

# Package ‘Require’

September 23, 2022

**Type** Package

**Title** Installing and Loading R Packages for Reproducible Workflows

**Description** A single key function, 'Require' that makes rerun-tolerant versions of 'install.packages' and 'require' for CRAN packages, packages no longer on CRAN (i.e., archived), specific versions of packages, and GitHub packages. This approach is developed to create reproducible workflows that are flexible and fast enough to use while in development stages, while able to build snapshots once a stable package collection is found. As with other functions in a reproducible workflow, this package emphasizes functions that return the same result whether it is the first or subsequent times running the function, with subsequent times being sufficiently fast that they can be run every time without undue waiting burden on the user or developer.

**URL** <https://Require.predictiveecology.org>,  
<https://github.com/PredictiveEcology/Require>

**Date** 2022-09-21

**Version** 0.1.2

**Depends** R (>= 4.0)

**Imports** data.table (>= 1.10.4), methods, utils

**Suggests** covr, remotes, pak, testit

**Encoding** UTF-8

**Language** en-CA

**License** GPL-3

**BugReports** <https://github.com/PredictiveEcology/Require/issues>

**ByteCompile** yes

**RoxygenNote** 7.2.1

**NeedsCompilation** no

**Author** Eliot J B McIntire [aut, cre] (<<https://orcid.org/0000-0002-6914-8316>>),  
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** 2022-09-23 11:30:02 UTC

## R topics documented:

Require-package	2
checkPath	8
DESCRIPTIONFileVersionV	9
detachAll	10
extractPkgName	11
getOptionRPackageCache	12
getPkgVersions	12
installGithubPackage	14
invertList	15
linkOrCopy	15
messageDF	16
modifyList2	16
normPath	17
paddedFloatToChar	18
parseGitHub	19
pkgDep	20
pkgDepIfDepRemoved	23
pkgSnapshot	24
rCurrentVersion	26
RequireCacheDir	27
RequireOptions	27
rversions	29
setLibPaths	29
setLinuxBinaryRepo	31
setup	31
tempdir2	32
tempfile2	33
trimVersionNumber	34
useLinuxSourceCache	34
<b>Index</b>	<b>36</b>

## Description

A single key function, 'Require' that makes rerun-tolerant versions of 'install.packages' and 'require' for CRAN packages, packages no longer on CRAN (i.e., archived), specific versions of packages, and GitHub packages. This approach is developed to create reproducible workflows that are flexible and fast enough to use while in development stages, while able to build snapshots once a stable package collection is found. As with other functions in a reproducible workflow, this package emphasizes functions that return the same result whether it is the first or subsequent times running the function, with subsequent times being sufficiently fast that they can be run every time without undue waiting burden on the user or developer.

This is an "all in one" function that will run `install.packages` for CRAN and GitHub <https://github.com/> packages and will install specific versions of each package if versions are specified either via an (in)equality (e.g., "`glue (>=1.6.2)`" or "`glue (==1.6.2)`" for an exact version) or with a `packageVersionFile`. If `require = TRUE`, the default, the function will then run `require` on all named packages that satisfy their version requirements. If packages are already installed (packages supplied), and their optional version numbers are satisfied, then the "install" component will be skipped.

## Usage

```
Require(
  packages,
  packageVersionFile,
  libPaths,
  install_githubArgs = list(),
  install_packagesArgs = list(),
  standAlone = getOption("Require.standAlone", FALSE),
  install = getOption("Require.install", TRUE),
  require = getOption("Require.require", TRUE),
  repos = getOption("repos"),
  purge = getOption("Require.purge", FALSE),
  verbose = getOption("Require.verbose", FALSE),
  ...
)
```

## Arguments

- |                                 |  |
|---------------------------------|--|
| <code>packages</code>           | 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> ). In the case of a GitHub package, it will be assumed that the name of the repository is the name of the package. If this is not the case, then pass a named character vector here, where the names are the package names that could be different than the GitHub repository name. |
| <code>packageVersionFile</code> | If provided, then this will override all <code>install.package</code> calls with <code>versions::install.versions</code>   |
| <code>libPaths</code>           | The library path (or libraries) where all packages should be installed, and looked for to load (i.e., call <code>library</code> ). This can be used to create isolated, stand alone package installations, if used with <code>standAlone = TRUE</code> . Currently, the path supplied here will be prepended to <code>.libPaths()</code> (temporarily during this call)  |

	to Require if <code>standAlone = FALSE</code> or will set (temporarily) <code>.libPaths()</code> to <code>c(libPaths, tail(libPaths(), 1)</code> to keep base packages.
<code>install_githubArgs</code>	List of optional named arguments, passed to <code>install.packages</code> inside <code>installGitHubPackage</code> .
<code>install.packagesArgs</code>	List of optional named arguments, passed to <code>install.packages</code> .
<code>standAlone</code>	Logical. If TRUE, all packages will be installed to and loaded from the <code>libPaths</code> only. NOTE: If TRUE, THIS WILL CHANGE THE USER'S <code>.libPaths()</code> , similar to e.g., the <code>checkpoint</code> package. If FALSE, then <code>libPath</code> will be prepended to <code>.libPaths()</code> during the Require call, resulting in shared packages, i.e., it will include the user's default package folder(s). This can be create dramatically faster installs if the user has a substantial number of the packages already in their personal library. Default FALSE to minimize package installing.
<code>install</code>	Logical or "force". If FALSE, this will not try to install anything. If "force", then it will force installation of requested packages, mimicking a call to e.g., <code>install.packages</code> . If TRUE, the default, then this function will try to install any missing packages or dependencies.
<code>require</code>	Logical or character string. If TRUE, the default, then the function will attempt to call <code>require</code> on all requested packages, possibly after they are installed. If a character string, then it will only call <code>require</code> on those specific packages (i.e., it will install the ones listed in <code>packages</code> , but load the packages listed in <code>require</code> )
<code>repos</code>	The remote repository (e.g., a CRAN mirror), passed to either <code>install.packages</code> , <code>install_github</code> or <code>installVersions</code> .
<code>purge</code>	Logical. Should all caches be purged? Default is <code>getOption("Require.purge", FALSE)</code> . There is a lot of internal caching of results throughout the Require package. These help with speed and reduce calls to internet sources. However, sometimes these caches must be purged. The cached values are renewed when found to be too old, with the age limit. This maximum age can be set in seconds with the environment variable <code>R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE</code> , or if unset, defaults to 3600 (one hour – see <a href="#">utils::available.packages</a> ). Internally, there are calls to <code>available.packages</code> .
<code>verbose</code>	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.
<code>...</code>	Passed to <code>install.packages</code> . Good candidates are e.g., <code>type</code> or <code>dependencies</code> . This can be used with <code>install_githubArgs</code> or <code>install.packageArgs</code> which give individual options for those 2 internal function calls.

## Details

`standAlone` will either put the Required packages and their dependencies *all* within the `libPaths` (if TRUE) or if FALSE will only install packages and their dependencies that are otherwise not installed in `.libPaths()[1]`, i.e., the current active R package directory. Any packages or dependencies that are not yet installed will be installed in `libPaths`.

## GitHub Package

Follows `remotes::install_github` standard. As with `remotes::install_github`, it is not possible to specify a past version of a GitHub package unless that version is a tag or the user passes the SHA that had that package version. Similarly, if a developer does a local install e.g., via `pkgload::install`, of an active project, this package will not be able know of the GitHub state, and thus `pkgSnapshot` will not be able to recover this state as there is no SHA associated with a local installation. Use `Require` (or `remotes::install_github`) to create a record of the GitHub state.

## Package Snapshots

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

## Mutual Dependencies

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 addressed (if there are no conflicts), allowing a call to `pkgSnapshot()` to take a snapshot or "record" of the current collection of packages and versions.

## Local Cache of Packages

When installing new packages, `Require` will put all source and binary files in an R-version specific subfolder of `getOption("Require.RPackageCache")` whose default is `RPackageCache()`, meaning *cache packages locally in a project-independent location*, and will reuse them if needed. To turn off this feature, set `options("Require.RPackageCache" = FALSE)`.

## Note

For advanced use and diagnosis, the user can set `verbose = TRUE` or 1 or 2 (or via `options("Require.verbose")`). This will attach an attribute `attr(obj, "Require")` to the output of this function.

## Author(s)

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

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://Require.predictiveecology.org>
- <https://github.com/PredictiveEcology/Require>
- Report bugs at <https://github.com/PredictiveEcology/Require/issues>

**Examples**

```

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

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

# make a package version snapshot of installed packages
packageVersionFile <- "_packageVersionTest.txt"
(pkgSnapshot(libPath = tempPkgFolder, packageVersionFile, standAlone = TRUE))

# Restart R -- to remove the old temp folder (it disappears with restarting R)
library(Require)
tempPkgFolder <- file.path(tempdir(), "Packages")
packageVersionFile <- "_packageVersionTest.txt"
# Reinstall and reload the exact version from previous
Require(packageVersionFile = packageVersionFile, libPaths = tempPkgFolder, standAlone = TRUE)

# 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)
newTempPkgFolder <- file.path(tempdir(), "Packages2")

# Note this will install the 1.3.2 version (older than current on CRAN), but
# because crayon is still loaded in memory, it will return TRUE, using the current version
# of crayon. To start using the older 1.3.2, need to unload or restart R
Require("crayon",
  packageVersionFile = packageVersionFile,
  libPaths = newTempPkgFolder, standAlone = TRUE
)

# restart R again to get access to older version
# run again, this time, correct "older" version installs in place of newer one
library(Require)
packageVersionFile <- "_packageVersionTest.txt"
newTempPkgFolder <- file.path(tempdir(), "Packages3")
Require("crayon",
  packageVersionFile = packageVersionFile,
  libPaths = newTempPkgFolder, standAlone = TRUE
)

# Mutual dependencies, only installs once -- e.g., httr
tempPkgFolder <- file.path(tempdir(), "Packages")
Require(c("cranlogs", "covr"), libPaths = tempPkgFolder, standAlone = TRUE)

```

```
#####
```

```

# Isolated projects -- Just use a project folder and pass to libPaths or set .libPaths() #
#####
# GitHub packages -- restart R because crayon is needed
library(Require)
ProjectPackageFolder <- file.path(tempdir(), "ProjectA")
# THIS ONE IS LARGE -- > 100 dependencies -- use standAlone = FALSE to
# reuse already installed packages --> this won't allow as much control
# of package versioning
Require("PredictiveEcology/SpaDES@development",
  libPaths = ProjectPackageFolder, standAlone = FALSE
)

# To keep totally isolated: use standAlone = TRUE
# --> setting .libPaths() directly means standAlone is not necessary; it will only
# use .libPaths()
library(Require)
ProjectPackageFolder <- file.path("~", "ProjectA")
setLibPaths(ProjectPackageFolder)
Require("PredictiveEcology/SpaDES@development") # the latest version on GitHub
Require("PredictiveEcology/SpaDES@23002b2a92a92df4ccba7f51cdd82798800b2fa7")
# a specific commit (by using the SHA)

#####
# Mixing and matching GitHub, CRAN, with and without version numbering
#####
# Restart R -- when installing/loading packages, start fresh
pkgs <- c(
  "glue (<=1.0.4)", "digest (<= 0.6.28)", "glm (<=1.3.0)",
  "achubaty/amc@development", "PredictiveEcology/LandR@development (>=0.0.1)",
  "PredictiveEcology/LandR@development (>=0.0.2)", "ianmseddy/LandR.CS (<=0.0.1)"
)
Require::Require(pkgs)

#####
# Using libPaths -- This will only be used inside this function;
# To change .libPaths() for the whole session use a manually call to
# setLibPaths(newPath) first
#####
Require::Require("SpaDES", libPaths = "~/TempLib2", standAlone = FALSE)

#####
# Persistent separate packages
#####
setLibPaths("~/TempLib2", standAlone = TRUE)
Require::Require("SpaDES") # not necessary to specify standAlone here because .libPaths are set

#####
# Installing on many machines that are connected by a shared drive
#####
options("Require.RPackageCache" = TRUE) # will binaries on the fly.
# Put thes in a shared location.
# May need to install Require in main user library before setting library paths for project

```

```

if (!require("Require")) install.packages("Require")
setLibPaths("./packages") # not shared location for library path; no longer using main user lib
Require::Require(
  packageVersionFile = "./packageVersions.txt",
  standAlone = TRUE
)

## 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 does not 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\(\)](#). To normalize a path to a file, use [normPath\(\)](#) or [normalizePath\(\)](#).



**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) # clean up
```

---

DESCRIPTIONFileVersionV

*GitHub package tools*

---

**Description**

A series of helpers to access and deal with GitHub packages

**Usage**

```
DESCRIPTIONFileVersionV(file, purge = getOption("Require.purge", FALSE))

DESCRIPTIONFileOtherV(file, other = "RemoteSha")

getGitHubDESCRIPTION(
  pkg,
  purge = getOption("Require.purge", FALSE),
  verbose = getOption("Require.verbose")
)
```

**Arguments**

file	A file path to a DESCRIPTION file
purge	Logical. Should all caches be purged? Default is <code>getOption("Require.purge", FALSE)</code> . There is a lot of internal caching of results throughout the Require package. These help with speed and reduce calls to internet sources. However, sometimes these caches must be purged. The cached values are renewed when found to be too old, with the age limit. This maximum age can be set in seconds with the environment variable <code>R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE</code> , or if unset, defaults to 3600 (one hour – see <code>utils::available.packages</code> ). Internally, there are calls to <code>available.packages</code> .
other	Any other keyword in a DESCRIPTION file that precedes a ":". The rest of the line will be retrieved.
pkg	A character string with a GitHub package specification (c.f. remotes)
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.

**Details**

`getGitHubDESCRIPTION` retrieves the DESCRIPTION file from GitHub.com

---

<code>detachAll</code>	<i>Detach and unload all packages</i>
------------------------	---------------------------------------

---

**Description**

This uses `pkgDepTopoSort` internally so that the package dependency tree is determined, and then packages are unloaded in the reverse order. Some packages don't unload successfully for a variety of reasons. Several known packages that have this problem are identified internally and *not* unloaded. Currently, these are `glue`, `rlang`, `ps`, `ellipsis`, and, `processx`.

**Usage**

```
detachAll(
  pkgs,
  dontTry = NULL,
  doSort = TRUE,
  verbose = getOption("Require.verbose")
)
```

**Arguments**

pkgs	A character vector of packages to detach. Will be topologically sorted unless <code>doSort</code> is FALSE.
------	---

dontTry	A character vector of packages to not try. This can be used by a user if they find a package fails in attempts to unload it, e.g., "ps"
doSort	If TRUE (the default), then the pkgs will be topologically sorted. If FALSE, then it won't. Useful if the pkgs are already sorted.
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.

**Value**

A numeric named vector, with names of the packages that were attempted. 2 means the package was successfully unloaded, 1 it was tried, but failed, 3 it was in the search path and was detached and unloaded.

---

extractPkgName	<i>Extract info from package character strings</i>
----------------	--

---

**Description**

Cleans a character vector of non-package name related information (e.g., version)

**Usage**

```
extractPkgName(pkgs)
extractVersionNumber(pkgs)
extractInequality(pkgs)
extractPkgGitHub(pkgs)
```

**Arguments**

pkgs            A character string vector of packages with or without GitHub path or versions

**Value**

Just the package names without extraneous info.

**See Also**

[trimVersionNumber\(\)](#)

**Examples**

```
extractPkgName("Require (>=0.0.1)")
extractVersionNumber(c("Require (<=0.0.1)", "PredictiveEcology/Require@development (<=0.0.4)"))
extractInequality("Require (<=0.0.1)")
extractPkgGitHub("PredictiveEcology/Require")
```

---

```
getOptionRPackageCache
```

*Get the option for Require.RPackageCache*

---

### Description

First checks if an environment variable `Require.RPackageCache` is set and defines a path. If not set, checks whether the options(`"Require.RPackageCache"`) is set. If a character string, then it returns that. If `TRUE`, then use `RequirePkgCacheDir()`. If `FALSE` then returns `NULL`.

### Usage

```
getOptionRPackageCache()
```

---

```
getPkgVersions
```

*Internals used by Require*

---

### Description

While these are not intended to be called manually by users, they may be of some use for advanced users.

### Usage

```
getPkgVersions(pkgDT, install = TRUE)

getAvailable(
  pkgDT,
  purge = FALSE,
  repos = getOption("repos"),
  verbose = getOption("Require.verbose")
)

installFrom(
  pkgDT,
  purge = FALSE,
  repos = getOption("repos"),
  verbose = getOption("Require.verbose")
)

doInstalls(
  pkgDT,
  install_githubArgs,
  install_packagesArgs,
  install = TRUE,
```

```

    repos = getOption("repos"),
    verbose = getOption("Require.verbose"),
    ...
)

doLoading(pkgDT, require = TRUE, verbose = getOption("Require.verbose"), ...)

archiveVersionsAvailable(package, repos)

```

## Arguments

pkgDT	A character string with full package names or a <code>data.table</code> with at least 2 columns "Package" and "packageFullName".
install	Logical or "force". If FALSE, this will not try to install anything. If "force", then it will force installation of requested packages, mimicking a call to <code>e.g., install.packages</code> . If TRUE, the default, then this function will try to install any missing packages or dependencies.
purge	Logical. Should all caches be purged? Default is <code>getOption("Require.purge", FALSE)</code> . There is a lot of internal caching of results throughout the Require package. These help with speed and reduce calls to internet sources. However, sometimes these caches must be purged. The cached values are renewed when found to be too old, with the age limit. This maximum age can be set in seconds with the environment variable <code>R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE</code> , or if unset, defaults to 3600 (one hour – see <code>utils::available.packages</code> ). Internally, there are calls to <code>available.packages</code> .
repos	The remote repository (e.g., a CRAN mirror), passed to either <code>install.packages</code> , <code>install_github</code> or <code>installVersions</code> .
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.
install_githubArgs	List of optional named arguments, passed to <code>install.packages</code> inside <code>installGitHubPackage</code> .
install.packagesArgs	List of optional named arguments, passed to <code>install.packages</code> .
...	Passed to <code>install.packages</code> . Good candidates are e.g., <code>type</code> or <code>dependencies</code> . This can be used with <code>install_githubArgs</code> or <code>install.packageArgs</code> which give individual options for those 2 internal function calls.
require	Logical or character string. If TRUE, the default, then the function will attempt to call <code>require</code> on all requested packages, possibly after they are installed. If a character string, then it will only call <code>require</code> on those specific packages (i.e., it will install the ones listed in <code>packages</code> , but load the packages listed in <code>require</code> )
package	A single package name (without version or github specifications)

## Details

`doInstall` is a wrapper around `utils::install.packages`, `installGithub`, and `installCRAN`, and `installArchive`

doLoading is a wrapper around require.

archiveVersionsAvailable searches CRAN Archives for available versions. It has been borrowed from a sub-set of the code in a non-exported function: `remotes:::download_version_url`

## Value

In general, these functions return a `data.table` with various package information, installation status, version, available version etc.

---

`installGithubPackage` *Install R Package from GitHub source code*

---

## Description

A lightweight alternative to `devtools::install_github`. All dependencies must have been installed already for this to work.

## Usage

```
installGithubPackage(
  gitRepo,
  libPath = .libPaths()[1],
  verbose = getOption("Require.verbose"),
  ...
)
```

```
installGitHubPackage(
  gitRepo,
  libPath = .libPaths()[1],
  verbose = getOption("Require.verbose"),
  ...
)
```

## Arguments

<code>gitRepo</code>	A repository in the form: <code>Account/Repository@Branch</code> or <code>Account/Repository@SHA</code>
<code>libPath</code>	The folder where you would like the package installed. Defaults to <code>.libPaths()[1]</code>
<code>verbose</code>	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.
<code>...</code>	Passed to R CMD INSTALL

---

invertList	<i>Invert a 2-level list</i>
------------	------------------------------

---

**Description**

This is a simple version of `purrr::transpose`, only for lists with 2 levels.

**Usage**

```
invertList(l)
```

**Arguments**

l                    A list with 2 levels. If some levels are absent, they will be NULL

**Value**

A list with 2 levels deep, inverted from l

**Examples**

```
# create a 2-deep, 2 levels in first, 3 levels in second
a <- list(a = list(d = 1, e = 2:3, f = 4:6), b = list(d = 5, e = 55))
invertList(a) # creates 2-deep, now 3 levels outer --> 2 levels inner
```

---

linkOrCopy	<i>Create link to file, falling back to making a copy if linking fails.</i>
------------	---

---

**Description**

First try to create a hardlink to the file. If that fails, try a symbolic link (`symlink`) before falling back to copying the file. "File" here can mean a file or a directory.

**Usage**

```
linkOrCopy(from, to, allowSymlink = FALSE)
```

**Arguments**

from, to            character vectors, containing file names or paths.  
allowSymlink        Logical. If FALSE, the default, then it will try `file.link` first, then `file.copy`, omitting the `file.symlink` step

---

messageDF	<i>Use message to print a clean square data structure</i>
-----------	---

---

### Description

Sends to message, but in a structured way so that a data.frame-like can be cleanly sent to messaging. This will only show a message if the value of verbose is greater than the verboseLevel.

### Usage

```
messageDF(df, round, verbose = getOption("Require.verbose"), verboseLevel = 1)
messageVerbose(..., verbose = getOption("Require.verbose"), verboseLevel = 1)
```

### Arguments

df	A data.frame, data.table, matrix
round	An optional numeric to pass to round
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.
verboseLevel	A numeric indicating what verbose threshold (level) above which this message will show.
...	Passed to install.packages. Good candidates are e.g., type or dependencies. This can be used with install_githubArgs or install_packageArgs which give individual options for those 2 internal function calls.

---

modifyList2	<i>modifyList for multiple lists</i>
-------------	--------------------------------------

---

### Description

This calls `utils::modifyList` iteratively using `base::Reduce`, so it can handle >2 lists. The subsequent list elements that share a name will override previous list elements with that same name. It also will handle the case where any list is a NULL

### Usage

```
modifyList2(...)
```

### Arguments

...	One or more named lists.
-----	--------------------------



**Details**

Simply a convenience around Reduce(modifyList, list(...)), with some checks.

**Examples**

```
modifyList2(list(a = 1), list(a = 2, b = 2))
modifyList2(list(a = 1), NULL, list(a = 2, b = 2))
modifyList2(list(a = 1), NULL, list(a = 2, b = 2), list(a = 3, c = list(1:10)))
```

---

normPath	<i>Normalize filepath</i>
----------	---------------------------

---

**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()

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

**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) # clean up
```

---

paddedFloatToChar      *Convert numeric to character with padding*

---

**Description**

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

**Usage**

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

**Arguments**

<code>x</code>	numeric. Number to be converted to character with padding
<code>padL</code>	numeric. Desired number of digits on left side of decimal. If not enough, pad will be used to pad.
<code>padR</code>	numeric. Desired number of digits on right side of decimal. If not enough, pad will be used to pad.
<code>pad</code>	character to use as padding ( <code>nchar(pad) == 1</code> must be TRUE). Currently, can be only "0" or " " (i.e., space).

**Value**

Character string representing the filename.

**Author(s)**

Eliot McIntire and Alex Chubaty

**Examples**

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

---

parseGitHub

*GitHub specific helpers*

---

**Description**

installGitHub is a vectorized installGithubPackages. This will attempt to identify all dependencies of all supplied packages first, then load the packages in the correct order so that each of their dependencies are met before each is installed.

**Usage**

```
parseGitHub(pkgDT, verbose = getOption("Require.verbose"))

installGitHub(
  pkgDT,
  toInstall,
  install_githubArgs = list(),
  dots = dots,
  verbose = getOption("Require.verbose")
)
```

**Arguments**

pkgDT	A character string with full package names or a data.table with at least 2 columns "Package" and "packageFullName".
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.
toInstall	DESCRIPTION NEEDED
install_githubArgs	Any arguments passed to install_github
dots	A list of ..., e.g., list(...). Only for internal use.

**Details**

parseGitHub turns the single character string representation into 3 or 4: Account, Repo, Branch, SubFolder.

**Value**

parseGitHub returns a `data.table` with added columns.

installGitHub returns a named character vector indicating packages successfully installed, unless the word "Failed" is returned, indicating installation failure. The names will be the full GitHub package name, as provided to `gitPkgNames` in the function call.

**Examples**

```
## Not run:
  installGitHub(c("PredictiveEcology/Require", "PredictiveEcology/quickPlot"))

## End(Not run)
```

---

pkgDep

*Determine package dependencies*

---

**Description**

This will first look in local filesystem (in `.libPaths()`) and will use a local package to find its dependencies. If the package does not exist locally, including whether it is the correct version, then it will look in (currently) CRAN and its archives (if the current CRAN version is not the desired version to check). It will also look on GitHub if the package description is of the form of a GitHub package with format `account/repo@branch` or `account/repo@commit`. For this, it will attempt to get package dependencies from the GitHub 'DESCRIPTION' file. This is intended to replace `tools::package_dependencies` or `pkgDep` in the **miniCRAN** package, but with modifications to allow multiple sources to be searched in the same function call.

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

This is a wrapper around `tools::dependsOnPkgs`, but with the added option of `sorted`, which will sort them such that the packages at the top will have the least number of dependencies that are in `pkgs`. This is essentially a topological sort, but it is done heuristically. This can be used to e.g., detach or unloadNamespace packages in order so that they each of their dependencies are detached or unloaded first.

**Usage**

```
pkgDep(
  packages,
  libPath = .libPaths(),
  which = c("Depends", "Imports", "LinkingTo"),
```

```

    recursive = FALSE,
    depends,
    imports,
    suggests,
    linkingTo,
    repos = getOption("repos"),
    keepVersionNumber = TRUE,
    includeBase = FALSE,
    sort = TRUE,
    purge = getOption("Require.purge", FALSE),
    verbose = getOption("Require.verbose")
)

pkgDep2(
  packages,
  recursive = TRUE,
  which = c("Depends", "Imports", "LinkingTo"),
  depends,
  imports,
  suggests,
  linkingTo,
  repos = getOption("repos"),
  sorted = TRUE,
  purge = getOption("Require.purge", FALSE)
)

pkgDepTopoSort(
  pkgs,
  deps,
  reverse = FALSE,
  topoSort = TRUE,
  useAllInSearch = FALSE,
  returnFull = TRUE,
  recursive = TRUE,
  purge = getOption("Require.purge", FALSE),
  verbose = getOption("Require.verbose")
)

```

### Arguments

packages	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> ). In the case of a GitHub package, it will be assumed that the name of the repository is the name of the package. If this is not the case, then pass a named character vector here, where the names are the package names that could be different than the GitHub repository name.
libPath	A path to search for installed packages. Defaults to <code>.libPaths()</code>
which	a character vector listing the types of dependencies, a subset of <code>c("Depends",</code>

	"Imports", "LinkingTo", "Suggests", "Enhances"). Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances".
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 <code>install.packages</code> , <code>install_github</code> or <code>installVersions</code> .
keepVersionNumber	Logical. If TRUE, then the package dependencies returned will include version number. Default is FALSE
includeBase	Logical. Should R base packages be included, specifically, those in <code>tail(.libPath(), 1)</code>
sort	Logical. If TRUE, the default, then the packages will be sorted alphabetically. If FALSE, the packages will not have a discernible order as they will be a concatenation of the possibly recursive package dependencies.
purge	Logical. Should all caches be purged? Default is <code>getOption("Require.purge", FALSE)</code> . There is a lot of internal caching of results throughout the Require package. These help with speed and reduce calls to internet sources. However, sometimes these caches must be purged. The cached values are renewed when found to be too old, with the age limit. This maximum age can be set in seconds with the environment variable <code>R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE</code> , or if unset, defaults to 3600 (one hour – see <code>utils::available.packages</code> ). Internally, there are calls to <code>available.packages</code> .
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.
sorted	Logical. If TRUE, the default, the packages will be sorted in the returned list from most number of dependencies to least.
pkgs	A vector of package names to evaluate their reverse depends (i.e., the packages that <i>use</i> each of these packages)
deps	An optional named list of (reverse) dependencies. If not supplied, then <code>tools::dependsOnPkgs(..., recursive = TRUE)</code> will be used
reverse	Logical. If TRUE, then this will use <code>tools::pkgDependsOn</code> to determine which packages depend on the pkgs
topoSort	Logical. If TRUE, the default, then the returned list of packages will be in order with the least number of dependencies listed in pkgs at the top of the list.
useAllInSearch	Logical. If TRUE, then all non-core R packages in <code>search()</code> will be appended to pkgs to allow those to also be identified

`returnFull` Logical. Primarily useful when `reverse = TRUE`. If `TRUE`, then all installed packages will be searched. If `FALSE`, the default, only packages that are currently in the `search()` path and passed in `pkgs` will be included in the possible reverse dependencies.

### Value

A possibly ordered, named (with packages as names) list where list elements are either full reverse depends.

### Note

`tools::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

```
## Not run:
pkgDep("Require")
pkgDep("Require", keepVersionNumber = FALSE) # just names
pkgDep("PredictiveEcology/reproducible") # GitHub
pkgDep("PredictiveEcology/reproducible", recursive = TRUE) # GitHub
pkgDep(c("PredictiveEcology/reproducible", "Require")) # GitHub package and local packages
pkgDep(c("PredictiveEcology/reproducible", "Require", "plyr")) # GitHub, local, and CRAN packages

## End(Not run)
## Not run:
pkgDep2("Require")
# much bigger one
pkgDep2("reproducible")

## End(Not run)
## Not run:
pkgDepTopoSort(c("Require", "data.table"), reverse = TRUE)

## End(Not run)
```

---

`pkgDepIfDepRemoved`      *Package dependencies when one or more packages removed*

---

### Description

This is primarily for package developers. It allows the testing of what the recursive dependencies would be if a package was removed from the immediate dependencies.

**Usage**

```
pkgDepIfDepRemoved(
  pkg = character(),
  depsRemoved = character(),
  verbose = getOption()
)
```

**Arguments**

pkg	A package name to be testing the dependencies
depsRemoved	A vector of package names who are to be "removed" from the pkg immediate dependencies
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.

**Value**

A list with 3 named lists `Direct`, `Recursive` and `IfRemoved`. `Direct` will show the top level direct dependencies, either `Remaining` or `Removed`. `Recursive` will show the full recursive dependencies, either `Remaining` or `Removed`. `IfRemoved` returns all package dependencies that are removed for each top level dependency. If a top level dependency is not listed in this final list, then it means that it is also a recursive dependency elsewhere, so its removal has no effect.

**Examples**

```
## Not run:
pkgDepIfDepRemoved("Require", "remotes")

## End(Not run)
```

---

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 = "packageVersions.txt",
  libPaths,
  standAlone = FALSE,
  purge = getOption("Require.purge", FALSE),
  exact = TRUE,
  verbose = getOption("Require.verbose")
)
```



**Arguments**

packageVersionFile	A filename to save the packages and their currently installed version numbers. Defaults to "packageVersions.txt". If this is specified to be NULL, the function will return the exact Require call needed to install all the packages at their current versions. This can be useful to add to a script to allow for reproducibility of a script.
libPaths	The path to the local library where packages are installed. Defaults to the <code>.libPaths()[1]</code> .
standAlone	Logical. If TRUE, all packages will be installed to and loaded from the <code>libPaths</code> only. NOTE: If TRUE, THIS WILL CHANGE THE USER'S <code>.libPaths()</code> , similar to e.g., the checkpoint package. If FALSE, then <code>libPath</code> will be prepended to <code>.libPaths()</code> during the Require call, resulting in shared packages, i.e., it will include the user's default package folder(s). This can be create dramatically faster installs if the user has a substantial number of the packages already in their personal library. Default FALSE to minimize package installing.
purge	Logical. Should all caches be purged? Default is <code>getOption("Require.purge", FALSE)</code> . There is a lot of internal caching of results throughout the Require package. These help with speed and reduce calls to internet sources. However, sometimes these caches must be purged. The cached values are renewed when found to be too old, with the age limit. This maximum age can be set in seconds with the environment variable <code>R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE</code> , or if unset, defaults to 3600 (one hour – see <a href="#">utils::available.packages</a> ). Internally, there are calls to <code>available.packages</code> .
exact	Logical. If TRUE, the default, then for GitHub packages, it will install the exact SHA, rather than the head of the <code>account/repo@branch</code> . For CRAN packages, it will install the exact version. If FALSE, then GitHub packages will identify their branch if that had been specified upon installation, not a SHA. If the package had been installed with reference to a SHA, then it will return the SHA as it does not know what branch it came from. Similarly, CRAN packages will report their version and specify with a <code>&gt;=</code> , allowing a subsequent user to install with a minimum version number, as opposed to an exact version number.
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.

**Details**

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

**Examples**

```
tmpdir <- checkPath(file.path(tmpdir(), "example_pkgSnapshot"), create = TRUE)
pkgSnapFile <- tempfile(tmpdir = tmpdir)
pkgSnapshot(pkgSnapFile, .libPaths()[1])
data.table::fread(pkgSnapFile)
```

```

## Not run:
# An example to move this file to a new computer
library(Require)
setLibPaths(.libPaths()[1]) # this will only do a snapshot of the main user library
fileName <- "packageSnapshot.txt"
pkgSnapshot(fileName)
# Get file on another computer -- via email, slack, cloud, etc.
# library(googledrive)
# (out <- googledrive::drive_upload(fileName)) # copy the file id to clipboard

# On new machine
fileName <- "packageSnapshot.txt"
library(Require)
# get the file from email, slack, cloud etc.
# library(googledrive)
# drive_download(as_id(PASTE-THE-FILE-ID-HERE), path = fileName)
setLibPaths("~/Rpackages") # start with an empty folder for new
# library to minimize package version conflicts
Require(packageVersionFile = fileName)

# Passing NULL --> results in output to console with exact Require call to
# achieve the packages installations
pkgSnapshot(NULL, libPaths = .libPaths()[1], exact = FALSE)

# Or shunt it to a file
sink("packages2.R")
pkgSnapshot(NULL, libPaths = .libPaths()[1])
sink()

# Will show "minimum package version"
pkgSnapshot(NULL, libPaths = .libPaths()[1], exact = FALSE)

## End(Not run)

# cleanup
unlink(tmpdir, recursive = TRUE)

```

---

rCurrentVersion

*Get or compare current R version to a known version*


---

## Description

Compares R version to a known version

## Usage

```
rCurrentVersion(testVers)
```

**Arguments**

testVers            A character string using format "==" 4.1" or ">= 4.1"

**Value**

If no testVers is supplied, then it will just return the current R version. If testVers is supplied, then it will return a TRUE or FALSE.

**Examples**

```
rCurrentVersion(">= 4.1")
```

---

RequireCacheDir	<i>Path to (package) cache directory</i>
-----------------	--

---

**Description**

Path to (package) cache directory

**Usage**

```
RequireCacheDir()
```

```
RequirePkgCacheDir()
```

---

RequireOptions	Require <i>options</i>
----------------	------------------------

---

**Description**

These provide top-level, powerful settings for a comprehensive reproducible workflow. See Details below.

**Usage**

```
RequireOptions()
```

```
getRequireOptions()
```

## Details

`RequireOptions()` prints the default values of package options set at startup, which may have been changed (e.g., by the user) during the current session.

`getRequireOptions()` prints the current values of package options.

Below are options that can be set with `options("Require.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:

`RPackageCache` Default: `getOptionRPackageCache()`, which must be either a path or a logical. This can be set using an environment variable e.g. `Sys.setenv(Require.RPackageCache = "somePath")`, or `Sys.setenv(Require.RPackageCache = "TRUE")`; if that is not set, then an either a path or logical option (`options(Require.RPackageCache = "somePath")` or `options(Require.RPackageCache = TRUE)`). If `TRUE`, the default folder location `RequirePkgCacheDir()` will be used. If this is `TRUE` or a path is provided, then binary and source packages will be cached here. Subsequent downloads of same package will use local copy. Default is to have packages not be cached locally so each install of the same version will be from the original source, e.g., CRAN, GitHub.

`buildBinaries` Default: `TRUE`. Only relevant on `*nix` systems and if `getOption("Require.RPackageCache")` is set to a path. If `TRUE` or a valid path, then `Require` will pass `INSTALL_OPTS = "--build"`, meaning the package binary will be built and then saved in the `getOption("Require.RPackageCache")`. This means that subsequent installs of this package on this or identical system will be faster.

`persistentPkgEnv` Default: `FALSE`. (ADVANCED USE) `Require` stashes a lot of information in a hidden environment, located at `Require:::pkgEnv`. This gets reset at each restart of R and each reload of `Require`. To make the stashes more persistent, set this option to `TRUE`. A file will be placed at `file.path("~/", "_Require_pkgEnv.rdata")`, which will be restored at package load

`purge` Default: `FALSE`. If set to (almost) all internal caches used by `Require` will be deleted and rebuilt. This should not generally be necessary as it will automatically be deleted after (by default) 1 hour (set via `R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE` environment variable in seconds)

`setupVerbose` Default: `TRUE`. Logical. Once `setup` is called, there are several important changes that are made to the user's experience. For beginners with `Require`, the messages that are written are important to see. However, these can be turned off setting this to `FALSE`

`unloadNamespaces` Default: `TRUE`. (ADVANCED USE) `Require` will attempt to detach and unload packages that conflict with the requested package installing via `Require`. This can be complicated, resulting in broken states that can only be recovered by restarting R. Default is to attempt to do this. `FALSE` will not attempt to do this. User must deal with inability to install packages due to package already being loaded.

`usePak` Default: `FALSE`. Should `pak` be used to resolve package dependencies and installation.

`verbose` Default: `0`. During a `Require`, there is a lot of information collected and used. With `verbose` set to 1 or 2, more of this information will be reported as an attribute attached to the return object of `Require`. This may help diagnosing problems.

---

rversions

*R versions*


---

**Description**

Reference table of R versions and their release dates (2018 and later).

**Usage**

```
rversions
```

**Format**

An object of class `data.frame` with 21 rows and 2 columns.

**Details**

Update this as needed using `rversions::r_versions()`:

```
# install.packages("rversions")
v = rversions::r_versions()
keep = which(as.Date(v$date, format = "
dput(v[keep, c("version", "date")])
```

---

setLibPaths

*Set .libPaths*


---

**Description**

This will set the `.libPaths()` by either adding a new path to it if `standAlone = FALSE`, or will concatenate `c(libPath, tail(.libPaths(), 1))` if `standAlone = TRUE`. Currently, the default is to make this new `.libPaths()` "sticky", meaning it becomes associated with the current directory even through a restart of R. It does this by adding and/updating the `‘.Rprofile’` file in the current directory. If this current directory is a project, then the project will have the new `.libPaths()` associated with it, even through an R restart.

**Usage**

```
setLibPaths(
  libPaths,
  standAlone = TRUE,
  updateRprofile = getOption("Require.updateRprofile", FALSE),
  exact = FALSE,
  verbose = getOption("Require.verbose")
)
```

**Arguments**

libPaths	A new path to append to, or replace all existing user components of <code>.libPaths()</code>
standAlone	Logical. If TRUE, all packages will be installed to and loaded from the libPaths only. NOTE: If TRUE, THIS WILL CHANGE THE USER'S <code>.libPaths()</code> , similar to e.g., the checkpoint package. If FALSE, then libPath will be prepended to <code>.libPaths()</code> during the Require call, resulting in shared packages, i.e., it will include the user's default package folder(s). This can be create dramatically faster installs if the user has a substantial number of the packages already in their personal library. Default FALSE to minimize package installing.
updateRprofile	Logical or Character string. If TRUE, then this function will put several lines of code in the current directory's <code>.Rprofile</code> file setting up the package libraries for this and future sessions. If a character string, then this should be the path to an <code>.Rprofile</code> file. To reset back to normal, run <code>setLibPaths()</code> without a libPath. Default: <code>getOption("Require.updateRprofile", FALSE)</code> , meaning FALSE, but it can be set with an option or within a single call.
exact	Logical. This function will automatically append the R version number to the libPaths to maintain separate R package libraries for each R version on the system. There are some cases where this behaviour is not desirable. Set exact to TRUE to override this automatic appending and use the exact, unaltered libPaths. Default is FALSE
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.

**Details**

This details of this code were modified from <https://github.com/milesmcbain>. A different, likely non-approved by CRAN approach that also works is here: <https://stackoverflow.com/a/36873741/3890027>.

**Value**

The main point of this function is to set `.libPaths()`, which will be changed as a side effect of this function. As when setting options, this will return the previous state of `.libPaths()` allowing the user to reset easily.

**Examples**

```
## Not run:
origDir <- setwd(tempdir())
td <- tempdir()
setLibPaths(td) # set a new R package library locally
setLibPaths() # reset it to original
setwd(origDir)
# Using standAlone = FALSE means that newly installed packages will be installed
# in the new package library, but loading packages can come from any of the ones
# listed in .libPaths()
setLibPaths("~/newProjectLib", standAlone = FALSE) # will have 2 or more paths
```

```
# Can restart R, and changes will stay

# remove the custom .libPaths()
Require::setLibPaths() # reset to previous; remove from .Rprofile because libPath arg is empty

## End(Not run)
```

---

```
setLinuxBinaryRepo      Setup for binary Linux repositories
```

---

### Description

Enable use of binary package builds for Linux from the Rstudio Package Manager repo. This will set the repos option, affecting the current R session.

### Usage

```
setLinuxBinaryRepo(binaryLinux = "https://packagemanager.rstudio.com/")
```

### Arguments

binaryLinux      A CRAN repository serving binary Linux packages.

---

```
setup                  Setup a project library, cache, options
```

---

### Description

This can be placed as the first line of any/all scripts and it will be create a reproducible, self-contained project with R packages. Some of these have direct relationships with RequireOptions and arguments in setLibPaths and Require.

### Usage

```
setup(
  RPackageFolders = getOption("Require.RPackageFolders", "R"),
  RPackageCache = getOptionRPackageCache(),
  buildBinaries = getOption("Require.buildBinaries", TRUE),
  standAlone = getOption("Require.standAlone", TRUE),
  verbose = getOption("Require.verbose")
)

setupOff(removePackages = FALSE, verbose = getOption("Require.verbose"))
```

**Arguments**

RPackageFolders	One or more folders where R packages are installed to and loaded from. In the case of more than one folder provided, installation will only happen in the first one.
RPackageCache	See ?RequireOptions.
buildBinaries	See ?RequireOptions.
standAlone	Logical. If TRUE, all packages will be installed to and loaded from the libPaths only. NOTE: If TRUE, THIS WILL CHANGE THE USER'S .libPaths(), similar to e.g., the checkpoint package. If FALSE, then libPath will be prepended to .libPaths() during the Require call, resulting in shared packages, i.e., it will include the user's default package folder(s). This can be create dramatically faster installs if the user has a substantial number of the packages already in their personal library. Default FALSE to minimize package installing.
verbose	Numeric or logical indicating how verbose should the function be. If -1 or less, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more.
removePackages	Logical. If TRUE, then all packages that were installed in the custom library will be deleted when setupOff is run. The default is FALSE, and when TRUE is selected, and it is an interactive session, the user will be prompted to confirm deletions.

**Examples**

```
## Not run:
# Place this as the first line of a project
Require::setup()

# To turn it off and return to normal
Require::setupOff()

## End(Not run)
```

---

tempdir2

*Make a temporary (sub-)directory*


---

**Description**

Create a temporary subdirectory in .RequireTempPath(), or a temporary file in that temporary subdirectory.

**Usage**

```
tempdir2(sub = "", tempdir = getOption("Require.tempPath", .RequireTempPath()))
```



**Arguments**

sub	Character string, length 1. Can be a result of <code>file.path("smth", "smth2")</code> for nested temporary sub directories.
tempdir	Optional character string where the temporary dir should be placed. Defaults to <code>.RequireTempPath()</code>

**See Also**

[tempfile2\(\)](#)

---

tempfile2	<i>Make a temporary subfile in a temporary (sub-)directory</i>
-----------	--

---

**Description**

Make a temporary subfile in a temporary (sub-)directory

**Usage**

```
tempfile2(
  sub = "",
  tempdir = getOption("Require.tempPath", .RequireTempPath()),
  ...
)
```

**Arguments**

sub	Character string, length 1. Can be a result of <code>file.path("smth", "smth2")</code> for nested temporary sub directories.
tempdir	Optional character string where the temporary dir should be placed. Defaults to <code>.RequireTempPath()</code>
...	passed to <code>tempfile</code> , e.g., <code>fileext</code>

**See Also**

[tempdir2\(\)](#)

---

trimVersionNumber	<i>Trim version number off a compound package name</i>
-------------------	--

---

**Description**

The resulting string(s) will have only name (including github.com repository if it exists).

**Usage**

```
trimVersionNumber(pkgs)
```

**Arguments**

pkgs                    A character string vector of packages with or without GitHub path or versions

**See Also**

[extractPkgName\(\)](#)

**Examples**

```
trimVersionNumber("PredictiveEcology/Require (<=0.0.1)")
```

---

useLinuxSourceCache	<i>Use cache for R package compilation</i>
---------------------	--

---

**Description**

Simple `ccache` configuration for compiling R packages on Linux, based on [http://dirk.eddelbuettel.com/blog/2017/11/27/#011\\_faster\\_package\\_installation\\_one](http://dirk.eddelbuettel.com/blog/2017/11/27/#011_faster_package_installation_one).

**Usage**

```
useLinuxSourceCache(overwrite = FALSE)
```

**Arguments**

overwrite            Logical. Should the existing configuration be overwritten? For safety, a backup copy of the old configuration is created.

**Value**

Invoked for the side effect of copying files needed to configure `ccache` for R packages.

**Note**

This is typically run once per user, per machine to configure the cache.

**Author(s)**

Dirk Eddelbuettel and Alex Chubaty

**Examples**

```
## Not run:  
useLinuxSourceCache()
```

```
## End(Not run)
```

# Index

- \* **datasets**
  - rversions, [29](#)
- archiveVersionsAvailable
  - (getPkgVersions), [12](#)
- base::Reduce, [16](#)
- checkPath, [8](#)
- checkPath, character, logical-method
  - (checkPath), [8](#)
- checkPath, character, missing-method
  - (checkPath), [8](#)
- checkPath, missing, ANY-method
  - (checkPath), [8](#)
- checkPath, NULL, ANY-method (checkPath), [8](#)
- DESCRIPTIONFileOtherV
  - (DESCRIPTIONFileVersionV), [9](#)
- DESCRIPTIONFileVersionV, [9](#)
- detachAll, [10](#)
- dir.create(), [9](#)
- doInstalls (getPkgVersions), [12](#)
- doLoading (getPkgVersions), [12](#)
- extractInequality (extractPkgName), [11](#)
- extractPkgGitHub (extractPkgName), [11](#)
- extractPkgName, [11](#)
- extractPkgName(), [34](#)
- extractVersionNumber (extractPkgName),  
[11](#)
- file.exists(), [8](#), [9](#)
- getAvailable (getPkgVersions), [12](#)
- getGitHubDESCRIPTION
  - (DESCRIPTIONFileVersionV), [9](#)
- getOptionRPackageCache, [12](#)
- getPkgVersions, [12](#)
- getRequireOptions (RequireOptions), [27](#)
- installFrom (getPkgVersions), [12](#)
- installGitHub (parseGitHub), [19](#)
- installGitHubPackage
  - (installGithubPackage), [14](#)
- installGithubPackage, [14](#)
- invertList, [15](#)
- linkOrCopy, [15](#)
- messageDF, [16](#)
- messageVerbose (messageDF), [16](#)
- modifyList2, [16](#)
- normalizePath(), [8](#)
- normPath, [17](#)
- normPath(), [8](#)
- normPath, character-method (normPath), [17](#)
- normPath, list-method (normPath), [17](#)
- normPath, logical-method (normPath), [17](#)
- normPath, missing-method (normPath), [17](#)
- normPath, NULL-method (normPath), [17](#)
- paddedFloatToChar, [18](#)
- parseGitHub, [19](#)
- pkgDep, [20](#)
- pkgDep2 (pkgDep), [20](#)
- pkgDepIfDepRemoved, [23](#)
- pkgDepTopoSort (pkgDep), [20](#)
- pkgSnapshot, [24](#)
- pkgSnapshot(), [5](#)
- rCurrentVersion, [26](#)
- Require (Require-package), [2](#)
- Require-package, [2](#)
- RequireCacheDir, [27](#)
- RequireOptions, [27](#)
- RequirePkgCacheDir (RequireCacheDir), [27](#)
- rversions, [29](#)
- setLibPaths, [29](#)
- setLinuxBinaryRepo, [31](#)

setup, [31](#)  
setupOff (setup), [31](#)

tempdir2, [32](#)  
tempdir2(), [33](#)  
tempfile2, [33](#)  
tempfile2(), [33](#)  
trimVersionNumber, [34](#)  
trimVersionNumber(), [11](#)

useLinuxSourceCache, [34](#)  
utils::available.packages, [4](#), [10](#), [13](#), [22](#),  
[25](#)  
utils::modifyList, [16](#)