoogleDrive.
drv	an object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).
conn	A DBIConnection object, as returned by dbConnect().
...	Other arguments. Currently, regexp, a logical, can be provided. This must be TRUE if the use is passing a regular expression. Otherwise, userTags will need to be exact matches. Default is missing, which is the same as TRUE. If there are errors due to regular expression problem, try FALSE. For cc, it is passed to clearCache, e.g., ask, userTags
secs	Currently 3 options: the number of seconds to pass to clearCache(after = secs), a POSIXct time e.g., from Sys.time(), or missing. If missing, the default, then it will delete the most recent entry in the Cache.

Details

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.

cc(secs) is just a shortcut for clearCache(repo = Paths$cachePath,after = secs), i.e., to remove any cache entries touched in the last secs seconds.

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

Note

If the cache is larger than 10MB, and clearCache is used, there will be a message and a pause, if interactive, to prevent accidentally deleting of a large cache repository.

See Also

mergeCache. Many more examples in Cache.

Examples

```
library(raster)

tmpDir <- file.path(tempdir(), "reproducible_examples", "Cache")
try(clearCache(tmpDir, ask = FALSE), silent = TRUE) # just to make sure it is clear

# Basic use
ranNumsA <- Cache(rnorm, 10, 16, cacheRepo = tmpDir)

# All same
ranNumsB <- Cache(rnorm, 10, 16, cacheRepo = tmpDir) # recovers cached copy
ranNumsD <- Cache(quote(rnorm(n = 10, 16)), cacheRepo = tmpDir) # recovers cached copy

# Any minor change makes it different
ranNumsE <- Cache(rnorm, 10, 6, cacheRepo = tmpDir) # different

## Example 1: basic cache use with tags
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
ranNumsB <- Cache(runif, 4, cacheRepo = tmpDir, userTags = "objectName:b")
ranNumsC <- Cache(runif, 40, cacheRepo = tmpDir, userTags = "objectName:b")

showCache(tmpDir, userTags = c("objectName"))
showCache(tmpDir, userTags = c("^a$")) # regular expression ... "a" exactly

# Fine control of cache elements -- pick out only the large runif object, and remove it
cache1 <- showCache(tmpDir, userTags = c("runif")) # show only cached objects made during runif
toRemove <- cache1[tagKey == "object.size"][as.numeric(tagValue) > 700]$cacheId
clearCache(tmpDir, userTags = toRemove, ask = FALSE)
cacheAfter <- showCache(tmpDir, userTags = c("runif")) # Only the small one is left

tmpDir <- file.path(tempdir(), "reproducible_examples", "Cache")
try(clearCache(tmpDir, ask = FALSE), silent = TRUE) # just to make sure it is clear

Cache(rnorm, 1, cacheRepo = tmpDir)
thisTime <- Sys.time()
Cache(rnorm, 2, cacheRepo = tmpDir)
Cache(rnorm, 3, cacheRepo = tmpDir)
Cache(rnorm, 4, cacheRepo = tmpDir)
showCache(x = tmpDir) # shows all 4 entries
cc(ask = FALSE, x = tmpDir)
showCache(x = tmpDir) # most recent is gone
cc(thisTime, ask = FALSE, x = tmpDir)
showCache(x = tmpDir) # all those after thisTime gone, i.e., only 1 left
cc(ask = FALSE, x = tmpDir) # Cache is
cc(ask = FALSE, x = tmpDir) # Cache is already empty
```

---

clearStubArtifacts *Clear erroneous archivist artifacts*

Description

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.

Details

Stub artifacts can result from several causes. The most common being erroneous removal of a file in the SQLite database. This can be caused sometimes if an archive object is being saved multiple times by multiple threads. This function will clear entries in the SQLite database which have no actual file with data.

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
showCache(tmpDir)

file2Remove <- dir(CacheStorageDir(tmpDir), full.name = TRUE)[1]
file.remove(file2Remove)
showCache(tmpDir) # repository directory still thinks files are there

# run clearStubArtifacts
suppressWarnings(clearStubArtifacts(tmpDir))
showCache(tmpDir) # stubs are removed

# cleanup
```

```
clearCache(tmpDir, ask = FALSE)
unlink(tmpDir, recursive = TRUE)
```

`cloudCache` *Deprecated*

Description

Usage

```
cloudCache(...)
```

Arguments

`...`	Passed to `Cache`

Details

Please use `Cache`, with args `useCloud` and `cloudFolderID`.

See Also

`cloudSyncCacheOld`, `Cache`, `cloudWriteOld`, `cloudCheckOld`

`cloudCheckOld` *Basic tool for using cloud-based caching*

Description

Very experimental

Usage

```
cloudCheckOld(toDigest, checksumsFileID = NULL, cloudFolderID = NULL)
```

Arguments

`toDigest`	The R object to consider, e.g., all the arguments to a function.
`checksumsFileID`	A google file ID where the checksums data.table is located, provided as a character string.
`cloudFolderID`	The google folder ID where a new checksums file should be written. This will only be used if `checksumsFileID` is not provided provided as a character string.

See Also

cloudSyncCacheOld, Cache, cloudWriteOld

---

cloudDownload *Download from cloud, if necessary*

Description

Meant for internal use, as there are internal objects as arguments.

Usage

```
cloudDownload(
  outputHash,
  newFileName,
  gdriveLs,
  cacheRepo,
  cloudFolderID,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL)
)
```

Arguments

outputHash	The cacheId of the object to upload
newFileName	The character string of the local filename that the downloaded object will have
gdriveLs	The result of googledrive::drive_ls(googledrive::as_id(cloudFolderID),pattern = "outputHash")
cacheRepo	A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
cloudFolderID	A googledrive dribble of a folder, e.g., using drive_mkdir(). If left as NULL, the function will create a cloud folder with name from last two folder levels of the cacheRepo path, : paste0(basename(dirname(cacheRepo)),"_",basename(cacheRepo)). This cloudFolderID will be added to options("reproducible.cloudFolderID"), but this will not persist across sessions. If this is a character string, it will treat this as a folder name to create or use on GoogleDrive.
drv	an object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).
conn	A DBIConnection object, as returned by dbConnect().

`cloudSyncCacheOld` *Sync cloud with local Cache*

Description

This is still experimental, see examples.

Usage

```
cloudSyncCacheOld(
  cacheRepo = getOption("reproducible.cachePath"),
  checksumsFileID = NULL,
  cloudFolderID = NULL,
  delete = TRUE,
  upload = TRUE,
  download = !delete,
  ask = getOption("reproducible.ask"),
  cacheIds = NULL,
  ...
)
```

Arguments

`cacheRepo`	See `x` in `showCache`
`checksumsFileID`	A google file ID where the checksums data.table is located, provided as a character string.
`cloudFolderID`	A googledrive dribble of a folder, e.g., using `drive_mkdir()`. If left as `NULL`, the function will create a cloud folder with name from last two folder levels of the `cacheRepo` path, : `paste0(basename(dirname(cacheRepo)),"_",basename(cacheRepo))`. This `cloudFolderID` will be added to `options("reproducible.cloudFolderID")`, but this will not persist across sessions. If this is a character string, it will treat this as a folder name to create or use on GoogleDrive.
`delete`	Logical. If `TRUE`, the default, it will delete any objects that are in `cloudFolderID` that are absent from local `cacheRepo`. If `FALSE`, it will not delete objects.
`upload`	Logical. If `TRUE`, the default, it will upload any objects identified by the internal `showCache(...)` call. See examples. If `FALSE`, then no files will be uploaded. Can be used in conjunction with `delete` to create behaviours similar to `clearCache` and `keepCache`.
`download`	Logical. If `FALSE`, the default, then the function will either delete the remote copy if `delete = TRUE` and there is no local copy, or upload the local copy if `upload = TRUE` and there is a local copy. If `TRUE`, then this will override `delete`, and download to local machine if it exists remotely.
`ask`	Logical. If `FALSE`, then it will not ask to confirm deletions using `clearCache` or `keepCache`. Default is `TRUE`

cacheIds	If supplied, then only this/these cacheId objects will be uploaded or deleted. Default is NULL, meaning do full sync (i.e., match cloudFolder with local cacheRepo, constrained by delete or upload)
...	Passed to showCache to get the artifacts to delete.

Details

cloudSyncCacheOld will remove any entries in a cloudCache that are not in a

See Also

cloudCache, Cache, cloudWriteOld, cloudCheckOld

---

cloudUpload *Upload to cloud, if necessary*

Description

Meant for internal use, as there are internal objects as arguments.

Usage

```
cloudUpload(isInRepo, outputHash, gdriveLs, cacheRepo, cloudFolderID, output)
```

Arguments

isInRepo	A data.table with the information about an object that is in the local cacheRepo
outputHash	The cacheId of the object to upload
gdriveLs	The result of googledrive::drive_ls(googledrive::as_id(cloudFolderID),pattern = "outputHash")
cacheRepo	A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
cloudFolderID	A googledrive dribble of a folder, e.g., using drive_mkdir(). If left as NULL, the function will create a cloud folder with name from last two folder levels of the cacheRepo path, : paste0(basename(dirname(cacheRepo)),"_",basename(cacheRepo)). This cloudFolderID will be added to options("reproducible.cloudFolderID"), but this will not persist across sessions. If this is a character string, it will treat this as a folder name to create or use on GoogleDrive.
output	The output object of FUN that was run in Cache

cloudWriteOld *Basic tool for using cloud-based caching*

Description

Very experimental

Usage

```
cloudWriteOld(
  object,
  digest,
  cloudFolderID = NULL,
  checksums,
  checksumsFileID,
  futurePlan = getOption("reproducible.futurePlan")
)
```

Arguments

object	The R object to write to cloud
digest	The cacheId of the input arguments, outputted from cloudCheckOld
cloudFolderID	The google folder ID where a new object should be written
checksums	A data.table that is outputted from cloudCheckOld that is the the checksums file
checksumsFileID	A google file ID where the checksums data.table is located, provided as a character string.
futurePlan	Which future::plan to use. Default: getOption("reproducible.futurePlan")

See Also

cloudSyncCacheOld, cloudCheckOld

compareNA *NA-aware comparison of two vectors*

Description

Copied from http://www.cookbook-r.com/Manipulating_data/Comparing_vectors_or_factors_with_NA/. This function returns TRUE wherever elements are the same, including NA's, and FALSE everywhere else.

Usage

```
compareNA(v1, v2)
```

Arguments

v1	A vector
v2	A vector

Examples

```
a <- c(NA, 1, 2, NA)
b <- c(1, NA, 2, NA)
compareNA(a, b)
```

---

convertPaths *Change the absolute path of a file*

Description

convertPaths is simply a wrapper around gsub for changing the first part of a path. convertRasterPaths is useful for changing the path to a file-backed raster (e.g., after copying the file to a new location).

Usage

```
convertPaths(x, patterns, replacements)

convertRasterPaths(x, patterns, replacements)
```

Arguments

x	For convertPaths, a character vector of file paths. For convertRasterPaths, a disk-backed RasterLayer object, or a list of such rasters.
patterns	Character vector containing a pattern to match (see ?gsub).
replacements	Character vector of the same length of patterns containing replacement text (see ?gsub).

Author(s)

Eliot McIntire and Alex Chubaty

Eliot McIntire and Alex Chubaty

Examples

```
filenames <- c("/home/user1/Documents/file.txt", "/Users/user1/Documents/file.txt")
oldPaths <- dirname(filenames)
newPaths <- c("/home/user2/Desktop", "/Users/user2/Desktop")
convertPaths(filenames, oldPaths, newPaths)

r1 <- raster::raster(system.file("external/test.grd", package = "raster"))
r2 <- raster::raster(system.file("external/rlogo.grd", package = "raster"))
rasters <- list(r1, r2)
oldPaths <- system.file("external", package = "raster")
newPaths <- file.path("~/rasters")
rasters <- convertRasterPaths(rasters, oldPaths, newPaths)
lapply(rasters, raster::filename)
```

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, ...)

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

## S4 method for signature 'SQLiteConnection'
Copy(object, ...)

## S4 method for signature 'data.table'
Copy(object, ...)

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

## S4 method for signature 'refClass'
Copy(object, ...)

## S4 method for signature 'data.frame'
Copy(object, ...)
```

```
## S4 method for signature 'Raster'
Copy(
  object,
  filebackedDir,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)
```

Arguments

object	An R object (likely containing environments) or an environment.
...	Only used for custom Methods
filebackedDir	A directory to copy any files that are backing R objects, currently only valid for Raster classes. Defaults to .reproducibleTempPath(), which is unlikely to be very useful. Can be NULL, which means that the file will not be copied and could therefore cause a collision as the pre-copied object and post-copied object would have the same file backing them.
drv	an object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).
conn	A DBIConnection object, as returned by dbConnect().

Details

To create a new Copy method for a class that needs its own method, try something like shown in example and put it in your package (or other R structure).

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

## Not run:
setMethod("Copy", signature = "the class", # where = specify here if not in a package,
  definition = function(object, filebackendDir, ...) {
  # write deep copy code here
})

## End(Not run)
```

`copySingleFile` *Copy a file using* `robocopy` *on Windows and* `rsync` *on Linux/macOS*

Description

This is replacement for `file.copy`, but for one file at a time. The additional feature is that it will use `robocopy` (on Windows) or `rsync` on Linux or Mac, if they exist. It will default back to `file.copy` if none of these exists. If there is a possibility that the file already exists, then this function should be very fast as it will do "update only", i.e., nothing.

Usage

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

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 for robocopy and –delete for rsync.
create	Passed to checkPath.
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")
tmpFile1 <- tempfile("file1", tmpDirFrom, ".csv")
tmpFile2 <- tempfile("file2", tmpDirFrom, ".csv")
checkPath(tmpDirFrom, create = TRUE)
f1 <- normalizePath(tmpFile1, mustWork = FALSE)
f2 <- normalizePath(tmpFile2, mustWork = FALSE)
t1 <- normalizePath(file.path(tmpDirTo, basename(tmpFile1)), mustWork = FALSE)
t2 <- normalizePath(file.path(tmpDirTo, basename(tmpFile2)), mustWork = FALSE)

write.csv(data.frame(a = 1:10, b = runif(10), c = letters[1:10]), f1)
write.csv(data.frame(c = 11:20, d = runif(10), e = letters[11:20]), f2)
copyFile(c(f1, f2), c(t1, t2))
file.exists(t1) ## TRUE
file.exists(t2) ## TRUE
identical(read.csv(f1), read.csv(f2)) ## FALSE
identical(read.csv(f1), read.csv(t1)) ## TRUE
identical(read.csv(f2), read.csv(t2)) ## TRUE

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

createCache *Create a new cache*

Description

Create a new cache

Low level tools to work with Cache

Usage

```
createCache(
  cachePath = getOption("reproducible.cachePath"),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  force = FALSE
)

saveToCache(
  cachePath = getOption("reproducible.cachePath"),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  obj,
  userTags,
  cacheId,
  linkToCacheId = NULL
)

loadFromCache(
  cachePath = getOption("reproducible.cachePath"),
  cacheId,
  format = getOption("reproducible.cacheSaveFormat", "rds"),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL)
)

rmFromCache(
  cachePath = getOption("reproducible.cachePath"),
  cacheId,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  format = getOption("reproducible.cacheSaveFormat", "rds")
)
```

Arguments

cachePath	A path describing the directory in which to create the database file(s)
drv	A driver, passed to dbConnect

conn	A DBIConnection object, as returned by dbConnect().
force	Logical. Should it create a cache in the cachePath, even if it already exists, overwriting.
obj	The R object to save to the cache
userTags	A character vector with descriptions of the Cache function call. These will be added to the Cache so that this entry in the Cache can be found using userTags e.g., via showCache.
cacheId	The hash string representing the result of .robustDigest
linkToCacheId	Optional. If a cacheId is provided here, then a file.link will be made to the file with that cacheId name in the cache repo. This is used when identical outputs exist in the cache. This will save disk space.
format	The text string representing the file extension used normally by different save formats; currently only "rds" or "qs". Defaults to getOption("reproducible.cacheSaveFormat","rd

---

cropInputs *Crop a* Spatial* *or* Raster* *object*

Description

This function can be used to crop or reproject module inputs from raw data.

Usage

```
cropInputs(
  x,
  studyArea,
  rasterToMatch,
  verbose = getOption("reproducible.verbose", 1),
  ...
)

## Default S3 method:
cropInputs(x, studyArea, rasterToMatch, ...)

## S3 method for class 'spatialClasses'
cropInputs(
  x,
  studyArea = NULL,
  rasterToMatch = NULL,
  verbose = getOption("reproducible.verbose", 1),
  extentToMatch = NULL,
  extentCRS = NULL,
  useGDAL = getOption("reproducible.useGDAL", TRUE),
  useCache = getOption("reproducible.useCache", FALSE),
  ...
```

```
)

## S3 method for class 'sf'
cropInputs(
  x,
  studyArea = NULL,
  rasterToMatch = NULL,
  verbose = getOption("reproducible.verbose", 1),
  extentToMatch = NULL,
  extentCRS = NULL,
  useCache = getOption("reproducible.useCache", FALSE),
  ...
)
```

Arguments

x	A Spatial*, sf, or Raster* object.
studyArea	SpatialPolygons* object used for masking and possibly cropping if no rasterToMatch is provided. If not in same CRS, then it will be spTransformed to CRS of x before masking. Currently, this function will not reproject the x. Optional in postProcess.
rasterToMatch	Template Raster* object used for cropping (so extent should be the extent of desired outcome) and reprojecting (including changing the resolution and projection). See details in postProcess.
verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal
...	Passed to raster::crop
extentToMatch	Optional. Can pass an extent here and a crs to extentCRS instead of rasterToMatch. These will override rasterToMatch, with a warning if both passed.
extentCRS	Optional. Can pass a crs here with an extent to extentTomatch instead of rasterToMatch
useGDAL	Logical or "force". Defaults to getOption("reproducible.useGDAL" = TRUE). If TRUE, then this function will use gdalwarp only when not small enough to fit in memory (i.e., *if the operation fails* the raster::canProcessInMemory(x,3) test). Using gdalwarp will usually be faster than raster::projectRaster, the function used if this is FALSE. Since since the two options use different algorithms, there may be different projection results. "force" will cause it to use GDAL regardless of the memory test described here.
useCache	Logical, default getOption("reproducible.useCache",FALSE), whether Cache is used internally.

Author(s)

Eliot McIntire, Jean Marchal, Ian Eddy, and Tati Micheletti

Examples

```
# Add a study area to Crop and Mask to
# Create a "study area"
library(sp)
library(raster)
ow <- setwd(tempdir())

# make a SpatialPolygon
coords1 <- structure(c(-123.98, -117.1, -80.2, -100, -123.98, 60.9, 67.73, 65.58, 51.79, 60.9),
                     .Dim = c(5L, 2L))
Sr1 <- Polygon(coords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
crs(shpEcozone) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
coords <- structure(c(-118.98, -116.1, -99.2, -106, -118.98, 59.9, 65.73, 63.58, 54.79, 59.9),
                    .Dim = c(5L, 2L))
Sr1 <- Polygon(coords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

##########
shpEcozonePostProcessed <- postProcess(shpEcozone, studyArea = StudyArea)

# Try manually, individual pieces
shpEcozoneReprojected <- projectInputs(shpEcozone, StudyArea)
shpEcozoneCropped <- cropInputs(shpEcozone, StudyArea)
shpEcozoneClean <- fixErrors(shpEcozone)
shpEcozoneMasked <- maskInputs(shpEcozone, StudyArea)

setwd(ow)
```

`determineFilename` *Determine filename, either automatically or manually*

Description

Determine the filename, given various combinations of inputs.

Usage

```
determineFilename(
  filename2 = NULL,
  filename1 = NULL,
  destinationPath = getOption("reproducible.destinationPath", "."),
  verbose = getOption("reproducible.verbose", 1),
  prefix = "Small",
```

```
  ...
)
```

Arguments

filename2	filename2 is optional, and is either NULL (no writing of outputs to disk), or several options for writing the object to disk. If TRUE (the default), it will give it a file name determined by .prefix(basename(filename1),prefix). If a character string, it will use this as its file name. See determineFilename.
filename1	Character strings giving the file paths of the *input* object (filename1) filename1 is only used for messaging (i.e., the object itself is passed in as x) and possibly naming of output (see details and filename2).
destinationPath	Optional. If filename2 is a relative file path, then this will be the directory of the resulting absolute file path.
verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal
prefix	The character string to prepend to filename1, if filename2 not provided.
...	Additional arguments passed to methods. For spatialClasses, these are: cropInputs, fixErrors, projectInputs, maskInputs, determineFilename, and writeOutputs. Each of these may also pass ... into other functions, like writeRaster, or sf::st_write. This might include potentially important arguments like datatype, format. Also passed to projectRaster, with likely important arguments such as method = "bilinear". See details.

... passed to::

cropInputs**:** crop

projectInputs projectRaster

maskInputs fastMask or intersect

fixErrors buffer

writeOutputs writeRaster or shapefile

determineFilename

* Can be overridden with useSAcrs ** Will mask with NAs from rasterToMatch if maskWithRTM

Details

The post processing workflow, which includes this function, addresses several scenarios, and depending on which scenario, there are several file names at play. For example, Raster objects may have file-backed data, and so *possess a file name*, whereas Spatial objects do not. Also, if post processing is part of a prepInputs workflow, there will always be a file downloaded. From the perspective of postProcess, these are the "inputs" or filename1. Similarly, there may or may not be a desire to write an object to disk after all post processing, filename2.

This subtlety means that there are two file names that may be at play: the "input" file name (`filename1`), and the "output" filename (`filename2`). When this is used within `postProcess`, it is straight forward.

However, when `postProcess` is used within a `prepInputs` call, the `filename1` file is the file name of the downloaded file (usually automatically known following the downloading, and refered to as `targetFile`) and the `filename2` is the file name of the of post-processed file.

If `filename2` is `TRUE`, i.e., not an actual file name, then the cropped/masked raster will be written to disk with the original `filenam1/targetFile` name, with `prefix` prefixed to the basename(`targetFile`).

If `filename2` is a character string, it will be the path of the saved/written object e.g., passed to `writeOutput`. It will be tested whether it is an absolute or relative path and used as is if absolute or prepended with `destinationPath` if relative.

If `filename2` is `logical`, then the output filename will be `prefix` prefixed to the basename(`filename1`). If a character string, it will be the path returned. It will be tested whether it is an absolute or relative path and used as is if absolute or prepended with `destinationPath` if provided, and if `filename2` is relative.

Examples

```
# Add a study area to Crop and Mask to
# Create a "study area"
library(sp)
library(raster)
ow <- setwd(tempdir())

# make a SpatialPolygon
coords1 <- structure(c(-123.98, -117.1, -80.2, -100, -123.98, 60.9, 67.73, 65.58, 51.79, 60.9),
                     .Dim = c(5L, 2L))
Sr1 <- Polygon(coords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
crs(shpEcozone) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
coords <- structure(c(-118.98, -116.1, -99.2, -106, -118.98, 59.9, 65.73, 63.58, 54.79, 59.9),
                    .Dim = c(5L, 2L))
Sr1 <- Polygon(coords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

##########
shpEcozonePostProcessed <- postProcess(shpEcozone, studyArea = StudyArea)

# Try manually, individual pieces
shpEcozoneReprojected <- projectInputs(shpEcozone, StudyArea)
shpEcozoneCropped <- cropInputs(shpEcozone, StudyArea)
shpEcozoneClean <- fixErrors(shpEcozone)
shpEcozoneMasked <- maskInputs(shpEcozone, StudyArea)
```

```
setwd(ow)
```

`downloadFile` *A wrapper around a set of downloading functions*

Description

Currently, this only deals with `drive_download`, and `download.file`.

Usage

```
downloadFile(
  archive,
  targetFile,
  neededFiles,
  destinationPath = getOption("reproducible.destinationPath"),
  quick,
  checksumFile,
  dlFun = NULL,
  checkSums,
  url,
  needChecksums,
  overwrite = getOption("reproducible.overwrite", TRUE),
  verbose = getOption("reproducible.verbose", 1),
  purge = FALSE,
  .tempPath,
  ...
)
```

Arguments

`archive`	Optional character string giving the path of an archive containing `targetFile`, or a vector giving a set of nested archives (e.g., `c("xxx.tar","inner.zip","inner.rar")`). If there is/are (an) inner archive(s), but they are unknown, the function will try all until it finds the `targetFile`. See table in `preProcess`. If it is `NA`, then it will *not* attempt to see it as an archive, even if it has archive-like file extension (e.g., `.zip`). This may be useful when an R function is expecting an archive directly.
`targetFile`	Character string giving the path to the eventual file (raster, shapefile, csv, etc.) after downloading and extracting from a zip or tar archive. This is the file *before* it is passed to `postProcess`. Currently, the internal checksumming does not checksum the file after it is `postProcessed` (e.g., cropped/reprojected/masked). Using `Cache` around `prepInputs` will do a sufficient job in these cases. See table in `preProcess`.
`neededFiles`	Character string giving the name of the file(s) to be extracted.

destinationPath
: Character string of a directory in which to download and save the file that comes from `url` and is also where the function will look for `archive` or `targetFile`. NOTE (still experimental): To prevent repeated downloads in different locations, the user can also set `options("reproducible.inputPaths")` to one or more local file paths to search for the file before attempting to download. Default for that option is `NULL` meaning do not search locally.

quick
: Logical. This is passed internally to `Checksums` (the quickCheck argument), and to `Cache` (the quick argument). This results in faster, though less robust checking of inputs. See the respective functions.

checksumFile
: A character string indicating the absolute path to the `CHECKSUMS.txt` file.

dlFun
: Optional "download function" name, such as `"raster::getData"`, which does custom downloading, in addition to loading into R. Still experimental.

checkSums
: A checksums file, e.g., created by Checksums(..., write = TRUE)

url
: Optional character string indicating the URL to download from. If not specified, then no download will be attempted. If not entry exists in the `CHECKSUMS.txt` (in `destinationPath`), an entry will be created or appended to. This `CHECKSUMS.txt` entry will be used in subsequent calls to `prepInputs` or `preProcess`, comparing the file on hand with the ad hoc `CHECKSUMS.txt`. See table in `preProcess`.

needChecksums
: A numeric, with `0` indicating do not write a new checksums, `1` write a new one, `2` append new information to existing one.

overwrite
: Logical. Should downloading and all the other actions occur even if they pass the checksums or the files are all there.

verbose
: Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., `options('reproducible.verbose' = 0)` to reduce to minimal

purge
: Logical or Integer. `0/FALSE` (default) keeps existing `CHECKSUMS.txt` file and `prepInputs` will write or append to it. `1/TRUE` will deleted the entire `CHECKSUMS.txt` file. Other options, see details.

.tempPath
: Optional temporary path for internal file intermediate steps. Will be cleared on.exit from this function.

...
: Passed to `dlFun`. Still experimental.

Author(s)

Eliot McIntire

`extractFromArchive` *Extract files from archive*

Description

Extract zip or tar archive files, possibly nested in other zip or tar archives.

Usage

```
extractFromArchive(
  archive,
  destinationPath = getOption("reproducible.destinationPath", dirname(archive)),
  neededFiles = NULL,
  extractedArchives = NULL,
  checkSums = NULL,
  needChecksums = 0,
  filesExtracted = character(),
  checkSumFilePath = character(),
  quick = FALSE,
  verbose = getOption("reproducible.verbose", 1),
  .tempPath,
  ...
)
```

Arguments

`archive`	Character string giving the path of the archive containing the `file` to be extracted. This path must exist or be `NULL`
`destinationPath`	Character string giving the path where `neededFiles` will be extracted. Defaults to the archive directory.
`neededFiles`	Character string giving the name of the file(s) to be extracted.
`extractedArchives`	Used internally to track archives that have been extracted from.
`checkSums`	A checksums file, e.g., created by Checksums(..., write = TRUE)
`needChecksums`	A numeric, with `0` indicating do not write a new checksums, `1` write a new one, `2` append new information to existing one.
`filesExtracted`	Used internally to track files that have been extracted.
`checkSumFilePath`	The full path to the checksum.txt file
`quick`	Passed to `Checksums`
`verbose`	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., `options('reproducible.verbose' = 0)` to `reduce to minimal`

.tempPath	Optional temporary path for internal file intermediate steps. Will be cleared on.exit from this function.
...	Passed to unzip or untar, e.g., overwrite

Value

A character vector listing the paths of the extracted archives.

Author(s)

Jean Marchal and Eliot McIntire

---

fastMask *Faster operations on rasters*

Description

This alternative to raster::mask is included here.

Usage

```
fastMask(
  x,
  y,
  cores = NULL,
  useGDAL = getOption("reproducible.useGDAL", TRUE),
  verbose = getOption("reproducible.verbose", 1),
  ...
)
```

Arguments

x	A Raster* object.
y	A SpatialPolygons object. If it is not in the same projection as x, it will be reprojected on the fly to that of x
cores	An integer* or 'AUTO'. This will be used if gdalwarp is triggered. 'AUTO' will calculate 90 number of cores in the system, while an integer or rounded float will be passed as the exact number of cores to be used.
useGDAL	Logical or "force". Defaults to getOption("reproducible.useGDAL" = TRUE). If TRUE, then this function will use gdalwarp only when not small enough to fit in memory (i.e., *if the operation fails* the raster::canProcessInMemory(x,3) test). Using gdalwarp will usually be faster than raster::projectRaster, the function used if this is FALSE. Since since the two options use different algorithms, there may be different projection results. "force" will cause it to use GDAL regardless of the memory test described here.

verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., `options('reproducible.verbose' = 0)` to reduce to `minimal`
...	Currently unused.

Value

A `Raster*` object, masked (i.e., smaller extent and/or several pixels converted to NA)

Author(s)

Eliot McIntire

Examples

```
library(sp)
library(raster)

Sr1 <- Polygon(cbind(c(2, 4, 4, 0.9, 2), c(2, 3, 5, 4, 2)))
Sr2 <- Polygon(cbind(c(5, 4, 2, 5), c(2, 3, 2, 2)))
Sr3 <- Polygon(cbind(c(4, 4, 5, 10, 4), c(5, 3, 2, 5, 5)))

Srs1 <- Polygons(list(Sr1), "s1")
Srs2 <- Polygons(list(Sr2), "s2")
Srs3 <- Polygons(list(Sr3), "s3")
shp <- SpatialPolygons(list(Srs1, Srs2, Srs3), 1:3)
d <- data.frame(vals = 1:3, other = letters[3:1], stringsAsFactors = FALSE)
row.names(d) <- names(shp)
shp <- SpatialPolygonsDataFrame(shp, data = d)
poly <- list()
poly[[1]] <- raster(raster::extent(shp), vals = 0, res = c(1, 1))
poly[[2]] <- raster(raster::extent(shp), vals = 1, res = c(1, 1))
origStack <- stack(poly)
# original mask function in raster
newStack1 <- mask(x= origStack, mask = shp)
newStack2 <- fastMask(x = origStack, y = shp)

# test all equal
all.equal(newStack1, newStack2)

newStack1 <- stack(newStack1)
newStack2 <- stack(newStack2)

if (interactive()) {
  plot(newStack2[[1]])
  plot(shp, add = TRUE)
}
```

file.move *Move a file to a new location*

Description

Move a file to a new location

Usage

```
file.move(from, to, overwrite = FALSE)
```

Arguments

from, to	character vectors, containing file names or paths.
overwrite	logical indicating whether to overwrite destination file if it exists.

Value

Logical indicating whether operation succeeded.

Filenames *Return the filename(s) from a* Raster* *object*

Description

This is mostly just a wrapper around filename from the **raster** package, except that instead of returning an empty string for a RasterStack object, it will return a vector of length >1 for RasterStack.

Usage

```
Filenames(obj, allowMultiple = TRUE)

## S4 method for signature 'ANY'
Filenames(obj, allowMultiple = TRUE)

## S4 method for signature 'Raster'
Filenames(obj, allowMultiple = TRUE)

## S4 method for signature 'RasterStack'
Filenames(obj, allowMultiple = TRUE)

## S4 method for signature 'environment'
Filenames(obj, allowMultiple = TRUE)

## S4 method for signature 'list'
Filenames(obj, allowMultiple = TRUE)
```

Arguments

obj	A Raster* object (i.e., RasterLayer, RasterStack, RasterBrick)
allowMultiple	Logical. If TRUE, the default, then all relevant filenames will be returned, i.e., in cases such as .grd where multiple files are required. If FALSE, then only the first file will be returned, e.g., filename.grd, in the case of default Raster format in R.

Author(s)

Eliot McIntire

---

getGDALVersion *Check the GDAL version in use*

Description

Check the GDAL version in use

Usage

```
getGDALVersion()
```

Value

numeric_version

Author(s)

Alex Chubaty and Eliot McIntire

---

isTopLevelEnv *Determine if an environment is a top level environment*

Description

Here, we define that as .GlobalEnv, any namespace, emptyenv, or baseenv. This is useful to determine the effective size of an R function, due to R including the objects from enclosing environments

Usage

```
isTopLevelEnv(x)
```

Arguments

x	Any environment

Value

A logical. FALSE if it is not one of the "Top Level Environments", TRUE otherwise.

---

linkOrCopy *Hardlink, symlink, or copy a file*

Description

Attempt first to make a hardlink. If that fails, try to make a symlink (on non-windows systems and symlink = TRUE). If that fails, copy the file.

Usage

```
linkOrCopy(
  from,
  to,
  symlink = TRUE,
  verbose = getOption("reproducible.verbose", 1)
)
```

Arguments

from, to	Character vectors, containing file names or paths. to can alternatively be the path to a single existing directory.
symlink	Logical indicating whether to use symlink (instead of hardlink). Default FALSE.
verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal

Note

Use caution with files-backed objects (e.g., rasters). See examples.

Author(s)

Alex Chubaty and Eliot McIntire

See Also

file.link, file.symlink, file.copy.

Examples

```
library(datasets)
library(magrittr)
library(raster)

tmpDir <- file.path(tempdir(), "symlink-test") %>%
  normalizePath(winslash = '/', mustWork = FALSE)
dir.create(tmpDir)

f0 <- file.path(tmpDir, "file0.csv")
write.csv(iris, f0)

d1 <- file.path(tmpDir, "dir1")
dir.create(d1)
write.csv(iris, file.path(d1, "file1.csv"))

d2 <- file.path(tmpDir, "dir2")
dir.create(d2)
f2 <- file.path(tmpDir, "file2.csv")

## create link to a file
linkOrCopy(f0, f2)
file.exists(f2) ## TRUE
identical(read.table(f0), read.table(f2)) ## TRUE

## deleting the link shouldn't delete the original file
unlink(f0)
file.exists(f0) ## FALSE
file.exists(f2) ## TRUE

## using rasters and other file-backed objects
f3a <- system.file("external/test.grd", package = "raster")
f3b <- system.file("external/test.gri", package = "raster")
r3a <- raster(f3a)
f4a <- file.path(tmpDir, "raster4.grd")
f4b <- file.path(tmpDir, "raster4.gri")
linkOrCopy(f3a, f4a) ## hardlink
linkOrCopy(f3b, f4b) ## hardlink
r4a <- raster(f4a)

isTRUE(all.equal(r3a, r4a)) # TRUE

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

`makeMemoisable` *Generic method to make or unmake objects memoisable*

Description

This is just a pass through for all classes in **reproducible**. This generic is here so that downstream methods can be created.

Usage

```
makeMemoisable(x)

## Default S3 method:
makeMemoisable(x)

unmakeMemoisable(x)

## Default S3 method:
unmakeMemoisable(x)
```

Arguments

x	An object to make memoisable. See individual methods in other packages.

Value

The same object, but with any modifications, especially dealing with saving of environments, which memoising doesn't handle correctly in some cases.

---

maskInputs *Mask module inputs*

Description

This function can be used to mask inputs from data. Masking here is equivalent to `raster::mask` (though `fastMask` is used here) or `raster::intersect`.

Usage

```
maskInputs(x, studyArea, ...)

## S3 method for class 'Raster'
maskInputs(
  x,
  studyArea,
  rasterToMatch,
  maskWithRTM = NULL,
  verbose = getOption("reproducible.verbose", 1),
  ...
)
```

```
## S3 method for class 'Spatial'
maskInputs(
  x,
  studyArea,
  rasterToMatch,
  maskWithRTM = FALSE,
  verbose = getOption("reproducible.verbose", 1),
  useCache = getOption("reproducible.useCache", FALSE),
  ...
)

## S3 method for class 'sf'
maskInputs(
  x,
  studyArea,
  verbose = getOption("reproducible.verbose", 1),
  useCache = getOption("reproducible.useCache", FALSE),
  ...
)
```

Arguments

x	An object to do a geographic raster::mask/raster::intersect. See methods.
studyArea	SpatialPolygons* object used for masking and possibly cropping if no rasterToMatch is provided. If not in same CRS, then it will be spTransformed to CRS of x before masking. Currently, this function will not reproject the x. Optional in postProcess.
...	Passed to methods. None currently implemented.
rasterToMatch	Template Raster* object used for cropping (so extent should be the extent of desired outcome) and reprojecting (including changing the resolution and projection). See details in postProcess.
maskWithRTM	Logical. If TRUE, then the default,
verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal
useCache	Logical, default getOption("reproducible.useCache",FALSE), whether Cache is used internally.

Author(s)

Eliot McIntire and Jean Marchal

Examples

```
# Add a study area to Crop and Mask to
```

```
# Create a "study area"
library(sp)
library(raster)
ow <- setwd(tempdir())

# make a SpatialPolygon
coords1 <- structure(c(-123.98, -117.1, -80.2, -100, -123.98, 60.9, 67.73, 65.58, 51.79, 60.9),
                     .Dim = c(5L, 2L))
Sr1 <- Polygon(coords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
crs(shpEcozone) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
coords <- structure(c(-118.98, -116.1, -99.2, -106, -118.98, 59.9, 65.73, 63.58, 54.79, 59.9),
                    .Dim = c(5L, 2L))
Sr1 <- Polygon(coords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

##########
shpEcozonePostProcessed <- postProcess(shpEcozone, studyArea = StudyArea)

# Try manually, individual pieces
shpEcozoneReprojected <- projectInputs(shpEcozone, StudyArea)
shpEcozoneCropped <- cropInputs(shpEcozone, StudyArea)
shpEcozoneClean <- fixErrors(shpEcozone)
shpEcozoneMasked <- maskInputs(shpEcozone, StudyArea)

setwd(ow)
```

---

mergeCache *Merge two cache repositories together*

Description

Usage

```
mergeCache(
  cacheTo,
  cacheFrom,
  drvTo = getOption("reproducible.drv", RSQLite::SQLite()),
  drvFrom = getOption("reproducible.drv", RSQLite::SQLite()),
  connTo = NULL,
  connFrom = NULL
)
```

```
## S4 method for signature 'ANY'
mergeCache(
  cacheTo,
  cacheFrom,
  drvTo = getOption("reproducible.drv", RSQLite::SQLite()),
  drvFrom = getOption("reproducible.drv", RSQLite::SQLite()),
  connTo = NULL,
  connFrom = NULL
)
```

Arguments

cacheTo	The cache repository (character string of the file path) that will become larger, i.e., merge into this
cacheFrom	The cache repository (character string of the file path) from which all objects will be taken and copied from
drvTo	The database driver for the cacheTo.
drvFrom	The database driver for the cacheFrom
connTo	The connection for the cacheTo. If not provided, then a new one will be made from drvTo and cacheTo
connFrom	The database for the cacheFrom. If not provided, then a new one will be made from drvFrom and cacheFrom

Details

All the cacheFrom artifacts will be put into cacheTo repository. All userTags will be copied verbatim, including accessed, with 1 exception: date will be the current Sys.time() at the time of merging. The createdDate column will be similarly the current time of merging.

Value

The character string of the path of cacheTo, i.e., not the objects themselves.

---

messageDF *Use message to print a clean square data structure*

Description

Sends to message, but in a structured way so that a data.frame-like can be cleanly sent to messaging.

Usage

```
messageDF(df, round, colour = NULL, colnames = NULL)
```

Arguments

df	A data.frame, data.table, matrix
round	An optional numeric to pass to round
colour	Passed to getFromNamespace(colour,ns = "crayon"), so any colour that crayon can use
colnames	Logical or NULL. If TRUE, then it will print column names even if there aren't any in the df (i.e., they will) be V1 etc., NULL will print them if they exist, and FALSE which will omit them.

---

movedCache *Deal with moved cache issues*

Description

If a user manually copies a complete Cache folder (including the db file and rasters folder), there are issues that must be addressed. Primarily, the db table must be renamed. Run this function after a manual copy of a cache folder. See examples for one way to do that.

Usage

```
movedCache(
  new,
  old,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL)
)
```

Arguments

new	Either the path of the new cachePath where the cache was moved or copied to, or the new DB Table Name
old	Optional, if there is only one table in the new cache path. Either the path of the previous cachePath where the cache was moved or copied from, or the old DB Table Name
drv	an object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).
conn	A DBIConnection object, as returned by dbConnect().

Examples

```
tmpCache <- file.path(tempdir(), "tmpCache")
tmpdir <- file.path(tempdir(), "tmpdir")
bb <- Cache(rnorm, 1, cacheRepo = tmpCache)

# Copy all files from tmpCache to tmpdir
froms <- normPath(dir(tmpCache, recursive = TRUE, full.names = TRUE))
```

```
checkPath(file.path(tmpdir, "rasters"), create = TRUE)
checkPath(file.path(tmpdir, "cacheOutputs"), create = TRUE)
file.copy(from = froms, overwrite = TRUE,
          to = gsub(normPath(tmpCache), normPath(tmpdir), froms))

# Must use 'movedCache' to update the database table
movedCache(new = tmpdir, old = tmpCache)
bb <- Cache(rnorm, 1, cacheRepo = tmpdir) # should recover the previous call
```

---

objSize *Recursive* object.size

Description

This has methods for various types of things that may not correctly report their object size using object.size. Also, for lists and environments, it will return the object size separately for each element. These are estimates only, and could be inaccurate. Alternative, similar functions include object.size and pryr::object_size. See Details for the special case of functions and their enclosing environments.

Usage

```
objSize(x, quick, enclosingEnvs, .prevEnvirs, ...)

## Default S3 method:
objSize(
  x,
  quick = getOption("reproducible.quick", FALSE),
  enclosingEnvs = TRUE,
  .prevEnvirs = list(),
  ...
)

## S3 method for class 'list'
objSize(
  x,
  quick = getOption("reproducible.quick", FALSE),
  enclosingEnvs = TRUE,
  .prevEnvirs = list(),
  ...
)

## S3 method for class 'environment'
objSize(
  x,
  quick = getOption("reproducible.quick", FALSE),
  enclosingEnvs = TRUE,
```

```
  .prevEnvirs = list(),
  ...
)

## S3 method for class 'Path'
objSize(
  x,
  quick = getOption("reproducible.quick", FALSE),
  enclosingEnvs = TRUE,
  .prevEnvirs = list(),
  ...
)

## S3 method for class '`function`'
objSize(
  x,
  quick = getOption("reproducible.quick", FALSE),
  enclosingEnvs = TRUE,
  .prevEnvirs = list(),
  ...
)

objSizeSession(sumLevel = Inf, enclosingEnvs = TRUE, .prevEnvirs = list())
```

Arguments

x	An object
quick	Logical. Only some methods use this. e.g., Path class objects. In which case, file.size will be used instead of object.size.
enclosingEnvs	Logical indicating whether to include enclosing environments. Default TRUE.
.prevEnvirs	For internal account keeping to identify and prevent duplicate counting
...	Additional arguments (currently unused)
sumLevel	Numeric, indicating at which depth in the list of objects should the object sizes be summed (summarized). Default is Inf, meaning no sums. Currently, the only option other than Inf is 1: objSizeSession(1), which gives the size of each package.

Details

For functions, a user can include the enclosing environment as described https://www.r-bloggers.com/2015/03/using-closures-as-objects-in-r/ and http://adv-r.had.co.nz/memory.html. It is not entirely clear which estimate is better. However, if the enclosing environment is the .GlobalEnv, it will not be included even though enclosingEnvs = TRUE.

objSizeSession will give the size of the whole session, including loaded packages. Because of the difficulties in calculating the object size of base and methods packages and Autoloads, these are omitted.

Examples

```
library(utils)

foo <- new.env()
foo$b <- 1:10
foo$d <- 1:10

objSize(foo) # all the elements in the environment
object.size(foo) # different - only measuring the environment as an object

object.size(prepInputs) # only the function, without its enclosing environment
objSize(prepInputs)     # the function, plus its enclosing environment

# Size of all packages; includes their imported functions
## Not run:
  bar <- objSizeSession(1)
  print(bar, units = "auto")

## End(Not run)

os1 <- object.size(as.environment("package:reproducible"))
os2 <- objSize(as.environment("package:reproducible"))
(os1) # very small -- just the environment container
sum(unlist(os2)) # around 13 MB, with all functions, objects
                 # and imported functions
```

---

paddedFloatToChar *Convert numeric to character with padding*

Description

This will pad floating point numbers, right or left. For integers, either class integer or functionally integer (e.g., 1.0), it will not pad right of the decimal. For more specific control or to get exact padding right and left of decimal, try the `stringi` package. It will also not do any rounding. See examples.

Usage

```
paddedFloatToChar(x, padL = ceiling(log10(x + 1)), padR = 3, pad = "0")
```

Arguments

x	numeric. Number to be converted to character with padding
padL	numeric. Desired number of digits on left side of decimal. If not enough, pad will be used to pad.
padR	numeric. Desired number of digits on right side of decimal. If not enough, pad will be used to pad.
pad	character to use as padding (nchar(pad) == 1 must be TRUE).

Value

Character string representing the filename.

Author(s)

Eliot McIntire and Alex Chubaty

Examples

```
paddedFloatToChar(1.25)
paddedFloatToChar(1.25, padL = 3, padR = 5)
paddedFloatToChar(1.25, padL = 3, padR = 1) # no rounding, so keeps 2 right of decimal
```

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, nParentDirs = 0)

## S3 method for class 'character'
asPath(obj, nParentDirs = 0)

## S3 method for class 'null'
asPath(obj, nParentDirs = 0)
```

Arguments

obj	A character string to convert to a Path.
nParentDirs	A numeric indicating the number of parent directories starting from basename(obj) = 0 to keep for the digest

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 function can be used to remove any ambiguity about whether a character string is a path. It is primarily useful for achieving repeatability with Caching. Essentially, when Caching, arguments that are character strings should generally 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 expanding 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
```

---

pipe *A cache-aware pipe (currently not working)*

Description

With updates to magrittr to version 2.0, this Cache pipe is now broken. We are working on an update.

This pipe can only be used at any point in a pipe chain, but must be preceded by Cache(...) (which allows other Cache() %C% ... remaining pipes arguments to be passed).

This will take the input arguments of the first function immediately following the Cache() and the pipe chain until the special %C%, evaluate them both against the cacheRepo argument in Cache. If they exist, then the entire pipe chain will be skipped, and only the previous final result will be given. If there is no previous cached copy of the initial function's arguments, then all chain elements will be evaluated. The final result will be cached for future use. Therefore, the entire chain must be identical. The required usage should be straight forward to insert into existing code that uses pipes (Cache() %C% ... remaining pipes).

Still experimental and may change. This form cannot pass any arguments to]codeCache, such as cacheRepo, thus it is of limited utility. However, it is a clean alternative for simple cases.

Usage

```
lhs %C% rhs

lhs %<% rhs
```

Arguments

lhs	A name to assign to.
rhs	A function call

Examples

```
# THIS IS CURRENTLY BROKEN DUE TO UPGRADES TO INTERNALS OF magrittr %>%
library(magrittr) # standard pipe
## Not run:  # these can't be automatically run due to package conflicts with magrittr
tmpdir <- file.path(tempdir(), "testCache")
checkPath(tmpdir, create = TRUE)
a <- rnorm(10, 16) %>%
     mean() %>%
     prod(., 6)
b <- Cache(cacheRepo = tmpdir) %C% # use of the %C% pipe!
     rnorm(10, 16) %>% # everything after here is NOT cached!
     mean() %>%
     prod(., 6)
d <- Cache(cacheRepo = tmpdir) %C%
     rnorm(10, 16) %>%
     mean() %>%
     prod(., 6)
e <- Cache(cacheRepo = tmpdir) %C%
     rnorm(10, 16) %>%
     mean() %>%
     prod(., 5) # changed
all.equal(b,d) # TRUE
all.equal(a,d) # different because 'a' uses a unique rnorm, 'd' uses the Cached rnorm
               #   because the arguments to rnorm, i.e., 10 and 16, and
               #   the subsequent functions in the chain, are identical
all.equal(a,e) # different because the final function, prod, has a changed argument.

############
# multiple random elements shows Cached sequence up to %C%
a1 <- Cache(cacheRepo = tmpdir) %>%
       seq(1, 10) %>%
       rnorm(2, mean = .) %>%
       mean() %C%               # Cache pipe here --
                                # means this pipe is the last one that is Cached
       rnorm(3, mean = .) %>%
       mean(.) %>%
       rnorm(4, mean = .)  # Random 4 numbers, the mean is same each time
a2 <- Cache(cacheRepo = tmpdir) %>%
       seq(1, 10) %>%
       rnorm(2, mean = .) %>%
       mean() %C%               # Cache pipe here --
                                # means this pipe is the last one that is Cached
       rnorm(3, mean = .) %>%
       mean(.) %>%
       rnorm(4, mean = .)  # Random 4 numbers, the mean is same each time
sum(a1 - a2) # not 0 # i.e., numbers are different

# NOW DO WITH CACHE AT END
b1 <- Cache(cacheRepo = tmpdir) %>%
       seq(1, 10) %>%
       rnorm(2, mean = .) %>%
       mean() %>%
```

```
                                # means this pipe is the last one that is Cached
       rnorm(3, mean = .) %>%
       mean(.) %C%              # Cache pipe here --
       rnorm(4, mean = .)       # These are samethe mean is same each time
b2 <- Cache(cacheRepo = tmpdir) %>%
       seq(1, 10) %>%
       rnorm(2, mean = .) %>%
       mean() %>%
                                # means this pipe is the last one that is Cached
       rnorm(3, mean = .) %>%
       mean(.) %C%              # Cache pipe here --
       rnorm(4, mean = .)       # These are samethe mean is same each time
sum(b1 - b2) # 0 # i.e., numbers are same

unlink(tmpdir, recursive = TRUE)

## End(Not run)
# Equivalent
a <- Cache(rnorm, 1)
b %<% rnorm(1)
```

`postProcess` *Generic function to post process objects*

Description

The method for `spatialClasses` (`Raster*` and `Spatial*`) will crop, reproject, and mask, in that order. This is a wrapper for `cropInputs`, `fixErrors`, `projectInputs`, `maskInputs` and `writeOutputs`, with a decent amount of data manipulation between these calls so that the crs match.

Usage

```
postProcess(x, ...)

## Default S3 method:
postProcess(x, ...)

## S3 method for class 'list'
postProcess(x, ...)

## S3 method for class 'spatialClasses'
postProcess(
  x,
  filename1 = NULL,
  filename2 = NULL,
  studyArea = NULL,
  rasterToMatch = NULL,
  overwrite = getOption("reproducible.overwrite", TRUE),
```

```
  useSAcrs = FALSE,
  useCache = getOption("reproducible.useCache", FALSE),
  verbose = getOption("reproducible.verbose", 1),
  ...
)

## S3 method for class 'sf'
postProcess(
  x,
  filename1 = NULL,
  filename2 = NULL,
  studyArea = NULL,
  rasterToMatch = NULL,
  overwrite = getOption("reproducible.overwrite", TRUE),
  useSAcrs = FALSE,
  useCache = getOption("reproducible.useCache", FALSE),
  verbose = getOption("reproducible.verbose", 1),
  ...
)
```

Arguments

x	An object of postProcessing, e.g., `spatialClasses`. See individual methods. This can be provided as a `rlang::quosure` or a normal R object.

...	Additional arguments passed to methods. For `spatialClasses`, these are: `cropInputs`, `fixErrors`, `projectInputs`, `maskInputs`, `determineFilename`, and `writeOutputs`. Each of these may also pass `...` into other functions, like `writeRaster`, or `sf::st_write`. This might include potentially important arguments like `datatype`, `format`. Also passed to `projectRaster`, with likely important arguments such as `method = "bilinear"`. See details.
	
	... passed to::
	`cropInputs`**:** `crop`
	`projectInputs` `projectRaster`
	`maskInputs` `fastMask` or `intersect`
	`fixErrors` `buffer`
	`writeOutputs` `writeRaster` or `shapefile`
	`determineFilename`
	* Can be overridden with `useSAcrs` ** Will mask with NAs from `rasterToMatch` if `maskWithRTM`

filename1	Character strings giving the file paths of the *input* object (`filename1`) `filename1` is only used for messaging (i.e., the object itself is passed in as `x`) and possibly naming of output (see details and `filename2`).
filename2	`filename2` is optional, and is either NULL (no writing of outputs to disk), or several options for writing the object to disk. If `TRUE` (the default), it will give it a file name determined by `.prefix(basename(filename1),prefix)`. If a character string, it will use this as its file name. See `determineFilename`.

studyArea	SpatialPolygons* object used for masking and possibly cropping if no rasterToMatch is provided. If not in same CRS, then it will be spTransformed to CRS of x before masking. Currently, this function will not reproject the x. Optional in postProcess.
rasterToMatch	Template Raster* object used for cropping (so extent should be the extent of desired outcome) and reprojecting (including changing the resolution and projection). See details in postProcess.
overwrite	Logical. Should downloading and all the other actions occur even if they pass the checksums or the files are all there.
useSAcrs	Logical. If FALSE, the default, then the desired projection will be taken from rasterToMatch or none at all. If TRUE, it will be taken from studyArea. See table in details below.
useCache	Passed to Cache in various places. Defaults to getOption("reproducible.useCache",2L) in prepInputs, and getOption("reproducible.useCache",FALSE) if calling any of the inner functions manually. For prepInputs, this mean it will use Cache only up to 2 nested levels, which will generally including postProcess and the first level of *Input functions, e.g., cropInputs, projectInputs, maskInputs, but not fixErrors.
verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal

Post processing sequence

If the rasterToMatch or studyArea are passed, then the following sequence will occur:

1. Fix errors fixErrors. Currently only errors fixed are for SpatialPolygons using buffer(...,width = 0).
2. Crop using cropInputs
3. Project using projectInputs
4. Mask using maskInputs
5. Determine file name determineFilename
6. Write that file name to disk, optionally writeOutputs

NOTE: checksumming does not occur during the post-processing stage, as there are no file downloads. To achieve fast results, wrap prepInputs with Cache

NOTE: sf objects are still very experimental.

Passing rasterToMatch and/or studyArea

Depending on which of these were passed, different things will happen to the targetFile located at filename1.

If targetFile is a Raster* object::

	rasterToMatch	studyArea	Both
extent	Yes	Yes	rasterToMatch
resolution	Yes	No	rasterToMatch
projection	Yes	No*	rasterToMatch*
alignment	Yes	No	rasterToMatch
mask	No**	Yes	studyArea**

* Can be overridden with useSAcrs. ** Will mask with NAs from rasterToMatch if maskWithRTM.

If targetFile **is a** Spatial* **object::**

	rasterToMatch	studyArea	Both
extent	Yes	Yes	rasterToMatch
resolution	NA	NA	NA
projection	Yes	No*	rasterToMatch*
alignment	NA	NA	NA
mask	No	Yes	studyArea

* Can be overridden with useSAcrs

See Also

prepInputs

Examples

```
# Add a study area to Crop and Mask to
# Create a "study area"
library(sp)
library(raster)
ow <- setwd(tempdir())

# make a SpatialPolygon
coords1 <- structure(c(-123.98, -117.1, -80.2, -100, -123.98, 60.9, 67.73, 65.58, 51.79, 60.9),
                     .Dim = c(5L, 2L))
Sr1 <- Polygon(coords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
crs(shpEcozone) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
coords <- structure(c(-118.98, -116.1, -99.2, -106, -118.98, 59.9, 65.73, 63.58, 54.79, 59.9),
                    .Dim = c(5L, 2L))
Sr1 <- Polygon(coords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

##########
```

```
shpEcozonePostProcessed <- postProcess(shpEcozone, studyArea = StudyArea)

# Try manually, individual pieces
shpEcozoneReprojected <- projectInputs(shpEcozone, StudyArea)
shpEcozoneCropped <- cropInputs(shpEcozone, StudyArea)
shpEcozoneClean <- fixErrors(shpEcozone)
shpEcozoneMasked <- maskInputs(shpEcozone, StudyArea)

setwd(ow)
# Add a study area to Crop and Mask to
# Create a "study area"
library(sp)
library(raster)
ow <- setwd(tempdir())

# make a SpatialPolygon
coords1 <- structure(c(-123.98, -117.1, -80.2, -100, -123.98, 60.9, 67.73, 65.58, 51.79, 60.9),
                     .Dim = c(5L, 2L))
Sr1 <- Polygon(coords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
crs(shpEcozone) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
coords <- structure(c(-118.98, -116.1, -99.2, -106, -118.98, 59.9, 65.73, 63.58, 54.79, 59.9),
                    .Dim = c(5L, 2L))
Sr1 <- Polygon(coords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

##########
shpEcozonePostProcessed <- postProcess(shpEcozone, studyArea = StudyArea)

# Try manually, individual pieces
shpEcozoneReprojected <- projectInputs(shpEcozone, StudyArea)
shpEcozoneCropped <- cropInputs(shpEcozone, StudyArea)
shpEcozoneClean <- fixErrors(shpEcozone)
shpEcozoneMasked <- maskInputs(shpEcozone, StudyArea)

setwd(ow)
```

`prepInputs` *Download and optionally post-process files*

Description

Usage

```
prepInputs(
  targetFile = NULL,
  url = NULL,
  archive = NULL,
  alsoExtract = NULL,
  destinationPath = getOption("reproducible.destinationPath", "."),
  fun = NULL,
  quick = getOption("reproducible.quick"),
  overwrite = getOption("reproducible.overwrite", FALSE),
  purge = FALSE,
  useCache = getOption("reproducible.useCache", 2),
  .tempPath,
  verbose = getOption("reproducible.verbose", 1),
  ...
)
```

Arguments

`targetFile`	Character string giving the path to the eventual file (raster, shapefile, csv, etc.) after downloading and extracting from a zip or tar archive. This is the file *before* it is passed to `postProcess`. Currently, the internal checksumming does not checksum the file after it is `postProcessed` (e.g., cropped/reprojected/masked). Using `Cache` around `prepInputs` will do a sufficient job in these cases. See table in `preProcess`.
`url`	Optional character string indicating the URL to download from. If not specified, then no download will be attempted. If not entry exists in the `CHECKSUMS.txt` (in `destinationPath`), an entry will be created or appended to. This `CHECKSUMS.txt` entry will be used in subsequent calls to `prepInputs` or `preProcess`, comparing the file on hand with the ad hoc `CHECKSUMS.txt`. See table in `preProcess`.
`archive`	Optional character string giving the path of an archive containing `targetFile`, or a vector giving a set of nested archives (e.g., `c("xxx.tar","inner.zip","inner.rar")`). If there is/are (an) inner archive(s), but they are unknown, the function will try all until it finds the `targetFile`. See table in `preProcess`. If it is `NA`, then it will *not* attempt to see it as an archive, even if it has archive-like file extension (e.g., `.zip`). This may be useful when an R function is expecting an archive directly.
`alsoExtract`	Optional character string naming files other than `targetFile` that must be extracted from the `archive`. If `NULL`, the default, then it will extract all files. Other options: `"similar"` will extract all files with the same filename without file extension as `targetFile`. `NA` will extract nothing other than `targetFile`. A character string of specific file names will cause only those to be extracted. See table in `preProcess`.
`destinationPath`	Character string of a directory in which to download and save the file that comes from `url` and is also where the function will look for `archive` or `targetFile`. NOTE (still experimental): To prevent repeated downloads in different locations, the user can also set `options("reproducible.inputPaths")` to one or

more local file paths to search for the file before attempting to download. Default for that option is NULL meaning do not search locally.

fun — Function, character string, or quoted call with which to load the targetFile into an R object. It must be either a function as a character string, or the function itself. If a character string or function, is should have the package name e.g., "raster::raster" or as an actual function, e.g., base::readRDS. If it is to be a custom function call, then use 'quote', e.g., 'quote(customFun(x = targetFilePath))', using 'targetFilePath' as the file path of the object that has been 'preProcess'ed. If the custom function is not in a package, 'prepInputs' may not find it. In such cases, simply pass the function as a named argument (with same name as function) e.g., 'prepInputs(..., fun = quote(customFun(x = targetFilePath), customFun = customFun)'. NOTE: passing NA will skip loading object into R. Note this will essentially replicate the functionality of simply calling preProcess directly.

quick — Logical. This is passed internally to Checksums (the quickCheck argument), and to Cache (the quick argument). This results in faster, though less robust checking of inputs. See the respective functions.

overwrite — Logical. Should downloading and all the other actions occur even if they pass the checksums or the files are all there.

purge — Logical or Integer. 0/FALSE (default) keeps existing CHECKSUMS.txt file and prepInputs will write or append to it. 1/TRUE will deleted the entire CHECKSUMS.txt file. Other options, see details.

useCache — Passed to Cache in various places. Defaults to getOption("reproducible.useCache",2L) in prepInputs, and getOption("reproducible.useCache",FALSE) if calling any of the inner functions manually. For prepInputs, this mean it will use Cache only up to 2 nested levels, which will generally including postProcess and the first level of *Input functions, e.g., cropInputs, projectInputs, maskInputs, but not fixErrors.

.tempPath — Optional temporary path for internal file intermediate steps. Will be cleared on.exit from this function.

verbose — Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal

... — Additional arguments passed to fun (i.e,. user supplied), postProcess and Cache. Since ... is passed to postProcess, these will ... will also be passed into the inner functions, e.g., cropInputs. Possibly useful other arguments include dlFun which is passed to preProcess. See details and examples.

Details

This function can be used to prepare R objects from remote or local data sources. The object of this function is to provide a reproducible version of a series of commonly used steps for getting, loading, and processing data. This function has two stages: Getting data (download, extracting from archives, loading into R) and post-processing (for Spatial* and Raster* objects, this is crop,

reproject, mask/intersect). To trigger the first stage, provide `url` or `archive`. To trigger the second stage, provide `studyArea` or `rasterToMatch`. See examples.

Stage 1 - Getting data

See `preProcess` for combinations of arguments.

1. Download from the web via either `drive_download`, `download.file`;
2. Extract from archive using `unzip` or `untar`;
3. Load into R using `raster`, `shapefile`, or any other function passed in with `fun`;
4. Checksumming of all files during this process. This is put into a '`CHECKSUMS.txt`' file in the `destinationPath`, appending if it is already there, overwriting the entries for same files if entries already exist.

Stage 2 - Post processing

This will be triggered if either `rasterToMatch` or `studyArea` is supplied.

1. Fix errors. Currently only errors fixed are for `SpatialPolygons` using `buffer(...,width = 0)`;
2. Crop using `cropInputs`;
3. Project using `projectInputs`;
4. Mask using `maskInputs`;
5. Determine file name `determineFilename` via `filename2`;
6. Optionally, write that file name to disk via `writeOutputs`.

NOTE: checksumming does not occur during the post-processing stage, as there are no file downloads. To achieve fast results, wrap `prepInputs` with `Cache`.

NOTE: `sf` objects are still very experimental.

postProcessing of `Raster*` and `Spatial*` objects::

If `rasterToMatch` or `studyArea` are used, then this will trigger several subsequent functions, specifically the sequence, *Crop, reproject, mask*, which appears to be a common sequence in spatial simulation. See `postProcess.spatialClasses`.

Understanding various combinations of `rasterToMatch` *and/or* `studyArea`*:* Please see `postProcess.spatialClasses`.

`purge`

In options for control of purging the `CHECKSUMS.txt` file are:

`0` keep file

`1` delete file

`2` delete entry for `targetFile`

`4` delete entry for `alsoExtract`

`3` delete entry for `archive`

`5` delete entry for `targetFile` & `alsoExtract`

6 delete entry for `targetFile`, `alsoExtract` & `archive`

7 delete entry that is failing (i.e., for the file downloaded by the `url`)

will only remove entries in the `CHECKSUMS.txt` that are associated with `targetFile`, `alsoExtract` or `archive` When `prepInputs` is called, it will write or append to a (if already exists) `CHECKSUMS.txt` file. If the `CHECKSUMS.txt` is not correct, use this argument to remove it.

Note

This function is still experimental: use with caution.

Author(s)

Eliot McIntire, Jean Marchal, and Tati Micheletti

See Also

`downloadFile`, `extractFromArchive`, `postProcess`.

Examples

```
# This function works within a module; however, currently,
#   \cde{sourceURL} is not yet working as desired. Use \code{url}.
## Not run:
# download a zip file from internet, unzip all files, load as shapefile, Cache the call
# First time: don't know all files - prepInputs will guess, if download file is an archive,
#   then extract all files, then if there is a .shp, it will load with raster::shapefile
dPath <- file.path(tempdir(), "ecozones")
shpEcozone <- prepInputs(destinationPath = dPath,
                 url = "http://sis.agr.gc.ca/cansis/nsdb/ecostrat/zone/ecozone_shp.zip")

# Robust to partial file deletions:
unlink(dir(dPath, full.names = TRUE)[1:3])
shpEcozone <- prepInputs(destinationPath = dPath,
                 url = "http://sis.agr.gc.ca/cansis/nsdb/ecostrat/zone/ecozone_shp.zip")
unlink(dPath, recursive = TRUE)

# Once this is done, can be more precise in operational code:
#  specify targetFile, alsoExtract, and fun, wrap with Cache
ecozoneFilename <- file.path(dPath, "ecozones.shp")
ecozoneFiles <- c("ecozones.dbf", "ecozones.prj",
                  "ecozones.sbn", "ecozones.sbx", "ecozones.shp", "ecozones.shx")
shpEcozone <- prepInputs(targetFile = ecozoneFilename,
                 url = "http://sis.agr.gc.ca/cansis/nsdb/ecostrat/zone/ecozone_shp.zip",
                         alsoExtract = ecozoneFiles,
                         fun = "shapefile", destinationPath = dPath)
unlink(dPath, recursive = TRUE)

#' # Add a study area to Crop and Mask to
# Create a "study area"
library(sp)
library(raster)
```

```
coords <- structure(c(-122.98, -116.1, -99.2, -106, -122.98, 59.9, 65.73, 63.58, 54.79, 59.9),
                    .Dim = c(5L, 2L))
Sr1 <- Polygon(coords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

#  specify targetFile, alsoExtract, and fun, wrap with Cache
ecozoneFilename <- file.path(dPath, "ecozones.shp")
# Note, you don't need to "alsoExtract" the archive... if the archive is not there, but the
#   targetFile is there, it will not redownload the archive.
ecozoneFiles <- c("ecozones.dbf", "ecozones.prj",
                  "ecozones.sbn", "ecozones.sbx", "ecozones.shp", "ecozones.shx")
shpEcozoneSm <- Cache(prepInputs,
                  url = "http://sis.agr.gc.ca/cansis/nsdb/ecostrat/zone/ecozone_shp.zip",
                       targetFile = reproducible::asPath(ecozoneFilename),
                       alsoExtract = reproducible::asPath(ecozoneFiles),
                       studyArea = StudyArea,
                       fun = "shapefile", destinationPath = dPath,
                       filename2 = "EcozoneFile.shp") # passed to determineFilename

plot(shpEcozone)
plot(shpEcozoneSm, add = TRUE, col = "red")
unlink(dPath)

# Big Raster, with crop and mask to Study Area - no reprojecting (lossy) of raster,
#   but the StudyArea does get reprojected, need to use rasterToMatch
dPath <- file.path(tempdir(), "LCC")
lcc2005Filename <- file.path(dPath, "LCC2005_V1_4a.tif")
url <- file.path("ftp://ftp.ccrs.nrcan.gc.ca/ad/NLCCLandCover",
                 "LandcoverCanada2005_250m/LandCoverOfCanada2005_V1_4.zip")

# messages received below may help for filling in more arguments in the subsequent call
LCC2005 <- prepInputs(url = url,
                      destinationPath = asPath(dPath),
                      studyArea = StudyArea)

plot(LCC2005)

# if wrapped with Cache, will be very fast second time (via memoised copy)
LCC2005 <- Cache(prepInputs, url = url,
                 targetFile = lcc2005Filename,
                 archive = asPath("LandCoverOfCanada2005_V1_4.zip"),
                 destinationPath = asPath(dPath),
                 studyArea = StudyArea)
# Using dlFun -- a custom download function -- passed to preProcess
test1 <- prepInputs(targetFile = "GADM_2.8_LUX_adm0.rds", # must specify currently
                    dlFun = "raster::getData", name = "GADM", country = "LUX", level = 0,
                     path = dPath)

## End(Not run)
```

preProcessParams *Download, Checksum, Extract files*

Description

This does downloading (via `downloadFile`), checksumming (`Checksums`), and extracting from archives (`extractFromArchive`), plus cleaning up of input arguments (e.g., paths, function names). This is the first stage of three used in `prepInputs`.

Usage

```
preProcessParams(n = NULL)

preProcess(
  targetFile = NULL,
  url = NULL,
  archive = NULL,
  alsoExtract = NULL,
  destinationPath = getOption("reproducible.destinationPath", "."),
  fun = NULL,
  dlFun = NULL,
  quick = getOption("reproducible.quick"),
  overwrite = getOption("reproducible.overwrite", FALSE),
  purge = FALSE,
  verbose = getOption("reproducible.verbose", 1),
  .tempPath,
  ...
)
```

Arguments

`n`	Number of non-null arguments passed to `preProcess`. E.g., passing `n = 1` returns combinations with only a single non-NULL parameter. If `NULL` (default), all parameter combinations are returned.
`targetFile`	Character string giving the path to the eventual file (raster, shapefile, csv, etc.) after downloading and extracting from a zip or tar archive. This is the file *before* it is passed to `postProcess`. Currently, the internal checksumming does not checksum the file after it is `postProcessed` (e.g., cropped/reprojected/masked). Using `Cache` around `prepInputs` will do a sufficient job in these cases. See table in `preProcess`.
`url`	Optional character string indicating the URL to download from. If not specified, then no download will be attempted. If not entry exists in the `CHECKSUMS.txt` (in `destinationPath`), an entry will be created or appended to. This `CHECKSUMS.txt` entry will be used in subsequent calls to `prepInputs` or `preProcess`, comparing the file on hand with the ad hoc `CHECKSUMS.txt`. See table in `preProcess`.

archive — Optional character string giving the path of an archive containing targetFile, or a vector giving a set of nested archives (e.g., c("xxx.tar","inner.zip","inner.rar")). If there is/are (an) inner archive(s), but they are unknown, the function will try all until it finds the targetFile. See table in preProcess. If it is NA, then it will *not* attempt to see it as an archive, even if it has archive-like file extension (e.g., .zip). This may be useful when an R function is expecting an archive directly.

alsoExtract — Optional character string naming files other than targetFile that must be extracted from the archive. If NULL, the default, then it will extract all files. Other options: "similar" will extract all files with the same filename without file extension as targetFile. NA will extract nothing other than targetFile. A character string of specific file names will cause only those to be extracted. See table in preProcess.

destinationPath — Character string of a directory in which to download and save the file that comes from url and is also where the function will look for archive or targetFile. NOTE (still experimental): To prevent repeated downloads in different locations, the user can also set options("reproducible.inputPaths") to one or more local file paths to search for the file before attempting to download. Default for that option is NULL meaning do not search locally.

fun — Function, character string, or quoted call with which to load the targetFile into an R object. It must be either a function as a character string, or the function itself. If a character string or function, is should have the package name e.g., "raster::raster" or as an actual function, e.g., base::readRDS. If it is to be a custom function call, then use 'quote', e.g., 'quote(customFun(x = targetFilePath))', using 'targetFilePath' as the file path of the object that has been 'preProcess'ed. If the custom function is not in a package, 'prepInputs' may not find it. In such cases, simply pass the function as a named argument (with same name as function) e.g., 'prepInputs(..., fun = quote(customFun(x = targetFilePath), customFun = customFun)'. NOTE: passing NA will skip loading object into R. Note this will essentially replicate the functionality of simply calling preProcess directly.

dlFun — Optional "download function" name, such as "raster::getData", which does custom downloading, in addition to loading into R. Still experimental.

quick — Logical. This is passed internally to Checksums (the quickCheck argument), and to Cache (the quick argument). This results in faster, though less robust checking of inputs. See the respective functions.

overwrite — Logical. Should downloading and all the other actions occur even if they pass the checksums or the files are all there.

purge — Logical or Integer. 0/FALSE (default) keeps existing CHECKSUMS.txt file and prepInputs will write or append to it. 1/TRUE will deleted the entire CHECKSUMS.txt file. Other options, see details.

verbose — Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal

.tempPath	Optional temporary path for internal file intermediate steps. Will be cleared on.exit from this function.
...	Additional arguments passed to fun (i.e,. user supplied), postProcess and Cache. Since ... is passed to postProcess, these will ... will also be passed into the inner functions, e.g., cropInputs. Possibly useful other arguments include dlFun which is passed to preProcess. See details and examples.

Value

A list with 5 elements: checkSums (the result of a Checksums after downloading), dots (cleaned up ..., including deprecated argument checks), fun (the function to be used to load the preProcessed object from disk), and targetFilePath (the fully qualified path to the targetFile).

Combinations of targetFile, url, archive, alsoExtract

Use preProcessParams() for a table describing various parameter combinations and their outcomes.

* If the url is a file on Google Drive, checksumming will work even without a targetFile specified because there is an initial attempt to get the remove file information (e.g., file name). With that, the connection between the url and the filename used in the 'CHECKSUMS.txt' file can be made.

Author(s)

Eliot McIntire

---

projectInputs *Project* Raster* *or Spatial** *or* sf *objects*

Description

A simple wrapper around the various different tools for these GIS types.

Usage

```
projectInputs(
  x,
  targetCRS,
  verbose = getOption("reproducible.verbose", 1),
  ...
)

## Default S3 method:
projectInputs(x, targetCRS, ...)

## S3 method for class 'Raster'
projectInputs(
  x,
```

```
  targetCRS = NULL,
  verbose = getOption("reproducible.verbose", 1),
  rasterToMatch = NULL,
  cores = NULL,
  useGDAL = getOption("reproducible.useGDAL", TRUE),
  ...
)

## S3 method for class 'sf'
projectInputs(
  x,
  targetCRS,
  verbose = getOption("reproducible.verbose", 1),
  ...
)

## S3 method for class 'Spatial'
projectInputs(
  x,
  targetCRS,
  verbose = getOption("reproducible.verbose", 1),
  ...
)
```

Arguments

x	A Raster*, Spatial* or sf object
targetCRS	The CRS of x at the end of this function (i.e., the goal)
verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal
...	Passed to projectRaster.
rasterToMatch	Template Raster* object passed to the to argument of projectRaster, thus will changing the resolution and projection of x. See details in postProcess.
cores	An integer* or 'AUTO'. This will be used if gdalwarp is triggered. 'AUTO'* will calculate 90 number of cores in the system, while an integer or rounded float will be passed as the exact number of cores to be used.
useGDAL	Logical or "force". Defaults to getOption("reproducible.useGDAL" = TRUE). If TRUE, then this function will use gdalwarp only when not small enough to fit in memory (i.e., *if the operation fails* the raster::canProcessInMemory(x,3) test). Using gdalwarp will usually be faster than raster::projectRaster, the function used if this is FALSE. Since since the two options use different algorithms, there may be different projection results. "force" will cause it to use GDAL regardless of the memory test described here.

Value

A file of the same type as starting, but with projection (and possibly other characteristics, including resolution, origin, extent if changed).

Examples

```
# Add a study area to Crop and Mask to
# Create a "study area"
library(sp)
library(raster)
ow <- setwd(tempdir())

# make a SpatialPolygon
coords1 <- structure(c(-123.98, -117.1, -80.2, -100, -123.98, 60.9, 67.73, 65.58, 51.79, 60.9),
                     .Dim = c(5L, 2L))
Sr1 <- Polygon(coords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
crs(shpEcozone) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
coords <- structure(c(-118.98, -116.1, -99.2, -106, -118.98, 59.9, 65.73, 63.58, 54.79, 59.9),
                    .Dim = c(5L, 2L))
Sr1 <- Polygon(coords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

##########
shpEcozonePostProcessed <- postProcess(shpEcozone, studyArea = StudyArea)

# Try manually, individual pieces
shpEcozoneReprojected <- projectInputs(shpEcozone, StudyArea)
shpEcozoneCropped <- cropInputs(shpEcozone, StudyArea)
shpEcozoneClean <- fixErrors(shpEcozone)
shpEcozoneMasked <- maskInputs(shpEcozone, StudyArea)

setwd(ow)
```

---

reproducibleOptions reproducible *options*

Description

These provide top-level, powerful settings for a comprehensive reproducible workflow. To see defaults, run `reproducibleOptions()`. See Details below.

Usage

```
reproducibleOptions()
```

Details

Below are options that can be set with `options("reproducible.xxx" = newValue)`, where `xxx` is one of the values below, and `newValue` is a new value to give the option. Sometimes these options can be placed in the user's `.Rprofile` file so they persist between sessions.

The following options are likely of interest to most users:

`ask` Default: `TRUE`. Used in `clearCache` and `keepCache`.

`cachePath` Default: `.reproducibleTempCacheDir`. Used in `Cache` and many others. The default path for repositories if not passed as an argument.

`cacheSaveFormat` Default: `"rds"`. What save format to use; currently, `"qs"` or `"rds"`.

`cacheSpeed` Default `"slow"`. One of `"slow"` or `"fast"` (1 or 2). `"slow"` uses `digest::digest` internally, which is transferable across operating systems, but much slower than `fastdigest::fastdigest`. So, if all caching is happening on a single machine, `"fast"` would be a good setting.

`conn` Default: `NULL`. Sets a specific connection to a database, e.g., `dbConnect(drv = RSQLite::SQLite())` or `dbConnect(drv = RPostgres::Postgres()`. For remote database servers, setting one connection may be far faster than using `drv` which must make a new connection every time.

`destinationPath` Default: `NULL`. Used in `prepInputs` and `preProcess`. Can be set globally here.

`drv` Default: `RSQLite::SQLite()`. Sets the default driver for the backend database system. Only tested with `RSQLite::SQLite()` and `RPostgres::Postgres()`.

`futurePlan` Default: `FALSE`. On Linux OSes, `Cache` and `cloudCache` have some functionality that uses the `future` package. Default is to not use these, as they are experimental. They may, however, be very effective in speeding up some things, specifically, uploading cached elements via googledrive in `cloudCache`.

`inputPaths` Default: `NULL`. Used in `prepInputs` and `preProcess`. If set to a path, this will cause these functions to save their downloaded and preprocessed file to this location, with a hardlink (via `file.link`) to the file created in the `destinationPath`. This can be used so that individual projects that use common data sets can maintain modularity (by placing downloaded objects in their `destinationPath`, but also minimize re-downloading the same (perhaps large) file over and over for each project. Because the files are hardlinks, there is no extra space taken up by the apparently duplicated files.

`inputPathsRecursive` Default: `FALSE`. Used in `prepInputs` and `preProcess`. Should the `reproducible.inputPaths` be searched recursively for existence of a file?

`nThreads` Default: `1`. The number of threads to use for reading/writing cache files.

`overwrite` Default: `FALSE`. Used in `prepInputs`, `preProcess`, `downloadFile`, and `postProcess`.

`quick` Default: `FALSE`. Used in `Cache`. This will cause `Cache` to use `file.size(file)` instead of the `digest::digest(file)`. Less robust to changes, but faster. *NOTE: this will only affect objects on disk.*

`shapefileRead` Default `NULL`. Used during `prepInputs` when reading a `.shp` file. If `NULL`, it will use 'sf::st_read' if 'sf' package is available; otherwise, it will use 'raster::shapefile'

`showSimilar` Default `FALSE`. Passed to `Cache`.

`useCache` Default: `TRUE`. Used in `Cache`. If `FALSE`, then the entire `Cache` machinery is skipped and the functions are run as if there was no Cache occurring. Can also take 2 other values: `'overwrite'` and `'devMode'`. `'overwrite'` will cause no recovery of objects from the cache

repository, only new ones will be created. If the hash is identical to a previous one, then this will overwrite the previous one. `'devMode'` will function as normally `Cache` except it will use the `userTags` to determine if a previous function has been run. If the `userTags` are identical, but the digest value is different, the old value will be deleted from the cache repository and this new value will be added. This addresses a common situation during the development stage: functions are changing frequently, so any entry in the cache repository will be stale following changes to functions, i.e., they will likely never be relevant again. This will therefore keep the cache repository clean of stale objects. If there is ambiguity in the `userTags`, i.e., they do not uniquely identify a single entry in the `cacheRepo`, then this option will default back to the non-dev-mode behaviour to avoid deleting objects. This, therefore, is most useful if the user is using unique values for `userTags`.

`useCloud` Default `FALSE`. Passed to `Cache`.

`useDBI` Default: `TRUE`. As of version 0.3, the backend is now **DBI** instead of **archivist**.

`useGDAL` Default `TRUE`. Passed to `useGDAL` in `projectInputs.Raster`.

`useMemoise` Default: `FALSE`. Used in `Cache`. If `TRUE`, recovery of cached elements from the `cacheRepo` will use `memoise::memoise`. This means that the 2nd time running a function will be much faster than the first in a session (which either will create a new cache entry to disk or read a cached entry from disk). *NOTE: memoised values are removed when the R session is restarted.* **This option will use more RAM** and so may need to be turned off if RAM is limiting. `clearCache` of any sort will cause all memoising to be 'forgotten' (`memoise::forget`).

`useNewDigestAlgorithm` Default: `1`. Option 1 is the version that has existed for sometime. There is now and option `2` which is substantially faster. It will, however, create Caches that are not compatible with previous ones. Options `1` and `2` are not compatible with the earlier `0`. `1` and `2` will make `Cache` less sensitive to minor but irrelevant changes (like changing the order of arguments) and will work successfully across operating systems (especially relevant for the new `cloudCache` function.

`verbose` Default: `FALSE`. If set to `TRUE` then every `Cache` call will show a summary of the objects being cached, their `object.size` and the time it took to digest them and also the time it took to run the call and save the call to the cache repository or load the cached copy from the repository. This may help diagnosing some problems that may occur.

Advanced

The following options are likely not needed by a user.

`cloudChecksumsFilename` Default: `file.path(dirname(.reproducibleTempCacheDir()),"checksums.rds")`. Used in `cloudCache`

`length` Default: `Inf`. Used in `Cache`, specifically to the internal calls to `CacheDigest`. This is passed to `digest::digest`. Mostly this would be changed from default `Inf` if the digesting is taking too long. Use this with caution, as some objects will have *many* `NA` values in their first *many* elements

`useragent` Default: `"https://github.com/PredictiveEcology/reproducible"`. User agent for downloads using this package.

`retry` *A wrapper around* `try` *that retries on failure*

Description

This is useful for functions that are "flaky", such as `curl`, which may fail for unknown reasons that do not persist.

Usage

```
retry(
  expr,
  envir = parent.frame(),
  retries = 5,
  exponentialDecayBase = 1.3,
  silent = TRUE,
  exprBetween = NULL
)
```

Arguments

`expr`	Quoted expression to run, i.e., `quote(...)`
`envir`	The environment in which to evaluate the quoted expression, default to `parent.frame(1)`
`retries`	Numeric. The maximum number of retries.
`exponentialDecayBase`	Numeric > 1.0. The delay between successive retries will be `runif(1,min = 0,max = exponentialDecayBase ^ i -1)` where `i` is the retry number (i.e., follows `seq_len(retries)`)
`silent`	Logical indicating whether to `try` silently.
`exprBetween`	Another expression that should be run after a failed attempt of the ‘expr‘. It must include an assignment operator, specifying what object (that is used in ‘expr‘) will be updated prior to running the ‘expr‘ again.

Details

Based on https://github.com/jennybc/googlesheets/issues/219#issuecomment-195218525.

searchFull *Search up the full scope for functions*

Description

This is like base::search but when used inside a function, it will show the full scope (see figure in the section *Binding environments* on http://adv-r.had.co.nz/Environments.html). This full search path will be potentially much longer than just search() (which always starts at .GlobalEnv).

searchFullEx shows an example function that is inside this package whose only function is to show the Scope of a package function.

Usage

```
searchFull(env = parent.frame(), simplify = TRUE)

searchFullEx()
```

Arguments

env	The environment to start searching at. Default is calling environment, i.e., parent.frame()
simplify	Logical. Should the output be simplified to character, if possible (usually it is not possible because environments don't always coerce correctly)

Details

searchFullEx can be used to show an example of the use of searchFull.

Value

A list of environments that is the actual search path, unlike search() which only prints from .GlobalEnv up to base through user attached packages.

See Also

search

Examples

```
seeScope <- function() {
  searchFull()
}
seeScope()
searchFull()
searchFullEx()
```

spatialClasses-class *The* spatialClasses *class*

Description

This class is the union of several spatial objects from **raster** and **sp** packages.

Details

Members:

- RasterLayer, RasterLayerSparse, RasterStack;
- SpatialLines, SpatialLinesDataFrame;
- SpatialPixels, SpatialPixelsDataFrame;
- SpatialPoints, SpatialPointsDataFrame;
- SpatialPolygons, SpatialPolygonsDataFrame.

Notably missing is RasterBrick, for now.

Author(s)

Eliot McIntire

studyAreaName *Get a unique name for a given study area*

Description

Digest a spatial object to get a unique character string (hash) of the study area. Use .suffix() to append the hash to a filename, e.g., when using filename2 in prepInputs.

Usage

```
studyAreaName(studyArea, ...)

## S4 method for signature 'SpatialPolygonsDataFrame'
studyAreaName(studyArea, ...)

## S4 method for signature 'ANY'
studyAreaName(studyArea, ...)
```

Arguments

studyArea	Spatial object.
...	Other arguments (not currently used)

unrarPath *The known path for unrar or 7z*

Description

The known path for unrar or 7z

Usage

```
.unrarPath
```

Format

An object of class NULL of length 0.

writeFuture *Write to cache repository, using* future::future

Description

This will be used internally if options("reproducible.futurePlan" = TRUE). This is still experimental.

Usage

```
writeFuture(
  written,
  outputToSave,
  cacheRepo,
  userTags,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  cacheId,
  linkToCacheId = NULL
)
```

Arguments

written	Integer. If zero or positive then it needs to be written still. Should be 0 to start.
outputToSave	The R object to save to repository
cacheRepo	The file path of the repository
userTags	Character string of tags to attach to this outputToSave in the CacheRepo
drv	an object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).

conn	A DBIConnection object, as returned by dbConnect().
cacheId	Character string. If passed, this will override the calculated hash of the inputs, and return the result from this cacheId in the cacheRepo. Setting this is equivalent to manually saving the output of this function, i.e., the object will be on disk, and will be recovered in subsequent This may help in some particularly finicky situations where Cache is not correctly detecting unchanged inputs. This will guarantee the object will be identical each time; this may be useful in operational code.
linkToCacheId	Optional. If a cacheId is provided here, then a file.link will be made to the file with that cacheId name in the cache repo. This is used when identical outputs exist in the cache. This will save disk space.

---

writeOutputs *Write module inputs on disk*

Description

Can be used to write prepared inputs on disk.

Usage

```
writeOutputs(
  x,
  filename2,
  overwrite = getOption("reproducible.overwrite", NULL),
  ...
)

## S3 method for class 'Raster'
writeOutputs(
  x,
  filename2 = NULL,
  overwrite = getOption("reproducible.overwrite", FALSE),
  verbose = getOption("reproducible.verbose", 1),
  ...
)

## S3 method for class 'Spatial'
writeOutputs(
  x,
  filename2 = NULL,
  overwrite = getOption("reproducible.overwrite", TRUE),
  ...
)

## S3 method for class 'sf'
```

```
writeOutputs(
  x,
  filename2 = NULL,
  overwrite = getOption("reproducible.overwrite", FALSE),
  verbose = getOption("reproducible.verbose", 1),
  ...
)

## S3 method for class 'quosure'
writeOutputs(x, filename2, ...)

## Default S3 method:
writeOutputs(x, filename2, ...)
```

Arguments

x	The object save to disk i.e., write outputs
filename2	File name passed to writeRaster, or shapefile or st_write (dsn argument).
overwrite	Logical. Should file being written overwrite an existing file if it exists.
...	Passed into shapefile or writeRaster or st_write
verbose	Numeric, -1 silent (where possible), 0 being very quiet, 1 showing more messaging, 2 being more messaging, etc. Default is 1. Above 3 will output much more information about the internals of Caching, which may help diagnose Caching challenges. Can set globally with an option, e.g., options('reproducible.verbose' = 0) to reduce to minimal

Author(s)

Eliot McIntire and Jean Marchal

Examples

```
# Add a study area to Crop and Mask to
# Create a "study area"
library(sp)
library(raster)
ow <- setwd(tempdir())

# make a SpatialPolygon
coords1 <- structure(c(-123.98, -117.1, -80.2, -100, -123.98, 60.9, 67.73, 65.58, 51.79, 60.9),
                     .Dim = c(5L, 2L))
Sr1 <- Polygon(coords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
crs(shpEcozone) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
coords <- structure(c(-118.98, -116.1, -99.2, -106, -118.98, 59.9, 65.73, 63.58, 54.79, 59.9),
                    .Dim = c(5L, 2L))
```

```
Sr1 <- Polygon(coords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

##########
shpEcozonePostProcessed <- postProcess(shpEcozone, studyArea = StudyArea)

# Try manually, individual pieces
shpEcozoneReprojected <- projectInputs(shpEcozone, StudyArea)
shpEcozoneCropped <- cropInputs(shpEcozone, StudyArea)
shpEcozoneClean <- fixErrors(shpEcozone)
shpEcozoneMasked <- maskInputs(shpEcozone, StudyArea)

setwd(ow)
```

Index