Package ‘reproducible’

November 19, 2019

Type Package

Title A Set of Tools that Enhance Reproducibility Beyond Package Management

Description Collection of high-level, robust, machine- and OS-independent tools for making deeply reproducible and reusable content in R. This includes light weight package management (similar to 'packrat' and 'checkpoint', but more flexible, lightweight, simpler yet less tested than both), tools for caching, downloading and verifying or writing checksums, post-processing of common spatial datasets, and accessing GitHub repositories. Some features are still under active development.

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

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

Date 2019-11-17

Version 0.2.11

Depends R (>= 3.5)

Imports archivist (>= 2.1.2), backports, crayon, data.table (>= 1.10.4), digest, dplyr, fastdigest, fasterize, fpCompare, gdalUtils, git2r (>= 0.18), googledrive, httr, magrittr, memoise, methods, parallel, quickPlot, R.utils, raster, Rcurl (>= 1.95-4.8), remotes, rgdal, rgeos, rlang, sf, sp, testthat, tools, utils, versions

Suggests covr, future, knitr, rmarkdown, TimeWarp

Encoding UTF-8

Language en-CA

License GPL-3

VignetteBuilder knitr, rmarkdown

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

ByteCompile yes
RoxygenNote 7.0.0

Collate 'cache-helpers.R' 'cache-internals.R' 'cache-tools.R'
   'robustDigest.R' 'cache.R' 'checksums.R' 'cloud.R' 'cloudOld.R'
   'consistentPaths.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' 'reproducible-package.R' 'search.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 2019-11-19 10:00:02 UTC

R topics documented:

reproducible-package .............................................. 4
   .addChangedAttr .............................................. 5
   .addTagsToOutput ............................................. 6
   .cacheMessage ................................................ 6
   .checkCacheRepo .............................................. 7
   .debugCache .................................................... 8
   .installPackages ............................................. 9
   .objSizeInclEnviros .......................................... 10
   .preDigestByClass ............................................ 11
   .prefix ......................................................... 11
   .prepareFileBackedRaster .................................. 12
   .prepareOutput ............................................... 13
   .removeCacheAtts ............................................ 14
   .setSubAttrInList ............................................ 15
   .sortDotsUnderscoreFirst ................................... 15
   .tagsByClass ................................................. 16
   assessDataType ............................................... 17
   assessDataTypeGDAL ........................................... 19
   basename2 ...................................................... 21
   Cache ......................................................... 22
   CacheDigest ................................................... 29
   checkAndMakeCloudFolderID ................................ 30
   checkGDALVersion ............................................. 30
   checkoutVersion ............................................... 31
   checkoutPath .................................................. 32
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checksums</td>
<td>33</td>
</tr>
<tr>
<td>clearCache</td>
<td>35</td>
</tr>
<tr>
<td>clearStubArtifacts</td>
<td>39</td>
</tr>
<tr>
<td>cloudCache</td>
<td>40</td>
</tr>
<tr>
<td>cloudCheckOld</td>
<td>40</td>
</tr>
<tr>
<td>cloudDownload</td>
<td>41</td>
</tr>
<tr>
<td>cloudSyncCacheOld</td>
<td>41</td>
</tr>
<tr>
<td>cloudUpload</td>
<td>42</td>
</tr>
<tr>
<td>cloudWriteOld</td>
<td>43</td>
</tr>
<tr>
<td>compareNA</td>
<td>44</td>
</tr>
<tr>
<td>convertPaths</td>
<td>44</td>
</tr>
<tr>
<td>Copy</td>
<td>45</td>
</tr>
<tr>
<td>copySingleFile</td>
<td>47</td>
</tr>
<tr>
<td>cropInputs</td>
<td>48</td>
</tr>
<tr>
<td>determineFilename</td>
<td>50</td>
</tr>
<tr>
<td>downloadFile</td>
<td>52</td>
</tr>
<tr>
<td>extractFromArchive</td>
<td>54</td>
</tr>
<tr>
<td>fastMask</td>
<td>55</td>
</tr>
<tr>
<td>Filenames</td>
<td>56</td>
</tr>
<tr>
<td>getGDALVersion</td>
<td>57</td>
</tr>
<tr>
<td>getUserTags</td>
<td>58</td>
</tr>
<tr>
<td>installedVersions</td>
<td>58</td>
</tr>
<tr>
<td>installVersions</td>
<td>59</td>
</tr>
<tr>
<td>linkOrCopy</td>
<td>60</td>
</tr>
<tr>
<td>makeMemoisable</td>
<td>62</td>
</tr>
<tr>
<td>maskInputs</td>
<td>62</td>
</tr>
<tr>
<td>mergeCache</td>
<td>64</td>
</tr>
<tr>
<td>newLibPaths</td>
<td>65</td>
</tr>
<tr>
<td>normPath</td>
<td>65</td>
</tr>
<tr>
<td>objSize</td>
<td>66</td>
</tr>
<tr>
<td>package_dependenciesMem</td>
<td>69</td>
</tr>
<tr>
<td>Path-class</td>
<td>70</td>
</tr>
<tr>
<td>pipe</td>
<td>71</td>
</tr>
<tr>
<td>pkgDep</td>
<td>73</td>
</tr>
<tr>
<td>pkgSnapshot</td>
<td>74</td>
</tr>
<tr>
<td>postProcess</td>
<td>75</td>
</tr>
<tr>
<td>prepInputs</td>
<td>79</td>
</tr>
<tr>
<td>preProcess</td>
<td>84</td>
</tr>
<tr>
<td>projectInputs</td>
<td>86</td>
</tr>
<tr>
<td>reproducibleOptions</td>
<td>88</td>
</tr>
<tr>
<td>Require</td>
<td>89</td>
</tr>
<tr>
<td>retry</td>
<td>91</td>
</tr>
<tr>
<td>searchFull</td>
<td>92</td>
</tr>
<tr>
<td>tempdir2</td>
<td>93</td>
</tr>
<tr>
<td>unrarPath</td>
<td>93</td>
</tr>
<tr>
<td>writeFuture</td>
<td>94</td>
</tr>
<tr>
<td>writeOutputs</td>
<td>94</td>
</tr>
</tbody>
</table>

Index 97
Description

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

Main Tools

There are many elements within the reproducible package. However, there are currently 3 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.

Function

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, 2nd time will compare the inputs to a an SQLite 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 A function to download, or load objects, and possible 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 run without waiting for the code itself to run. 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.

Require A version of require that incorporates elements of install.packages, devtools::install_github, packrat. It allows for users code to work for a new user on a new machine that may or may not have all packages installed.

Package options

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

Author(s)

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

Authors:

- Alex M Chubaty <alex.chubaty@gmail.com> (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 [copyright holder]

See Also

Useful links:

- https://reproducible.predictiveecology.org
.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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Any R object returned from a function</td>
</tr>
<tr>
<td>preDigest</td>
<td>The full, element by element hash of the input arguments to that same function, e.g., from .robustDigest</td>
</tr>
<tr>
<td>origArguments</td>
<td>These are the actual arguments (i.e., the values, not the names) that were the source for preDigest</td>
</tr>
<tr>
<td>...</td>
<td>Anything passed to methods.</td>
</tr>
</tbody>
</table>

Value

The object, modified

Author(s)

Eliot McIntire

Examples

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

Description

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

```r
.cacheMessage(
  object,
  functionName,
  fromMemoise = getOption("reproducible.useMemoise", TRUE)
)
```

## S4 method for signature 'ANY'

```r
.cacheMessage(
  object,
  functionName,
  fromMemoise = getOption("reproducible.useMemoise", TRUE)
)
```

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.

Value

Nothing; called for its messaging side effect.

Author(s)

Eliot McIntire

Examples

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

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

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

## S4 method for signature 'ANY'

```r
.checkCacheRepo(object, create = FALSE)
```
**Arguments**

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

**Value**

A character string with a path to a cache repository.

**Author(s)**

Eliot McIntire

**Examples**

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

---

**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
.installPackages

Internal function to install packages

Description

Internal function to install packages

Usage

```
..installPackages(
  packages,
  repos = getOption("repos"),
  githubPkgs = character(0),
  githubPkgNames,
  nonLibPathPkgs = character(0),
  install_githubArgs,
  install.packagesArgs = list(),
  libPath = .libPaths()[1],
  standAlone = standAlone,
  forget = FALSE
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packages</td>
<td>Character vector of packages to install via <code>install.packages</code>, then load (i.e., with <code>library</code>). If it is one package, it can be unquoted (as in <code>require</code>)</td>
</tr>
<tr>
<td>repos</td>
<td>The remote repository (e.g., a CRAN mirror), passed to <code>install.packages</code>,</td>
</tr>
<tr>
<td>githubPkgs</td>
<td>Character vector of github repositories and packages, in the form username/package@branch, with branch being optional.</td>
</tr>
<tr>
<td>githubPkgNames</td>
<td>Character vector of the package names, i.e., just the R package name.</td>
</tr>
<tr>
<td>nonLibPathPkgs</td>
<td>Character vector of all installed packages that are in .libPaths, but not in libPath. This would normally include a listing of base packages, but may also include other library paths if standAlone if FALSE</td>
</tr>
<tr>
<td>install_githubArgs</td>
<td>List of optional named arguments, passed to <code>install_github</code></td>
</tr>
<tr>
<td>install.packagesArgs</td>
<td>List of optional named arguments, passed to <code>install.packages</code></td>
</tr>
<tr>
<td>libPath</td>
<td>The library path where all packages should be installed, and looked for to load (i.e., call <code>library</code>)</td>
</tr>
<tr>
<td>standAlone</td>
<td>Logical. If TRUE, all packages will be installed and loaded strictly from the libPaths only. If FALSE, all .libPaths will be used to find the correct versions. This can be create dramatically faster installs if the user has a substantial number of the packages already in their personal library. In the case of TRUE, there will be a hidden file place in the libPath directory that lists all the packages that were needed during the <code>require</code> call. Default FALSE to minimize package installing.</td>
</tr>
</tbody>
</table>
forget

Internally, this function identifies package dependencies using a memoised function for speed on reuse. But, it may be inaccurate in some cases, if packages were installed manually by a user. Set this to **TRUE** to refresh that dependency calculation.

Examples

```r
## Not run:
.installPackages("crayon")

## End(Not run)
```

---

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

**Description**

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

**Usage**

```
.objSizeInclEnviros(object)
```

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

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

**Arguments**

- **object** Any R object.

**Value**

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

**Author(s)**

Eliot McIntire

**Examples**

```r
a <- new.env()
a$b <- 1:10
object.size(a)
.objectSizeInclEnviros(a)  # much larger
```
Any miscellaneous things to do before \texttt{.robustDigest} and after \texttt{FUN} call

Description
The default method for \texttt{preDigestByClass} and simply returns \texttt{NULL}. There may be methods in other packages.

Usage

\begin{verbatim}
.preDigestByClass(object)

## S4 method for signature 'ANY'
.preDigestByClass(object)
\end{verbatim}

Arguments

\begin{itemize}
  \item \textbf{object} \hspace{1cm} Any R object.
\end{itemize}

Value
A list with elements that will likely be used in \texttt{.postProcessing}

Author(s)
Eliot McIntire

Examples

\begin{verbatim}
a <- 1
.preDigestByClass(a) # returns NULL in the simple case here.
\end{verbatim}

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

\begin{verbatim}
.prefix(f, prefix ="")

.suffix(f, suffix ="")
\end{verbatim}
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

```r
# 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"
```

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

```r
.prepareFileBackedRaster(obj, repoDir = NULL, overwrite = FALSE, ...)
```

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.
- **...**: passed to archivist::saveToRepo

Value

A raster object and its newly located file backing. Note that if this is a legitimate archivist repository, the new location will be a subdirectory called ‘rasters/’ of ‘repoDir/’. If this is not a repository, the new location will be within repoDir.
Author(s)
Eliot McIntire

Examples

```r
library(raster)
archivist::createLocalRepo(tempdir())

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()
```

Description

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

Usage

```
.composeOutput(object, cacheRepo, ...)
```

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

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

Arguments

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

Value

The object, modified
.removeCacheAtts

Author(s)

Eliot McIntire

Examples

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

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

# For rasters, it is same as .prepareFileBackedRaster
try(archivist::createLocalRepo(tempdir()))

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

```r
.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
.setSubAttrInList

\hspace*{1cm} \textit{Set subattributes within a list by reference}

Description

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

Usage

\begin{verbatim}
.setSubAttrInList(object, attr, subAttr, value)
\end{verbatim}

Arguments

\begin{verbatim}
object \hspace{1cm} An arbitrary object
attr \hspace{1cm} The attribute name (that is a list object) to change
subAttr \hspace{1cm} The list element name to change
value \hspace{1cm} The new value
\end{verbatim}

.sortDotsUnderscoreFirst

\hspace*{1cm} \textit{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

\begin{verbatim}
.sortDotsUnderscoreFirst(obj)
.orderDotsUnderscoreFirst(obj)
\end{verbatim}

Arguments

\begin{verbatim}
obj \hspace{1cm} An arbitrary R object for which a \texttt{names} function returns a character vector.
\end{verbatim}

Value

The same object as \texttt{obj}, but sorted with .\texttt{objects} first.
Author(s)
Eliot McIntire

Examples
```r
elements <- c(A = "a", Z = "z", ".D" = ".d", `_C` = "_C")
.sortDotsUnderscoreFirst(elements)

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

# 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.

**Usage**

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

### S3 method for class 'Raster'

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

### S3 method for class 'RasterStack'

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

### Default S3 method:

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

**Arguments**

- `ras` The RasterLayer or RasterStack for which data type will be assessed.
- `type` Character. 'writeRaster' or 'GDAL' to return the recommended data type for writing from the raster and gdalUtils packages, respectively, or 'projectRaster' to return recommended resampling type. Default is 'writeRaster'.

**Value**

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

**Author(s)**

- Eliot McIntire
- Ceres Barros
- Ian Eddy

**Examples**

```r
## LOGIS
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)

## 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)

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

ras[14] <- NA
assessDataType(ras)
assessDataTypeGDAL

Assess the appropriate raster layer data type for GDAL

Description

Can be used to write prepared inputs on disk.
Usage

assessDataTypeGDAL(ras)

Arguments

ras

The RasterLayer or RasterStack for which data type will be assessed.

Value

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, and Tati Micheletti

Examples

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
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
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
```
Cache(
  FUN,
  ..., notOlderThan = NULL,
  .objects = 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", 0),
  cacheId = NULL,
  useCache =getOption("reproducible.useCache", TRUE),
  useCloud = FALSE,
  cloudFolderID =getOption("reproducible.cloudFolderID", NULL),
  showSimilar =getOption("reproducible.showSimilar", FALSE)
)
```

## S4 method for signature 'ANY'
```
Cache(
  FUN,
  ..., notOlderThan = NULL,
  .objects = NULL,
  outputObjects = NULL,
  algo = "xxhash64",
  cacheRepo = NULL,
  length =getOption("reproducible.length", Inf),
  ```
Cache

```r
compareRasterFileLength,
userTags = c(),
digestPathContent,
omitArgs = NULL,
classOptions = list(),
debugCache = character(),
sideEffect = FALSE,
makeCopy = FALSE,
quick = getOption("reproducible.quick", FALSE),
verbose = getOption("reproducible.verbose", 0),
cacheId = NULL,
useCache = getOption("reproducible.useCache", TRUE),
useCloud = FALSE,
cloudFolderID = getOption("reproducible.cloudFolderID", NULL),
showSimilar = getOption("reproducible.showSimilar", FALSE)
)
```

lhs %<% rhs

**Arguments**

<table>
<thead>
<tr>
<th>FUN</th>
<th>Either a function or an unevaluated function call (e.g., using <code>quote</code>).</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Arguments of <code>FUN</code> function .</td>
</tr>
<tr>
<td>notOlderThan</td>
<td>load an artifact from the database only if it was created after <code>notOlderThan</code>.</td>
</tr>
<tr>
<td>.objects</td>
<td>Character vector of objects to be digested. This is only applicable if there is a list, environment (or similar) 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.</td>
</tr>
<tr>
<td>outputObjects</td>
<td>Optional character vector indicating which objects to return. This is only relevant for list, environment (or similar) objects</td>
</tr>
<tr>
<td>algo</td>
<td>The algorithms to be used; currently available choices are md5, which is also the default, sha1, crc32, sha256, sha512, xxhash32, xxhash64, murmur32 and spookyhash.</td>
</tr>
<tr>
<td>cacheRepo</td>
<td>A repository used for storing cached objects. This is optional if <code>Cache</code> is used inside a SpaDES module.</td>
</tr>
<tr>
<td>length</td>
<td>Numeric. If the element passed to <code>Cache</code> is a <code>Path</code> class object (from e.g., <code>asPath(filename)</code>) or it is a <code>Raster</code> with file-backing, then this will be passed to <code>digest::digest</code>, essentially limiting the number of bytes to digest (for speed). This will only be used if <code>quick = FALSE</code>. Default is <code>getOption(&quot;reproducible.length&quot;)</code>, which is set to <code>Inf</code>.</td>
</tr>
<tr>
<td>compareRasterFileLength</td>
<td>Being deprecated; use <code>length</code>.</td>
</tr>
<tr>
<td>userTags</td>
<td>A character vector with Tags. These Tags will be added to the repository along with the artifact.</td>
</tr>
<tr>
<td>digestPathContent</td>
<td>Being deprecated. Use <code>quick</code>.</td>
</tr>
</tbody>
</table>
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 evaluating 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. If TRUE, little or no disk-based information will be assessed, i.e., mostly its memory content. 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. The function will assess if it is a path to a file or directory first. If not, it will treat the object as a character string. 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.

verbose  Numeric, with 0 being off, 1 being a little, 2 being more verbose etc. Above 1 will output much more information about the internals of Caching, which may help diagnose Caching challenges.

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. 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.

**cloudFolderID** A googledrive id of a folder, e.g., using `drive_mkdir()`. If left as NULL, the function will create a cloud folder with a warning. The warning will have the cloudFolderID that should be used in subsequent calls. It will also be added to options("reproducible.cloudFolderID"), but this will not persist across sessions.

**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.

**lhs** A name to assign to.

**rhs** A function call

**Details**

Caching R objects using `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-base 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

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

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

**Value**

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

**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`.

**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.

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 if (`makeCopy` 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
`cache`, `.robustDigest`

Examples
```r
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
ranNumsC <- Cache(cacheRepo = tmpDir) %C% rnorm(10, 16) # recovers cached copy
ranNumsD <- Cache(quote(rnorm(n = 10, 16)), cacheRepo = tmpDir) # recovers cached copy

# experimental devMode

### 

# During development, we often redefine function internals
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 8be9cf2a072dbb0515c5f0b3578f474

# 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:
   browseVignettes(package = "reproducible")

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

---

**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", ...)`

### 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 and spookyhash.
- **calledFrom**: a Character string, length 1, with the function to compare with. Default is "Cache". All other values may not produce robust `CacheDigest` results.
- **...**: 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
```r
## 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 googledrive, with a warning.

Usage
checkAndMakeCloudFolderID(cloudFolderID = NULL)

Arguments
cloudFolderID The google folder ID where cloud caching will occur.

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

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

Usage

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

```r
## 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)
```
checkPath::summary(localRepo)
unlink(tmpDir, recursive = TRUE)

## get a particular commit by sha
sha <- "8179e1910e7c617fdeacad8f9d81323e6aad57c3"
localRepo <- checkoutVersion(paste0(repo, "@", sha), localRepoPath = tmpDir)
git2r::summary(localRepo)
unlink(tmpDir, recursive = TRUE)
rm(localRepo, repo)

## End(Not run)

---

**checkPath**

*Check directory path*

**Description**

Checks the specified path to a directory for formatting consistencies, such as trailing slashes, etc.

**Usage**

checkPath(path, create)

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

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

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

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

**Arguments**

- **path**
  A character string corresponding to a directory path.

- **create**
  A logical indicating whether the path should be created if it doesn’t exist. Default is FALSE.

**Value**

Character string denoting the cleaned up filepath.
Note

This will not work for paths to files. To check for existence of files, use `file.exists`, or use `file_test` with `op = "-f"`. To normalize a path to a file, use `normPath` or `normalizePath`.

See Also

`file.exists`, `dir.create`.

Examples

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

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

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

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

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

unlink(tmpdir, recursive = TRUE)
```

Checksums

### Calculate checksum

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

Usage

```r
Checksums(
    path,
    write,
    quickCheck = FALSE,
    checksumFile = file.path(path, "CHECKSUMS.txt"),
    files = NULL,
)```
...)

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

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

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.
- **...**: 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

```r
## 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 archivist package functions. They allow the user a bit of control over what is being cached.

Usage

```r
clearCache(
  x,
  userTags = character(),
  after,
  before,
  ask =getOption("reproducible.ask"),
  useCloud = FALSE,
```
Arguments

**x**
A simList or a directory containing a valid archivist 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 id of a folder, e.g., using `drive_mkdir()`. If left as `NULL`, the function will create a cloud folder with a warning. The warning will have the `cloudFolderID` that should be used in subsequent calls. It will also be added to `options("reproducible.cloudFolderID")`, but this will not persist across sessions.
- **...**: 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.
clearCache

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, splitTagsLocal. 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
ranNumsC <- Cache(cacheRepo = tmpDir) %C% rnorm(10, 16) # recovers cached copy
ranNumsD <- Cache(quote(rnorm(n = 10, 16)), cacheRepo = tmpDir) # recovers cached copy

# Any minor change makes it different
ranNumsE <- Cache(cacheRepo = tmpDir) %C% rnorm(10, 6) # 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]$artifact
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)
clearStubArtifacts

Clear erroneous archivist artifacts

Description

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.

Usage

clearStubArtifacts(repoDir = NULL)

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

Arguments

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

Value

Invoked for its side effect on the repoDir.

Author(s)

Eliot McIntire

Examples

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

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

# clear out any stub artifacts
showCache(tmpDir)

file2Remove <- dir(file.path(tmpDir, "gallery"), 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
Please use Cache, with args useCloud and cloudFolderID.

Usage
cloudCache(…)

Arguments
... Passed to Cache

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.
cloudDownload

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)

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(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 id of a folder, e.g., using drive_mkdir(). If left as NULL, the function will create a cloud folder with a warning. The warning will have the cloudFolderID that should be used in subsequent calls. It will also be added to options("reproducible.cloudFolderID"), but this will not persist across sessions.

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

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

Description

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

Usage

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

<table>
<thead>
<tr>
<th>isInRepo</th>
<th>A data.table with the information about an object that is in the local cacheRepo</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputHash</td>
<td>The cacheId of the object to upload</td>
</tr>
<tr>
<td>gdriveLs</td>
<td>The result of googledrive::drive_ls(as_id(cloudFolderID), pattern = &quot;outputHash&quot;)</td>
</tr>
<tr>
<td>cacheRepo</td>
<td>A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.</td>
</tr>
<tr>
<td>cloudFolderID</td>
<td>A googledrive id of a folder, e.g., using drive_mkdir(). If left as NULL, the function will create a cloud folder with a warning. The warning will have the cloudFolderID that should be used in subsequent calls. It will also be added to options(&quot;reproducible.cloudFolderID&quot;), but this will not persist across sessions.</td>
</tr>
<tr>
<td>output</td>
<td>The output object of FUN that was run in Cache</td>
</tr>
</tbody>
</table>

cloudWriteOld  

Basic tool for using cloud-based caching

Description

Very experimental

Usage

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

Arguments

<table>
<thead>
<tr>
<th>object</th>
<th>The R object to write to cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>digest</td>
<td>The cacheId of the input arguments, outputted from cloudCheckOld0ld</td>
</tr>
<tr>
<td>cloudFolderID</td>
<td>The google folder ID where a new object should be written</td>
</tr>
<tr>
<td>checksums</td>
<td>A data.table that is outputted from cloudCheckOld0ld that is the the checksums file</td>
</tr>
<tr>
<td>checksumsFileID</td>
<td>A google file ID where the checksums data.table is located, provided as a character string.</td>
</tr>
<tr>
<td>futurePlan</td>
<td>Which future::plan to use. Default: getOption(&quot;reproducible.futurePlan&quot;)</td>
</tr>
</tbody>
</table>

See Also

cloudSyncCacheOld0ld, cloudCheckOld0ld
**compareNA**  
NA-aware comparison of two vectors

**Description**
Copied from [http://www.cookbook-r.com/Manipulating_data/Comparing_vectors_or_factors_with_NA/](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**
```r
compareNA(v1, v2)
```

**Arguments**
- **v1**  
A vector
- **v2**  
A vector

**Examples**
```r
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**
```r
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`).
Copy

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, filebackedDir, ...)

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

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

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

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

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

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

Arguments

object An R object (likely containing environments) or an environment.

filebackedDir A directory to copy any files that are backing R objects, currently only valid for Raster classes. Defaults to tempdir(), which is unlikely to be very useful. 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.

... Only used for custom Methods

Author(s)

Eliot McIntire

See Also

.robustDigest

Examples

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

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

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

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

```r
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.
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, ...)

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

## S3 method for class 'spatialObjects'
cropInputs(
  x,
  studyArea = NULL,
  rasterToMatch = NULL,
  extentToMatch = NULL,
)
cropInputs

    extentCRS = NULL,
... )

## S3 method for class 'sf'
cropInputs(
    x,
    studyArea = NULL,
    rasterToMatch = NULL,
    extentToMatch = NULL,
    extentCRS = NULL,
    ...
)

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.

... 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

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) <- "+init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
# make a "study area" that is subset of larger dataset
cords <- 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(cords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+init=epsg:4326 +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 = TRUE,
  filename1 = NULL,
  destinationPath = getOption("reproducible.destinationPath"),
  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).
**determineFilename**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationPath</td>
<td>Optional. If filename2 is a relative file path, then this will be the directory of the resulting absolute file path.</td>
</tr>
<tr>
<td>prefix</td>
<td>The character string to prepend to filename1, if filename2 not provided.</td>
</tr>
</tbody>
</table>

Additional arguments passed to methods. For spatialObjects, 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.

---

* 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" file name (`filename2`). When this is used within `postProcess`, it is straightforward.

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 referred 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 `filename1/targetFile` name, with prefix prefixed to the base-name(`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 as 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
Examples

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

# make a SpatialPolygon
cords1 <- 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(cords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
crs(shpEcozone) <- "+init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
cords <- 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(cords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

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

# A wrapper around a set of downloading functions

downloadFile

Description

Currently, this only deals with drive_download, and download_file.

Usage

downloadFile(
  archive,
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 \texttt{preProcess}.

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 \texttt{postProcess}. Currently, the internal checksumming does not checksum the file after it is postProcessed (e.g., cropped/reprojected/masked). Using Cache around \texttt{prepInputs} will do a sufficient job in these cases. See table in \texttt{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 \texttt{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 \texttt{Checksums} (the quickCheck argument), and to \texttt{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 \texttt{Checksums(\ldots, 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 \texttt{prepInputs} or \texttt{preProcess}, comparing the file on hand with the ad hoc CHECKSUMS.txt. See table in \texttt{preProcess}.
extractFromArchive

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.

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

...  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
)

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)
fastMask

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
... 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

Description
This alternative to raster::mask is included here.

Usage
fastMask(x, y, cores = NULL, useGDAL = getOption("reproducible.useGDAL", TRUE))

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, 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.

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

---

**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.
getGDALVersion

Usage
Filenames(obj)

## S4 method for signature 'ANY'
Filenames(obj)

## S4 method for signature 'Raster'
Filenames(obj)

## S4 method for signature 'RasterStack'
Filenames(obj)

## S4 method for signature 'environment'
Filenames(obj)

## S4 method for signature 'list'
Filenames(obj)

Arguments
obj A Raster* object (i.e., RasterLayer, RasterStack, RasterBrick)

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
getUserTags

Cache helpers

Description
A few helpers to get specific things from the cache repository

Usage
getUserTags(cacheRepo, shownCache, cacheId, concatenated = TRUE)
getCacheId(cacheRepo, shownCache, artifact)
getArtifact(cacheRepo, shownCache, cacheId)

Arguments

- **cacheRepo**: A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
- **shownCache**: Primary way of supplying `cacheRepo`; the data.table obj resulting from `showCache`, i.e., it will override `cacheRepo`. If this and `cacheRepo` are missing, then it will default to `getOption('reproducible.cachePath')`
- **cacheId**: A character vector of cacheId values to use in the cache
- **concatenated**: Logical. If TRUE, the returned `userTags` will be concatenated `tagKey:tagValue`.
- **artifact**: Character vector of artifact values in the `artifact` column of `showCache`

Value

- `getCacheId` returns the `cacheId` values for 1 or more artifacts in the cache.
- `getArtifact` returns the `artifact` value for 1 or more entries in the cache, by `cacheId`.

installedVersions

Determine versions all installed packages

Description
This code is adapted from `installed.versions`. It will be anywhere from 2x to 10x faster than the `installed.versions` function. This is also many times faster than `utils::installed.packages`, especially if only a subset of "all" packages in `libPath` are desired (1000x ? for the 1 package case).

Usage

`installedVersions(packages, libPath)`
installVersions

Arguments

- `packages` Character vector of packages to determine which version is installed in the `libPath`.
- `libPath` The library path where all packages should be installed, and looked for to load (i.e., call `library()`)

Examples

```r
installedVersions("reproducible", .libPaths()[1])
```

installVersions

Install exact package versions from a package version text file & GitHub

Description

Install exact package versions from a package version text file & GitHub

Usage

```r
installVersions(
  gitHubPackages,
  packageVersionFile = ".packageVersions.txt",
  libPath = .libPaths()[1],
  standAlone = FALSE,
  repos = getOption("repos")["CRAN"]
)
```

Arguments

- `gitHubPackages` Character vectors indicating repository/packageName@branch
- `packageVersionFile` Path to the package version file, defaults to the `".packageVersions.txt"`. This uses CRAN, CRAN archives, or MRAN (accessed via `versions::install.versions`) for remote repositories. This will attempt to install all packages in the `packageVersionFile`, with their exact version described in that file. For GitHub packages, it will use `install_github`. This will be called internally by `require`, and so often doesn’t need to be used by a user.
  Because of potential conflicts with loaded packages, this function will run `install.packages` in a separate `R` process.
- `libPath` The library path where all packages should be installed, and looked for to load (i.e., call `library()`)

**standAlone**  Logical. If TRUE, all packages will be installed and loaded strictly from the libPaths only. If FALSE, all .libPaths will be used to find the correct versions. This can be create dramatically faster installs if the user has a substantial number of the packages already in their personal library. In the case of TRUE, there will be a hidden file place in the libPath directory that lists all the packages that were needed during the Require call. Default FALSE to minimize package installing.

repos  The remote repository (e.g., a CRAN mirror), passed to either install.packages, install_github or installVersions.

**Examples**

```r
## Not run:
# requires the packageVersionFile -- this doesn't work -- safer to use Require
installVersions("PredictiveEcology/reproducible@development")

# make a package version snapshot -- this will be empty because no packages in directory
tempPkgFolder <- file.path(tempdir(), "Packages")
dir.create(tempPkgFolder)
packageVersionFile <- file.path(tempPkgFolder, ".packageVersion.txt")
pkgSnapshot(libPath = tempPkgFolder, packageVersionFile)

Require("crayon", libPath = tempPkgFolder)  # install.packages first, then library

# install a specific version
# make a package version snapshot
packageVersionFile <- file.path(tempPkgFolder, ".packageVersion.txt")
pkgSnapshot(libPath = tempPkgFolder, packageVersionFile, standAlone = FALSE)

installVersions("crayon", packageVersionFile = packageVersionFile)

## End(Not run)
```

**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)`

**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.
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

```r
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**

```r
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`.
maskInputs

Usage

maskInputs(x, studyArea, ...)

## S3 method for class 'Raster'
maskInputs(x, studyArea, rasterToMatch, maskWithRTM = FALSE, ...)

## S3 method for class 'Spatial'
maskInputs(x, studyArea, ...)

## S3 method for class 'sf'
maskInputs(x, studyArea, ...)

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 `spTransform`ed 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.

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
cords1 <- 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(cords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
   crs(shpEcozone) <- "+init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
cords <- 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(cords)
mergeCache <- polygons(list(sr1), "s1")
studyArea <- SpatialPolygons(list(srs1), 1L)

###
crs(studyArea) <- "+init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# Try manually, individual pieces
shpEcozonePostProcessed <- postProcess(shpEcozone, studyArea = studyArea)

shpEcozoneReprojected <- projectInputs(shpEcozone, studyArea)
shpEcozoneCropped <- cropInputs(shpEcozone, studyArea)
shpEcozoneClean <- fixErrors(shpEcozone)
shpEcozoneMasked <- maskInputs(shpEcozone, studyArea)

setwd(ow)

mergeCache

### Merge two cache repositories together

Description

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.

Usage

mergeCache(cacheTo, cacheFrom)

## S4 method for signature 'ANY'
mergeCache(cacheTo, cacheFrom)

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

Details

This is still experimental

Value

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

Description

A shortcut to create a .libPaths() with only two directories

Usage

newLibPaths(libPath)

Arguments

libPath A path that will be the new .libPaths()[1]

Value

Invisibly, the new .libPaths().

Examples

## Not run:
newLibPaths("testPackages")
.libPaths() # new .libPaths

## End(Not run)

normPath

Normalize filepath

Description

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

Usage

normPath(path)

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

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

```r	normPath(path)
```

## S4 method for signature 'missing'

```r	normPath()
```

### Arguments

- `path` A character vector of filepaths.

### Value

Character vector of cleaned up filepaths.

### Examples

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

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

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

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

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

unlink(tmpdir, recursive = TRUE)
```

---

### 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.
objSize

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**

\[
\text{objSizeSession(sumLevel = Inf, enclosingEnvs = TRUE, .prevEnvirs = list())}
\]

**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 [here](https://www.r-bloggers.com/using-closures-as-objects-in-r/) and [here](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**

```r
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 31 MB, with all functions, objects
# and imported functions

package_dependenciesMem

Memoised version of package_dependencies

Description

This has a 6 minute memory time window.

Usage

package_dependenciesMem(
  packages = NULL,
  db = NULL,
  which = c("Depends", "Imports", "LinkingTo"),
  recursive = FALSE,
  reverse = FALSE,
  verbose = getOption("verbose")
)

Arguments

packages a character vector of package names.
db character matrix as from available.packages
which a character vector listing the types of dependencies, a subset of c("Depends", "Imports", "LinkingTo"), Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances".
recursive logical: should (reverse) dependencies of (reverse) dependencies (and so on) be included?
reverse logical: if FALSE (default), regular dependencies are calculated, otherwise reverse dependencies.
verbose logical indicating if output should monitor the package search cycles.
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 an 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
Description

STILL EXPERIMENTAL. THIS MAY NOT WORK AS ANTICIPATED.

Usage

lhs %C% rhs

Arguments

lhs     A value or the magrittr placeholder.

rhs     A function call using the magrittr semantics.

Details

This pipe can only be used at any point in a pipe chain, but must be preceded by Cache(...) (which allows other Cache 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 straightforward to insert into existing code that uses pipes (Cache(...) %C% ... remaining pipes).

Note

This is still experimental; use with care.

Examples

library(magrittr) # standard pipe
# dontrun( # 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) %>%
  rnorm(10, 16) %>%
  mean() %>%
  prod(., 6)
e <- Cache(cacheRepo = tmpdir) %>%
  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 %>%
a1 <- Cache(cacheRepo = tmpdir) %>%
  seq(1, 10) %>%
  rnorm(2, mean = .) %>%
  mean() %>% # Cache pipe here --
  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() %>% # Cache pipe here --
  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(.) %>% # Cache pipe here --
  rnorm(4, mean = .) # These are same the 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(.) %>% # Cache pipe here --
  rnorm(4, mean = .) # These are same the mean is same each time
sum(b1 - b2) # 0 # i.e., numbers are same
pkgDep

Determine package dependencies, first looking at local filesystem

Description

This is intended to replace package_dependencies or pkgDep in the miniCRAN package, but with modifications for speed. It will first check local package directories in libPath, and if the function cannot find the packages there, then it will use package_dependencies.

pkgDep2 is a convenience wrapper of pkgDep that "goes one level in" i.e., the first order dependencies, and runs the pkgDep on those.

Usage

pkgDep(
  packages,
  libPath,
  recursive = TRUE,
  depends = TRUE,
  imports = TRUE,
  suggests = FALSE,
  linkingTo = TRUE,
  repos = getOption("repos"),
  refresh = FALSE,
  verbose = getOption("reproducible.verbose")
)

pkgDep2(
  packages,
  recursive = TRUE,
  depends = TRUE,
  imports = TRUE,
  suggests = FALSE,
  linkingTo = TRUE,
  repos = getOption("repos"),
  refresh = FALSE,
  verbose = getOption("reproducible.verbose"),
  sorted = TRUE
)

Arguments

packages a character vector of package names.
libPath The library path where all packages should be installed, and looked for to load (i.e., call library)
recursive Logical. Should dependencies of dependencies be searched, recursively. NOTE: Dependencies of suggests will not be recursive. Default TRUE.
depends Logical. Include packages listed in "Depends". Default TRUE.
imports Logical. Include packages listed in "Imports". Default TRUE.
suggests Logical. Include packages listed in "Suggests". Default FALSE.
linkingTo Logical. Include packages listed in "LinkingTo". Default TRUE.
repos The remote repository (e.g., a CRAN mirror), passed to either install.packages, install_github or installVersions.
refresh There is an internal type of caching. If the results are wrong, likely set refresh = TRUE.
verbose logical indicating if output should monitor the package search cycles.
sorted Logical. If TRUE, the default, the packages will be sorted in the returned list from most number of dependencies to least.

Note

package_dependencies and pkgDep will differ under the following circumstances:

1. GitHub packages are not detected using tools::package_dependencies;
2. tools::package_dependencies does not detect the dependencies of base packages among themselves, e.g., methods depends on stats and graphics.

Examples

pkgDep("crayon")
pkgDep2("reproducible")

pkgSnapshot Take a snapshot of all the packages and version numbers

Description

This can be used later by installVersions to install or re-install the correct versions.

Usage

pkgSnapshot(packageVersionFile, libPath, standAlone = FALSE)
Arguments

packageVersionFile
A filename to save the packages and their currently installed version numbers. Defaults to ".packageVersions.txt".

libPath
The path to the local library where packages are installed. Defaults to the .libPaths()[1].

standAlone
Logical. If TRUE, all packages will be installed and loaded strictly from the libPaths only. If FALSE, all .libPaths will be used to find the correct versions. This can be create dramatically faster installs if the user has a substantial number of the packages already in their personal library. In the case of TRUE, there will be a hidden file place in the libPath directory that lists all the packages that were needed during the Require call. Default FALSE to minimize package installing.

Details

A file is written with the package names and versions of all packages within libPath. This can later be passed to Require.

Examples

pkgSnapFile <- tempfile()
pkgSnapshot(pkgSnapFile, .libPaths()[1])
data.table::fread(pkgSnapFile)

postProcess
Generic function to post process objects

Description

The method for spatialObjects (Raster* and Spatial*) will crop, reproject, and mask, in that order. This function is a wrapper for cropInputs, fixErrors, projectInputs, maskInputs and writeOutputs, with a decent amount of data manipulating 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 'spatialObjects'
postProcess(
postProcess

x,
filename1 = NULL,
filename2 = TRUE,
studyArea = NULL,
rasterToMatch = NULL,
overwrite =getOption("reproducible.overwrite", TRUE),
useSAcrs = FALSE,
useCache =getOption("reproducible.useCache", FALSE),
...
)

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

Arguments

x An object of postProcessing, e.g., spatialObjects. See individual methods.
...

... passed to::

<table>
<thead>
<tr>
<th>Function</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>cropInputs</td>
<td>crop</td>
</tr>
<tr>
<td>projectInputs</td>
<td>projectRaster</td>
</tr>
<tr>
<td>maskInputs</td>
<td>fastMask or intersect</td>
</tr>
<tr>
<td>fixErrors</td>
<td>buffer</td>
</tr>
<tr>
<td>writeOutputs</td>
<td>writeRaster or shapefile</td>
</tr>
<tr>
<td>determineFilename</td>
<td></td>
</tr>
</tbody>
</table>

* 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").

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::

<table>
<thead>
<tr>
<th></th>
<th>rasterToMatch</th>
<th>studyArea</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>extent</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>resolution</td>
<td>Yes</td>
<td>No</td>
<td>rasterToMatch</td>
</tr>
<tr>
<td>projection</td>
<td>Yes</td>
<td>No*</td>
<td>rasterToMatch*</td>
</tr>
<tr>
<td>alignment</td>
<td>Yes</td>
<td>No</td>
<td>rasterToMatch</td>
</tr>
<tr>
<td>mask</td>
<td>No**</td>
<td>Yes</td>
<td>studyArea**</td>
</tr>
</tbody>
</table>
* Can be overridden with useSAcrs

** Will mask with NAs from rasterToMatch if maskWithRTM

If `targetFile` is a Spatial* object:

<table>
<thead>
<tr>
<th></th>
<th>rasterToMatch</th>
<th>studyArea</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>extent</td>
<td>Yes</td>
<td>Yes</td>
<td>rasterToMatch</td>
</tr>
<tr>
<td>resolution</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>projection</td>
<td>Yes</td>
<td>No*</td>
<td>rasterToMatch*</td>
</tr>
<tr>
<td>alignment</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>mask</td>
<td>No</td>
<td>Yes</td>
<td>studyArea</td>
</tr>
</tbody>
</table>

* Can be overridden with useSAcrs

See Also

`prepInputs`

Examples

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

# make a SpatialPolygon
cords1 <- 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(cords1)
Srs1 <- Polygons(list(Sr1), "s1")
shpEcozone <- SpatialPolygons(list(Srs1), 1L)
crs(shpEcozone) <- "+init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# make a "study area" that is subset of larger dataset
cords <- 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(cords)
Srs1 <- Polygons(list(Sr1), "s1")
StudyArea <- SpatialPolygons(list(Srs1), 1L)
crs(StudyArea) <- "+init=epsg:4326 +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)
```
```r
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) <- "+init=epsg:4326 +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) <- "+init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

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

**Description**

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 postProcessing (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.

**Usage**

```r
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", FALSE),
...
)

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.

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 or character string indicating the function to use to load targetFile into an R object, e.g., in form with package name: "raster::raster".

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.
**prepInputs**

- **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").

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. See details and examples.

**Stage 1 - Getting data**

See prepProcess 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.spatialObjects.

Understanding various combinations of rasterToMatch and/or studyArea: Please see postProcess.spatialObjects.

**purge**

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

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,
# \code{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) <- "+init=epsg:4326 +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,
targetFile = asPath(ecozoneFilename),
alsoExtract = 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
LCC2005 <- prepInputs(url = url,
destinationPath = asPath(dPath),
studyArea = StudyArea)
LCC2005 <- Cache(prepInputs, url = url,
targetFile = asPath(lcc2005Filename),
archive = asPath("LandCoverOfCanada2005_V1_4a.zip"),
destinationPath = asPath(dPath),
studyArea = StudyArea)

# if wrapped with Cache, will be fast second time, very fast 3rd 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)

---
**preProcess**  
**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**

```r
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,
  useCache = getOption("reproducible.useCache", FALSE),
  ...
)
```

**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.

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 or character string indicating the function to use to load targetFile into an R object, e.g., in form with package name: "raster::raster".

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.

useCache  Passed to Cache in various places. Defaults to getOption("reproducible.useCache").

...  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. 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

<table>
<thead>
<tr>
<th># Params</th>
<th>url</th>
<th>targetFile</th>
<th>archive</th>
<th>alsoExtract</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>char</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Download, extract all files if an archive, guess at targetFile</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
<td>char</td>
<td>NULL</td>
<td>NULL</td>
<td>load targetFile into R</td>
</tr>
</tbody>
</table>
* 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, ...)
```

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

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

---
targetCRS = NULL,
rasterToMatch = NULL,
cores = NULL,
useGDAL = getOption("reproducible.useGDAL", TRUE),
...
)

## S3 method for class 'sf'
projectInputs(x, targetCRS, ...)

## S3 method for class 'Spatial'
projectInputs(x, targetCRS, ...)

Arguments

- **x**: A `Raster*`, `Spatial*` or `sf` object
- **targetCRS**: The CRS of x at the end of this function (i.e., the goal)
- **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, 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.

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)
reproducibleOptions

crs(shpEcozone) <- "+init=epsg:4326 +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) <- "+init=epsg:4326 +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

<table>
<thead>
<tr>
<th>OPTION</th>
<th>DEFAULT VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ask</td>
<td>TRUE</td>
<td>Used in clearCache and keepCache</td>
</tr>
<tr>
<td>cachePath</td>
<td>.reproducibleTempCacheDir</td>
<td>Used in Cache and many others,</td>
</tr>
<tr>
<td>destinationPath</td>
<td>NULL</td>
<td>Used in prepInputs, preProcess</td>
</tr>
<tr>
<td>futurePlan</td>
<td>FALSE</td>
<td>On Linux OSs, Cache and cloud</td>
</tr>
<tr>
<td>inputPaths</td>
<td>NULL</td>
<td>Used in prepInputs, preProcess</td>
</tr>
<tr>
<td>inputPathsRecursive</td>
<td>FALSE</td>
<td>Used in prepInputs, preProcess</td>
</tr>
<tr>
<td>overwrite</td>
<td>FALSE</td>
<td>Used in prepInputs, preProcess</td>
</tr>
</tbody>
</table>
**Require**

- quick: FALSE (Used in Cache. This will cause the file.size(file) to be used instead of the digest::digest(file). Less robust to changes, but faster. NOTE: this will only affect objects on disk.)
- showSimilar: Passed to Cache. Default FALSE.
- useCache: TRUE (Passed to Cache. Default FALSE.)
- useCloud: FALSE (Passed to Cache. Default FALSE.)
- useGDAL: TRUE (Passed to useGDAL in projectInputs. Raster. Default TRUE.)
- useMemoise: TRUE (Used in Cache. If TRUE, recovery of cached elements from the cacheRepo will use memoise::memoise. This means that the 3rd time running a function will be much faster than the 1st (create cache entry) or 2nd (recover from the SQLite database on dist). 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.)
- useNewDigestAlgorithm: TRUE (This will mean that previous cache repositories will be defunct. This new algorithm 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: 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: file.path(dirname(.reproducibleTempCacheDir), "checksums.rds") (Used in cloudCache)
- length: 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: "http://github.com/PredictiveEcology/reproducible" (User agent for downloads using this package.)

**Description**

This is an "all in one" function that will run install.packages for CRAN packages, remotes::install_github for https://github.com/ packages and will install specific versions of each package if there is a packageVersionFile supplied. Plus, when packages is provided as a character vector, or a packageVersionFile is supplied, all package dependencies will be first assessed for unique(dependencies) so the same package is not installed multiple times. Finally library is called on the packages. If packages are already installed (packages supplied), and their version numbers are exact (when packageVersionFile is supplied), then the "install" component will be skipped very quickly with a message.

**Usage**

```r
Require(  
  packages,  
  packageVersionFile,  
  libPath = .libPaths()[1],  
  install_githubArgs = list(),  
  install.packagesArgs = list(),  
  standAlone = FALSE,  
  repos =getOption("repos"),  
  forget = FALSE  
)
```
**Arguments**

- **packages**: Character vector of packages to install via `install.packages`, then load (i.e., with `library`). If it is one package, it can be unquoted (as in `require`)

- **packageVersionFile**: If provided, then this will override all `install.package` calls with `versions::install.versions`

- **libPath**: The library path where all packages should be installed, and looked for to load (i.e., call `library`)

- **install_githubArgs**: List of optional named arguments, passed to `install_github`.

- **install.packagesArgs**: List of optional named arguments, passed to `install.packages`.

- **standAlone**: Logical. If `TRUE`, all packages will be installed and loaded strictly from the `libPaths` only. If `FALSE`, all `.libPaths` will be used to find the correct versions. This can be create dramatically faster installs if the user has a substantial number of the packages already in their personal library. In the case of `TRUE`, there will be a hidden file place in the `libPath` directory that lists all the packages that were needed during the `Require` call. Default `FALSE` to minimize package installing.

- **repos**: The remote repository (e.g., a CRAN mirror), passed to either `install.packages`, `install_github` or `installVersions`.

- **forget**: Internally, this function identifies package dependencies using a memoised function for speed on reuse. But, it may be inaccurate in some cases, if packages were installed manually by a user. Set this to `TRUE` to refresh that dependency calculation.

**Details**

`standAlone` will either put the `Required` packages and their dependencies all within the `libPath` (if `TRUE`) or if `FALSE` will only install packages and their dependencies that are otherwise not installed in `.libPaths()`, i.e., the personal or base library paths. Any packages or dependencies that are not yet installed will be installed in `libPath`. Importantly, a small hidden file (named `_packageVersionsAuto.txt`) will be saved in `libPath` that will store the *information* about the packages and their dependencies, even if the version used is located in `.libPaths()`, i.e., not the `libPath` provided. This hidden file will be used if a user runs `pkgSnapshot`, enabling a new user to rebuild the entire dependency chain, without having to install all packages in an isolated directory (as does `packrat`). This will save potentially a lot of time and disk space, and yet maintain reproducibility. *NOTE*: since there is only one hidden file in a `libPath`, any call to `pkgSnapshot` will make a snapshot of the most recent call to `Require`.

To build a snapshot of the desired packages and their versions, first run `Require` with all packages, then `pkgSnapshot`. If a `libPath` is used, it must be used in both functions.

This function works best if all required packages are called within one `Require` call, as all dependencies can be identified together, and all package versions will be saved automatically (with `standAlone = TRUE` or `standAlone = FALSE`), allowing a call to `pkgSnapshot` when a more permanent record of versions can be made.
Note

This function will use memoise internally to determine the dependencies of all packages. This will speed up subsequent calls to `Require` dramatically. However, it will not take into account version numbers for this memoised step. If package versions are updated manually by the user, then this cached element should be wiped, using `forget = TRUE`.

Examples

```r
## Not run:
# simple usage, like conditional install.packages then library
Require("stats") # analogous to require(stats), but slower because it checks for
  # pkg dependencies, and installs them, if missing
  tempPkgFolder <- file.path(tempdir(), "Packages")

  # use standAlone, means it will put it in libPath, even if it already exists
  # in another local library (e.g., personal library)
  Require("crayon", libPath = tempPkgFolder, standAlone = TRUE)

  # make a package version snapshot
  packageVersionFile <- file.path(tempPkgFolder, ".packageVersion.txt")
  pkgSnapshot(libPath=tempPkgFolder, packageVersionFile)

  # confirms that correct version is installed
  Require("crayon", packageVersionFile = packageVersionFile)

  # Create mismatching versions -- desired version is older than current installed
  # This will try to install the older version, overwriting the newer version
  desiredVersion <- data.frame(instPkgs="crayon", instVers = "1.3.2", stringsAsFactors = FALSE)
  write.table(file = packageVersionFile, desiredVersion, row.names = FALSE)
  # won't work because newer crayon is loaded
  Require("crayon", packageVersionFile = packageVersionFile)

  # unload it first
  detach("package:crayon", unload = TRUE)

  # run again, this time, correct "older" version installs in place of newer one
  Require("crayon", packageVersionFile = packageVersionFile)

  # Mutual dependencies, only installs once -- e.g., httr
  Require(c("cranlogs", "covr"), libPath = tempPkgFolder)

  ## End(Not run)
```

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.
**searchFull**

Usage

```r
tretry(expr, retries = 5, silent = FALSE)
```

Arguments

- **expr**: Expression to run.
- **retries**: Numeric. The maximum number of retries.
- **silent**: Logical indicating whether to `try` silently.

Details

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

```r
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.`
tempdir2

See Also
    search

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

---

tempdir2 Make a temporary sub-directory

Description
    Create a temporary subdirectory in tempdir().

Usage
    tempdir2(sub)

Arguments
    sub Character string, length 1. Can be a result of file.path("smth","smth2") for
        nested temporary sub directories.

---

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 archivist repository, using future::future*

**Description**

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

**Usage**

```r
writeFuture(written, outputToSave, cacheRepo, userTags)
```

**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`

---

writeOutputs  
*Write module inputs on disk*

**Description**

Can be used to write prepared inputs on disk.

**Usage**

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

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

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

## 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),
  ...
)

## 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

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) <- "+init=epsg:4326 +proj=longlat +datum=WGS84 +no_def +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) <- "+init=epsg:4326 +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

*Topic datasets
  .unrarPath, 93
  .addChangedAttr, 5
  .addChangedAttr, ANY-method (.addChangedAttr), 5
  .addTagsToOutput, 6
  .addTagsToOutput, ANY-method (.addTagsToOutput), 6
  .cacheMessage, 6
  .cacheMessage, ANY-method (.cacheMessage), 6
  .checkCacheRepo, 7
  .checkCacheRepo, ANY-method (.checkCacheRepo), 7
  .debugCache, 8
  .digest, 33
  .installPackages, 9
  .objSizeInclEnviros, 10
  .objSizeInclEnviros, ANY-method (.objSizeInclEnviros), 10
  .objSizeInclEnviros, environment-method (.objSizeInclEnviros), 10
  .orderDotsUnderscoreFirst (.sortDotsUnderscoreFirst), 15
  .preDigestByClass, 11
  .preDigestByClass, ANY-method (.preDigestByClass), 11
  .prefix, 11
  .prepareFileBackedRaster, 12
  .prepareOutput, 13
  .prepareOutput, ANY-method (.prepareOutput), 13
  .prepareOutput, RasterLayer-method (.prepareOutput), 13
  .removeCacheAtts, 14
  .robustDigest, 28, 46
  .setSubAttrInList, 15
  .sortDotsUnderscoreFirst, 15
  .suffix (.prefix), 11
  .tagsByClass, 16
  .tagsByClass, ANY-method (.tagsByClass), 16
  .unrarPath (unrarPath), 93
  <% (Cache), 22
  % (pipe), 71
  asPath (Path-class), 70
  assessDataType, 17
  assessDataTypeGDAL, 19
  available.packages, 69
  basename, 21
  basename2, 21
  buffer, 51, 76
  Cache, 22, 38, 40–42, 53, 80, 81, 85, 88, 89
  cache, 25, 26, 28
  Cache, ANY-method (Cache), 22
  CacheDigest, 29, 89
  cc (clearCache), 35
  checkAndMakeCloudFolderID, 30
  checkGDALVersion, 30
  checkoutVersion, 31
  checkPath, 32
  checkPath, character, logical-method (checkPath), 32
  checkPath, character, missing-method (checkPath), 32
  checkPath, missing, ANY-method (checkPath), 32
  checkPath, NULL, ANY-method (checkPath), 32
  Checksums, 33, 53, 80, 85
  Checksums, character, logical-method (Checksums), 33
  Checksums, character, missing-method (Checksums), 33
  clearCache, 26, 35, 88
  clearCache, ANY-method (clearCache), 35
clearStubArtifacts, 39
cloudCache, 40, 42, 89
cloudCheckOld, 40, 42, 43
cloudDownload, 41
cloudSyncCacheOld, 40, 41, 43
cloudUpload, 42
cloudWriteOld, 40–42, 43
dataType, 17, 20
determineFilename, 50, 50, 51, 76, 77, 81
digest, 25, 34
dir.create, 33
download.file, 52, 81
downloadFile, 52, 82, 88
drive_download, 52, 81
extractFromArchive, 54, 82
fastMask, 51, 55, 62, 76
file.copy, 61
file.exists, 33
file.link, 61
file.symlink, 61
file_test, 33
Filenames, 56
Filenames,ANY-method (Filenames), 56
Filenames,environment-method (Filenames), 56
Filenames,List-method (Filenames), 56
Filenames,Raster-method (Filenames), 56
Filenames,RasterStack-method (Filenames), 56
fixErrors, 51, 75–77
getArtifact (getUserTags), 58
getCacheId (getUserTags), 58
getGDALVersion, 57
getUserTags, 58
install_github, 59
installed.versions, 58
installVersions, 59
intersect, 51, 76
keepCache, 88
keepCache (clearCache), 35
keepCache,ANY-method (clearCache), 35
linkOrCopy, 60
makeMemoisable, 62
maskInputs, 51, 62, 75–77, 81
mergeCache, 38, 64
mergeCache,ANY-method (mergeCache), 64
newLibPaths, 65
normalizePath, 33
normPath, 33, 65
normPath,character-method (normPath), 65
normPath,list-method (normPath), 65
normPath,missing-method (normPath), 65
normPath,NULL-method (normPath), 65
objSize, 66
objSizeSession (objSize), 66
options, 4
package_dependencies, 73
package_dependenciesMem, 69
Path-class, 70
pipe, 71
pkgDep, 73
pkgDep2 (pkgDep), 73
pkgSnapshot, 74
postProcess, 49, 63, 75, 77, 81, 82, 85, 87, 88
postProcess.spatialObjects, 81
preProcess, 51, 79, 88
preProcess, 53, 80, 81, 84, 84, 85, 88
projectInputs, 51, 75–77, 81, 86
projectRaster, 51, 76, 87
raster, 81
reproducible (reproducible-package), 4
reproducible-package, 4
reproducibleOptions, 4, 88
Require, 89
retry, 91

search, 93
searchFull, 92
searchFullEx (searchFull), 92
shapefile, 51, 76, 81, 95
showCache, 42
showCache (clearCache), 35
showCache, ANY-method (clearCache), 35
splitTagsLocal, 38
st_write, 95
suffix (.prefix), 11

tempdir2, 93

unmakeMemoisable (makeMemoisable), 62
unrarPath, 93
untar, 81
unzip, 81

write.table, 34
writeFuture, 94
writeOutputs, 51, 75–77, 81, 94
writeRaster, 51, 76, 95