Package 'reproducible'

January 25, 2018

Type Package

Title A Set of Tools that Enhance Reproducibility Beyond Package Management

Description Built on top of 'git2r' and 'archivist', this package aims at making high-level, robust, machine and OS independent tools for making deeply reproducible and reusable content in R. This includes light weight package management (similar to 'packrat' and 'checkpoint', but more flexible, lightweight and simpler than both), plus it includes tools for caching, and accessing GitHub repositories.

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

Date 2018-01-24

Version 0.1.4

Depends R (>= 3.3.0)

Imports archivist (>= 2.1.2), crayon, data.table (>= 1.10.4), devtools, digest, fastdigest, git2r (>= 0.18), magrittr, memoise, methods, raster, Rcpp (>= 0.12.13), RCurl, sp, versions

Suggests hunspell, knitr, rgdal, rmarkdown, testthat

LinkingTo Rcpp

License GPL-3

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

ByteCompile yes

RoxygenNote 6.0.1

Collate 'RcppExports.R' 'cache-helpers.R' 'cache-tools.R' 'robustDigest.R' 'cache.R' 'consistentPaths.R' 'git.R' 'helpers.R' 'packages.R' 'pipe.R' 'reproducible-package.R' 'search.R'

NeedsCompilation yes

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

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

Repository CRAN

Date/Publication 2018-01-25 12:24:39 UTC

R topics documented:

reproducible-package *The* reproducible *package*

Description

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

Author(s)

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

Authors:

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

Other contributors:

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

See Also

Useful links:

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

.addTagsToOutput *Add tags to object*

Description

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

Usage

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

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

Arguments

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

Value

New object with tags attached.

Author(s)

Eliot McIntire

.cacheMessage *Create a custom cache message by class*

Description

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

Usage

```
.cacheMessage(object, functionName)

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

Arguments

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

Value

Nothing; called for its messaging side effect.

Author(s)

Eliot McIntire

Examples

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

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

Description

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

Usage

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

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

Arguments

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

Value

A character string with a path to a cache repository.

Author(s)

Eliot McIntire

Examples

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

.debugCache *Attach debug info to return for Cache*

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Eliot McIntire

.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

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)
repos	The remote repository (e.g., a CRAN mirror), passed to install.packages,
githubPkgs	Character vector of github repositoriess and packages, in the form repository/package@branch, with branch being optional.
githubPkgNames	Character vector of the package names, i.e., just the R package name.
nonLibPathPkgs	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
install_githubArgs	List of optional named arguments, passed to install_github
install.packagesArgs	List of optional named arguments, passed to install.packages
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.
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

```
## 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)

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

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

Arguments

object	Any R object.

Value

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

Author(s)

Eliot McIntire

Examples

```
a <- new.env()
a$b <- 1:10
object.size(a)
.objSizeInclEnviros(a) # much larger
```

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

Description

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

Usage

```
.preDigestByClass(object)

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

Arguments

`object`	Any R object.

Value

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

Author(s)

Eliot McIntire

Examples

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

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

Description

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

Usage

```
.prepareFileBackedRaster(obj, repoDir = NULL)
```

Arguments

`obj`	The raster object to save to the repository.
`repoDir`	Character denoting an existing directory in which an artifact will be saved.
`...`	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

```
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()
```

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

Description

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

Usage

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

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

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

Arguments

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

Value

The object, modified

Author(s)

Eliot McIntire

Examples

```
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())
```

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

Description

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

Usage

```
.sortDotsUnderscoreFirst(obj)

.orderDotsUnderscoreFirst(obj)
```

Arguments

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

Value

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

Author(s)

Eliot McIntire

Examples

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

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

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

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

Description

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

Usage

```
.tagsByClass(object)

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

Arguments

object	Any R object.

Value

A character vector of new tags.

Author(s)

Eliot McIntire

Examples

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

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

Description

Cache method that accommodates environments, S4 methods, Rasters

Usage

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

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

Arguments

FUN	Either a function or an unevaluated function call (e.g., using quote.
...	Arguments of FUN function .
notOlderThan	load an artifact from the database only if it was created after notOlderThan.
objects	Character vector of objects to be digested. This is only applicable if there is a list, environment or simList with named objects within it. Only this/these objects will be considered for caching, i.e., only use a subset of the list, environment or simList objects.
outputObjects	Optional character vector indicating which objects to return. This is only relevant for simList objects
algo	The algorithms to be used; currently available choices are md5, which is also the default, sha1, crc32, sha256, sha512, xxhash32, xxhash64 and murmur32.
cacheRepo	A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
compareRasterFileLength	Numeric. Optional. When there are Rasters, that have file-backed storage, this is passed to the length arg in digest when determining if the Raster file is already in the database. Note: uses digest for file-backed Raster. Default 1e6. Passed to .prepareFileBackedRaster.
userTags	A character vector with Tags. These Tags will be added to the repository along with the artifact.
digestPathContent	Logical. Should arguments that are of class Path (see examples below) have their name digested (FALSE), or their file contents (TRUE; default).
omitArgs	Optional character string of arguments in the FUN to omit from the digest.
classOptions	Optional list. This will pass into .robustDigest for specific classes. Should be options that the .robustDigest knows what to do with.
debugCache	Character or Logical. Either "complete" or "quick" (uses partial matching, so "c" or "q" work). TRUE is equivalent to "complete". If "complete", then the

returned object from the Cache function will have two attributes, debugCache1 and debugCache2, which are the entire `list(...)` and that same object, but after all `.robustDigest` calls, at the moment that it is digested using `fastdigest`, respectively. This `attr(mySimOut, "debugCache2")` can then be compared to a subsequent call and individual items within the object `attr(mySimOut, "debugCache1")` can be compared. If `"quick"`, then it will return the same two objects directly, without evalutating the `FUN(...)`.

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

Details

Caching R objects using `cache` has four important limitations:

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

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

1. converting any environments into list equivalents;
2. identifying the dispatched S4 method (including those made through inheritance) before hashing so the correct method is being cached;
3. by hashing the linked file, rather than the Raster object. Currently, only file-backed `Raster*` objects are digested (e.g., not `ff` objects, or any other R object where the data are on disk instead of in RAM);
4. using `fastdigest` internally when the object is in RAM, which can be up to ten times faster than `digest`. Note that file-backed objects are still hashed using `digest`.

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

`Cache` will add a tag to the artifact in the database called `accessed`, which will assign the time that it was accessed, either read or write. That way, artifacts can be shown (using `showCache`) or removed

(using clearCache) selectively, based on their access dates, rather than only by their creation dates. See example in clearCache. Cache (uppercase C) is used here so that it is not confused with, and does not mask, the archivist::cache function.

Value

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

Filepaths

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

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

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

Stochasticity

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

sideEffect

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

Note

As indicated above, several objects require pre-treatment before caching will work as expected. The function .robustDigest accommodates this. It is an S4 generic, meaning that developers can

produce their own methods for different classes of objects. Currently, there are methods for several types of classes. See .robustDigest.

See .robustDigest for other specifics for other classes.

Author(s)

Eliot McIntire

See Also

cache, .robustDigest

Examples

```
library(raster)

tmpDir <- file.path(tempdir(), "reproducible_examples", "Cache")
try(clearCache(tmpDir), 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 <- rnorm(10, 16) %>% Cache(cacheRepo = tmpDir) # recovers cached copy
ranNumsD <- Cache(quote(rnorm(n = 10, 16)), cacheRepo = tmpDir) # recovers cached copy

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

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

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

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

clearCache(tmpDir)

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

# access it again, from Cache
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
wholeCache <- showCache(tmpDir)

# keep only items accessed "recently" (i.e., only objectName:a)
```

```
onlyRecentlyAccessed <- showCache(tmpDir, userTags = max(wholeCache[tagKey == "accessed"]$tagValue))

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

clearCache(tmpDir)

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

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

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

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

showCache(tmpDir) ## empty

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

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

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

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

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

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

clearCache(tmpDir)

## Example 5: using caching to speed up rerunning expensive computations
ras <- raster(extent(0, 10, 0, 10), res = 1,
              vals = sample(1:5, replace = TRUE, size = 1e2),
              crs = "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84")

# A slow operation, like GIS operation
```

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

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

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

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

## Example 6: working with file paths

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

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

clearCache(tmpDir)

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

---

cache *Deprecated functions*

Description

Deprecated functions

Usage

```
cache(cacheRepo = NULL, FUN, ..., notOlderThan = NULL, objects = NULL,
```

```
  outputObjects = NULL, algo = "xxhash64")

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

Arguments

cacheRepo	A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
FUN	Either a function or an unevaluated function call (e.g., using quote.
...	Arguments of FUN function .
notOlderThan	load an artifact from the database only if it was created after notOlderThan.
objects	Character vector of objects to be digested. This is only applicable if there is a list, environment or simList with named objects within it. Only this/these objects will be considered for caching, i.e., only use a subset of the list, environment or simList objects.
outputObjects	Optional character vector indicating which objects to return. This is only relevant for simList objects
algo	The algorithms to be used; currently available choices are md5, which is also the default, sha1, crc32, sha256, sha512, xxhash32, xxhash64 and murmur32.

---

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

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Eliot McIntire and Alex Chubaty

Examples

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

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

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

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

  rm(localRepo, repo)

## End(Not run)
```

---

checkPath *Check filepath*

Description

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

Usage

```
checkPath(path, create)

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

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

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

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

Arguments

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

Value

Character string denoting the cleaned up filepath.

See Also

file.exists, dir.create.

Examples

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

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

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

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

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

unlink(tmpdir, recursive = TRUE)
```

clearCache *Examining and modifying the cache*

Description

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

Usage

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

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

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

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

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

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

Arguments

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

Details

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

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

`showCache` display the contents of the cache.

Value

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

See Also

`splitTagsLocal`.

Examples

```
library(raster)

tmpDir <- file.path(tempdir(), "reproducible_examples", "Cache")
try(clearCache(tmpDir), 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 <- rnorm(10, 16) %>% Cache(cacheRepo = tmpDir) # recovers cached copy
ranNumsD <- Cache(quote(rnorm(n = 10, 16)), cacheRepo = tmpDir) # recovers cached copy

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

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

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

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

clearCache(tmpDir)

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

# access it again, from Cache
```

```
ranNumsA <- Cache(rnorm, 4, cacheRepo = tmpDir, userTags = "objectName:a")
wholeCache <- showCache(tmpDir)

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

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

clearCache(tmpDir)

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

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

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

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

showCache(tmpDir) ## empty

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

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

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

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

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

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

clearCache(tmpDir)

## Example 5: using caching to speed up rerunning expensive computations
ras <- raster(extent(0, 10, 0, 10), res = 1,
```

```
              vals = sample(1:5, replace = TRUE, size = 1e2),
              crs = "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84")

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

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

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

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

## Example 6: working with file paths

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

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

clearCache(tmpDir)

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

`clearStubArtifacts` *Clear erroneous archivist artifacts*

Description

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

compareNA *NA-aware comparison of two vectors*

Description

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

Usage

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

Arguments

v1	A vector
v2	A vector

Examples

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

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

Description

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

Usage

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

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

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

## S4 method for signature 'environment'
```

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

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

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

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

Arguments

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

Author(s)

Eliot McIntire

See Also

`.robustDigest`

Examples

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

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

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

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

Description

This 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. This will generally copy a large file faster using `Robocopy` on Windows, and using `rsync` on macOS and Linux. In particular, 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

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

Arguments

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

Author(s)

Eliot McIntire and Alex Chubaty

Examples

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

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

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

`installedVersions` *Determine versions all installed pacakges*

Description

This code is adapted from `installed.versions`, but uses an `Rcpp` alternative to readLines for speed. 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)
```

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

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

`installVersions` *Install exact package versions from a package version text file & GitHub*

Description

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.

Usage

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

Arguments

gitHubPackages	Character vectors indicating repository/packageName@branch
packageVersionFile	Path to the package version file, defaults to the .packageVersions.txt.
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.

Details

Because of potential conflicts with loaded packages, this function will run install.packages in a separate R process.

Examples

```
## 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)
```

newLibPaths *A shortcut to create a .libPaths() with only 2 folders*

Description

This will remove all but the top level of .libPaths(), which should be the base packages installed with R, and adds a second directory, the `libPath`.

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`'
normPath(path)

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

Arguments

path	A character vector of filepaths.

Value

Character vector of cleaned up filepaths.

Examples

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

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

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

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

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

unlink(tmpdir, recursive = TRUE)
```

---

package_dependenciesMem

Memoised version of package_dependencies

Description

This have 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() (with the default NULL the results of this call) or data frame variants thereof. Alternatively, a package database like the one available from https://cran.r-project.org/web/packages/packages.rds.
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.

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

Description

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

Usage

```
asPath(obj)

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

Arguments

obj	A character string to convert to a Path.

Details

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

This 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 exanding of ~ vs. not, double back slash vs. single forward slash, relative path vs. absolute path. All of these should be assessed for their actual file or directory location, NOT their character string. By converting all character string that are actual file or directory paths with this function, then `Cache` will correctly assess the location, NOT the character string representation.

Examples

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

---

pipe *A cache-aware pipe that does not mask with %>%*

Description

This pipe can only be used at the start of a pipe chain, and must be preceeded by `Cache(...)` to allow other Cache arguments to be passed.

Usage

```
lhs %C% rhs
```

Arguments

`lhs`	A value or the magrittr placeholder.
`rhs`	A function call using the magrittr semantics.

Details

This will take the input arguments of the first function immediately following the `Cache() %C%` and the entire pipe chain code, 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. The entire chain must be identical, therefore. The required usage should be straight forward to insert into existing code that uses pipes (`Cache() %C% ... remaining pipes`.

See Also

pipe2

Examples

```
tmpdir <- file.path(tempdir(), "testCache")
checkPath(tmpdir, create = TRUE)
a <- rnorm(10, 16) %>%
     mean() %>%
     prod(., 6)
b <- Cache(cacheRepo = tmpdir) %C%
     rnorm(10, 16) %>%
     mean() %>%
     prod(., 6)
d <- Cache(cacheRepo = tmpdir) %C%
     rnorm(10, 16) %>%
     mean() %>%
     prod(., 6)
e <- Cache(cacheRepo = tmpdir) %C%
     rnorm(10, 16) %>%
     mean() %>%
     prod(., 5) # changed
all.equal(b,d) # TRUE
all.equal(a,d) # different because 'a' uses a unique rnorm, 'd' uses the Cached rnorm
               #   because the arguments to rnorm, i.e., 10 and 16, and
               #   the subsequent functions in the chain, are identical
all.equal(a,e) # different because the final function, prod, has a changed argument.

unlink(tmpdir, recursive = TRUE)
```

---

pipe2 *Pipe that is Cache-aware, being deprecated*

Description

This pipe will likely be deprecated, as it masks other pipes in the R ecosystem. This is fine, except to work, the reproducible package must be guaranteed to be first on the search path, which is almost impossible in any realistic project. Please see the `%C%` function ?pipe

Usage

```
lhs %>% rhs
```

Arguments

lhs	A value or the magrittr placeholder.
rhs	A function call using the magrittr semantics.

Details

A pipe that works with Cache. The code for this is based on a verbatim copy from https://github.com/tidyverse/magrittr/blob/master/R/pipe.R on Sep 8, 2017, based on commit 81c2e2410ebb7c560a2b4a8384ef5c20946373c3, with enhancements to make it Cache-aware. This version is a drop-in replacement for %>% and will work identically when there is no Cache. To use this, simply add %>% Cache() to a pipe sequence. This can be in the middle or at the end. See examples. It has been tested with multiple Cache calls within the same (long) pipe.

If there is a Cache in the pipe, the behaviour of the pipe is altered. In the magrittr pipe, each step of the pipe chain is evaluated one at a time, in sequence. This will not allow any useful type of caching. Here, if there is a call to Cache in the pipe sequence, the entire pipe chain before the call to Cache will have its arguments examined and digested. This digest is compared to the cache repository database. If there is an identical pipe sequence in the Cache respository, then it will return the final result of the entire pipe up to the Cache call. If there is no identical copy in the cache repository, then it will evaluate the pipe sequence as per normal, caching the return value at the point of the Cache call into the cache repository for later use.

See Also

pipe

Examples

```
tmpdir <- file.path(tempdir(), "testCache")
checkPath(tmpdir, create = TRUE)
a <- rnorm(10, 16) %>% mean() %>% prod(., 6)
b <- rnorm(10, 16) %>% mean() %>% prod(., 6) %>% Cache(cacheRepo = tmpdir)
d <- rnorm(10, 16) %>% mean() %>% prod(., 6) %>% Cache(cacheRepo = tmpdir)
all.equal(b,d) # TRUE
all.equal(a,d) # different because 'a' uses a unique rnorm, 'd' uses the Cached rnorm

# Can put Cache in the middle of a pipe -- f2 and f4 use "cached result" until Cache
f1 <- rnorm(10, 16) %>% mean() %>% prod(., runif(1)) %>% Cache(cacheRepo = tmpdir)
f2 <- rnorm(10, 16) %>% mean() %>% prod(., runif(1)) %>% Cache(cacheRepo = tmpdir)
f3 <- rnorm(10, 16) %>% mean() %>% Cache(cacheRepo = tmpdir) %>% prod(., runif(1))
f4 <- rnorm(10, 16) %>% mean() %>% Cache(cacheRepo = tmpdir) %>% prod(., runif(1))
all.equal(f1, f2) # TRUE because the runif is before the Cache
all.equal(f3, f4) # different because the runif is after the Cache

unlink(tmpdir, recursive = TRUE)
```

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 modfications for speed. It will first check local package directory(ies) in libPath, and it if the function cannont find the packages there, then it will use package_dependencies.

Usage

```
pkgDep(packages, libPath, recursive = TRUE, depends = TRUE,
  imports = TRUE, suggests = FALSE, linkingTo = TRUE,
  repos = getOption("repos"))
```

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.

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")
```

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)
```

readLinesRcpp *Alternative to readLines that is faster*

Description

This alternative is from https://gist.github.com/hadley/6353939

Usage

```
readLinesRcpp(path)
```

Arguments

path	Path to text file to read.

Value

Similar to readLines. It may not return identical results.

Examples

```
readLinesRcpp(system.file(package = "reproducible", "DESCRIPTION"))
```

readLinesRcppInternal *Alternative to readLines that is faster*

Description

This alternative is from https://gist.github.com/hadley/6353939

Usage

```
readLinesRcppInternal(path)
```

Arguments

path	Path to text file to read.

Value

Similar to readLines, except with explicit \n embedded.

Examples

```
readLinesRcpp(system.file(package = "reproducible", "DESCRIPTION"))
```

`Require` *Repeatability-safe install and load packages, optionally with specific versions*

Description

This is an "all in one" function that will run `install.packages` for CRAN packages, `devtools::install_github` for 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

```
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

```
## 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)
```

`searchFull` *Search up the full scope for functions*

Description

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

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

Usage

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

searchFullEx()
```

Arguments

`env`	The environment to start searching at. Default is calling environment, i.e., `parent.frame()`
`simplify`	Logical. Should the output be simplified to character, where possible

Value

Similar to `readLines`. It may not return identical results.

Examples

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

```
searchFull()

searchFullEx()
```

Index